ASEN 6519 Final Project REPORT

Maneul 'Ricky' Puyana

TOTAL POINTS

(not graded)

QUESTION 1

1 Final Project Report

- 0 pts Correct

AlphaZeroNano
An AlphaZero Algorithm for 3x3 Chess

1st Manuel 'Ricky' Puyana
Aerospace Engineering Sciences
Colorado Boulder University
manuel.puyana@colorado.edu

Editor ChatGPT 3.5
Generative Text AI Model
OpenAI

***Abstract*—This paper focuses on implementing an AlphaZero-style algorithm for 3x3 chess, leveraging the solved nature of the game as an ideal testbed. Positioned to address the scarcity of detailed information for independent AlphaZero-style algorithm implementations, this work provides a comprehensive framework tailored for smaller boards. While 3x3 chess serves as a practical proving ground, the ultimate goal is to contribute to the future independent implementation of AlphaZero-style algorithms for full-scale chess. Despite observed performance limitations, the study reveals robust potential within the established framework.**

I. Introduction

The integration of Machine Learning, Computer Vision, and Artificial Intelligence into chess-playing systems is a well-established practice with roots dating back to the creation of the first fully automated Chess AI by an IBM engineer in 1957. Over time, these implementations have evolved to the extent that even the most proficient human players find it challenging to surpass the capabilities of the highest-rated Chess AIs.

Developing an AI for chess requires meticulous consideration, especially given the exponential growth in the number of possible moves to be evaluated as the AI engages earlier in the game. Additionally, managing the depth of search algorithms becomes crucial, as the computational complexity increases exponentially with each node explored. Prominent algorithms, such as DeepMind's *AlphaZero* [9], employ Neural Networks trained for use in board evaluation and move planning in tree search algorithms.

In this work, the focus is narrowed to the practical implementation of AlphaZero on a 3x3 chess board (AlphaZero-Nano). This reduction is motivated by the unique characteristic of 3x3 chess, where all positions are solved, and to reduce the state and action spaces. This deliberate simplification allows the addressing and navigation of the myriad challenges associated with practically implementing the AlphaZero algorithm.

Furthermore, this paper will leverage and pull from one of the authors' previous works, titled "The Application of Minimax Algorithms to Solve Mate in X Chess Puzzles," which contains an in-depth exploration of previous works in Chess AI and 3x3 Chess.

II. Previous Work

A. AlphaZero [9]

The inception of this project was profoundly influenced by the groundbreaking work of AlphaZero, a pivotal force in shaping the landscape of chess-playing AI. AlphaZero's innovative use of Monte-Carlo Tree Search (MCTS) coupled with a neural network for evaluating board states marked a paradigm shift in the realm of game-playing algorithms.

In AlphaZero, MCTS serves as the guiding force for decision-making, providing a robust framework for exploring potential moves and refining strategies. The integration of a neural network at the evaluation stage distinguishes AlphaZero from conventional approaches. Instead of relying on a rollout strategy for leaf nodes, AlphaZero employs the predictive power of its neural network to assess board positions.

This departure from traditional methods significantly enhances the efficiency and depth of the search process. The neural network, trained through an extensive self-play regimen, generates policy distributions and value estimates, aiding AlphaZero in making informed and strategic decisions during gameplay.

Inspired by the success and transformative nature of AlphaZero's approach, this project endeavors to adapt and implement similar methodologies, specifically tailored for 3x3 chess. The unique characteristics of this solved game serve as a fertile ground for exploring the practical implications of leveraging MCTS and neural network evaluations, mirroring the success of AlphaZero while addressing implementation challenges in the reduced board space.

B. Chess AI: Competing Paradigms for Machine Intelligence [7]

This paper, authored by the creators of *Deep Blue*, delves into the intricate realm of Chess endgames, recognizing the deterministic nature of most endgame positions. The ability to memorize or identify similar structures becomes crucial in solving these optimal solutions.

The study focuses on "Plaskett's Puzzle," an iconic endgame, comparing the performance of *LCZero* (a crowd-sourced version of *AlphaZero*) against *StockFish*. The puzzle accentuates the strengths and weaknesses of each algorithm, particularly highlighting the impact of the first move, which,

despite not providing the highest evaluation at a relatively low MCTS search depth, significantly influences subsequent outcomes.

The paper further explores the scenario where each algorithm is given the optimal first move in the puzzle. Here, *LCZero* successfully identifies the winning line with only 5.5 million nodes searched, while *Stockfish* requires 500 million nodes to reach the same conclusion.

In conclusion, the paper underscores a fundamental challenge in AI advancements: each algorithm exhibits distinct strengths and weaknesses. Addressing deviations from nominal behavior for edge cases becomes increasingly complex due to their vast and diverse nature, highlighting the ongoing complexities in AI development.

The determinist nature of most endgame positions was the impetus to focus AlphaZeroNano on 3x3 chess, where every position is solved. Similar to the paper's exploration of Chess endgames, the unique characteristic of 3x3 chess allows for the straightforward determination of accuracy in all AlphaZeroNano solutions.

C. BetaZero [11]

BetaZero stands as a significant advancement in addressing real-world planning problems, particularly in applications such as autonomous driving, carbon storage, and resource exploration. This algorithm models these challenges as partially observable Markov decision processes (POMDPs) and employs approximate methods to find solutions.

In contrast to fully observable domains, where learning approximations to replace heuristics has found success, BetaZero extends this concept to partially observed domains. The algorithm combines online Monte Carlo tree search with offline neural network approximations of the optimal policy and value function, addressing challenges specific to large-scale POMDPs.

While the crux of BetaZero's approach may not directly influence the development of AlphaZeroNano, its intricate details, particularly those related to parallel training, offer valuable insights. These details will inform the project's exploration of parallel training methodologies, contributing to a more nuanced and informed implementation in the context of 3x3 chess.

D. Deep Blue [6]

The "Deep Blue" paper from 2002 provides insights into the development and deployment of the *Deep Blue* AI, renowned for defeating Gary Kasparov in 1997. The paper offers a comprehensive overview of the hardware and software architecture of *Deep Blue*, emphasizing its primary focus on computational efficiency for chess.

Notably, *Deep Blue* achieved an impressive capability to evaluate up to 200 million positions per second, employing a Monte-Carlo Tree Search (MCTS) up to 40 layers deep each second. The design of the computer involved three layers: initial positions were computed by a set of chips, which were then passed to 'worker' computer chips for evaluating top-level moves. Finally, dedicated 'chess chips' delved more deeply into the MCTS, contributing to the AI's formidable processing power.

The paper concludes with an analysis of the games played between *Deep Blue* and Gary Kasparov, highlighting tactical mistakes and advantages observed at various stages of the match.

E. DeepChess: End-to-End Deep neural network for Automatic Learning in Chess [8]

DeepChess is a pioneering Deep neural network (DNN) for learning chess from scratch. The process is divided into three stages: training the DNN to evaluate positions, having it select optimal moves, and compressing it for rapid computation.

To teach the DNN chess evaluation, it is exposed to positions without prior knowledge of the rules. The DNN learns patterns over time, considering factors like passed pawns and King positions. The DNN is then trained on positions resulting in either a White or Black victory, creating a large training set for future encounters. Notably, providing draws did not yield benefits. In evaluation tests against *Falcon*, an established chess AI with over 100 hand-tuned hyperparameters, *DeepChess* performed comparably or superiorly, despite being trained without prior knowledge of chess rules.

This work marks a significant achievement, presenting the first end-to-end application of machine learning to a computer chess algorithm. *DeepChess* exhibits an aggressive playing style, resembling grandmasters, demonstrating the effectiveness of its training structure.

III. Problem Formulation

Chess unfolds as a turn-taking duel between two players, White and Black, aiming to strategically move pieces until one achieves checkmate. A comprehensive guide to the rules of chess is available on Wikipedia's "Rules of Chess" page [4].

The expansive state space of chess encompasses a staggering 10^{40} possible board states, with the action space representing all legal moves from any given board state; however, the rules of chess dictate a subset of permissible moves. Transitions within the game are deterministic, allowing players to move any piece directly to their desired square, provided the move adheres to the rules. Details on the reward function employed in AlphaZeroNano are explored in Section VI.

3x3 Chess, a variant of the classic game, differs only in its play on a condensed 3x3 board rather than the standard 8x8 board. While 3x3 Chess is not widely played and is primarily used for analytical purposes, it offers a unique perspective. The game lacks fixed starting positions, encompassing all legal configurations of standard chess pieces on a 3x3 board. In this work, we specifically focus on Major Piece 3x3 Chess (MP3x3 Chess), where each player is limited to at most one Rook and one Queen. Importantly, 3x3 Chess is a solved game, with every position having a deterministic evaluation of either a White win in X moves, a Black win in X moves, or a draw.

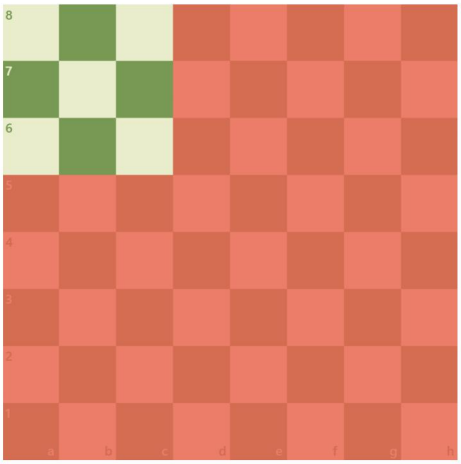


Fig. 1: Upper nine squares of chess board to use for 3x3 board

IV. Chess Environment

The Python chess library was utilized to create the 3x3 chess environment [1]. The Python chess library has functionality for setting up a full chess board, checking for legal moves, making moves, capturing pieces, promoting pieces, checking for checks and checkmates, and checking for draws; it contains all the functionality necessary to play out a game of chess. To create the 3x3 chess board. The following Python chess chess board methods were overloaded to abide by the rules of 3x3 chess in the upper-left nine squares of the board,

- Board.legal_moves()
  - Give all the legal moves for the current player
- Board.is_game_over()
  - Check if the current board position is checkmate or a draw
- Board.is_checkmate()
  - Check is checkmate is present on the current board

For example, if Python chess is used without the additional 3x3 constraints imposed, the position in Fig. 2 (Black to move), the king has three legal moves, so the position is not checkmate. In Fig. 3, the 3x3 constraints are imposed. After constraints, the overloaded methods show that Black has no legal moves, the game is over, and Black is checkmated.

V. Chess Notation

In chess, all squares are given a two-letter identifier. This identifier is generated based on a grid where the ranks of the board (rows) are given numbers 1-8 starting in the bottom left, and the files (columns) are given letters a-h starting in the bottom left. For example, the square highlighted in Fig. 4 is f6.

In this work Coordinate Algebraic Notation (CAN) will be used for moves. In this notation, a move is described by 4 characters, the first two signifying the starting square of the piece and the second two signifying the ending square of the piece. For example, in Fig. 5 White has 4 legal moves on the 3x3 board,

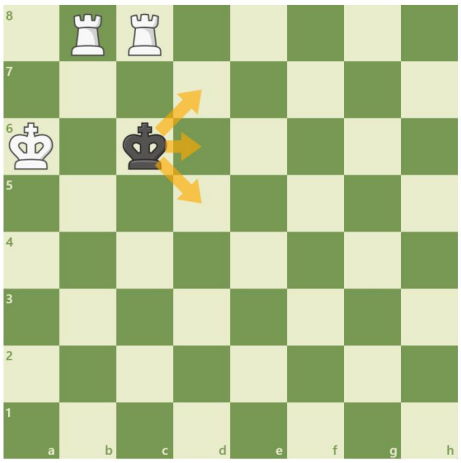


Fig. 2: Before 3x3 constraints, Checkmate Example

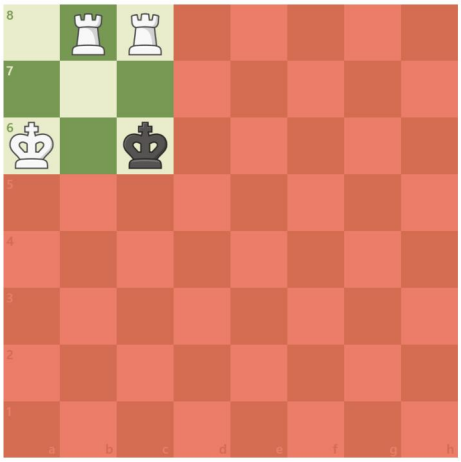


Fig. 3: After 3x3 constraints, Checkmate Example

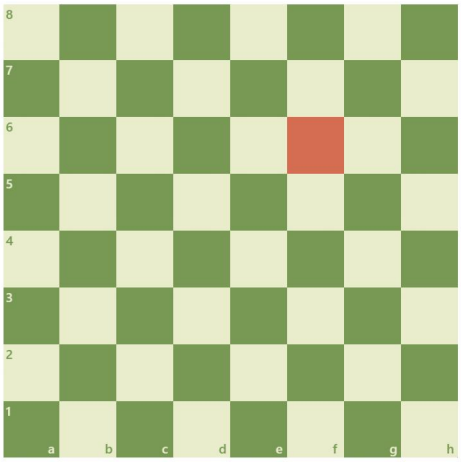


Fig. 4: f6 square

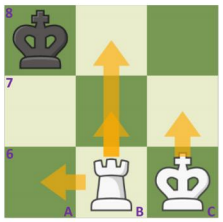


Fig. 5: White legal moves

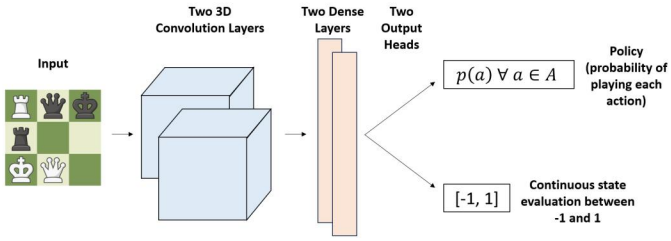


Fig. 6: AlphaZeroNano Neural Network Architecture

- Rook Moves: b6b7, b6b8, b6a6
- King Moves: c6c7

VI. AlphaZeroNano Algorithm

AlphaZeroNano, like AlphaZero [9], is a minimax-style MCTS algorithm that uses modified Upper Confidence Tree (UCT) and evaluates leaf nodes with a Neural Network evaluation of the state.

A. Neural Network Architecture

AlphaZeroNano employs a neural network architecture, implemented with TensorFlow [12], designed specifically for MP3x3 chess. The network takes a board encoding as input and has three key components:

1) **Convolutional Layers:** The initial layers consist of two 3D convolutional layers, capturing intricate spatial relationships within the 3x3 board configuration. These layers excel at learning hierarchical features critical for effective position evaluation.
2) **Dense Layers:** Following the convolutional layers, two dense layers are employed, further extracting abstract features from the learned representations. These layers contribute to the network's ability to discern complex patterns and strategic nuances in the limited space of 3x3 chess.
3) **Output Heads:** AlphaZeroNano integrates two output heads, each serving a distinct purpose.

1) State Evaluation (Value) Head: One output head focuses on continuous state evaluation, yielding a value between -1 and 1. This value encapsulates the network's assessment of the current position. A score of 1 signifies the position is a win for White if perfect play is assumed, -1 denotes the same for Black, and 0 indicates a Draw. This continuous evaluation aids in understanding the positional dynamics and strategic balance. It also negates the need for a specific reward function to be constructed, as this evaluation replaces leaf node rollouts in the tree search. When a terminal state is reached in the search tree, an evaluation equal to that of the terminal state reached is given definitively.

2) Policy Head: The second output head is dedicated to estimating the probability of playing each action. It generates probabilities for all possible actions, encompassing both legal and illegal moves. To ensure adherence to the rules of 3x3 chess, the output is masked and normalized, restricting it to a valid probability distribution over legal moves.

3) State Encoding: The board state is input to the neural network as a 3D tensor. Each 2D 'slice' of this tensor corresponds to a 3x3 array representation of the board for a specific piece type (Fig. 7). In this encoding scheme, the presence of pieces is denoted by numerical values: empty spaces are represented as 0s, White pieces as 1s, and Black pieces as -1s.

This 3D tensor encapsulates a comprehensive snapshot of the current game configuration, allowing the neural network to capture the distribution and arrangement of pieces on the 3x3 chess board. The use of distinct numerical values facilitates the differentiation between empty spaces and the pieces belonging to White or Black players. The encoding methodology is tailored to provide the neural network with a structured and informative input, enabling it to discern patterns, relationships, and positional intricacies crucial for effective decision-making.

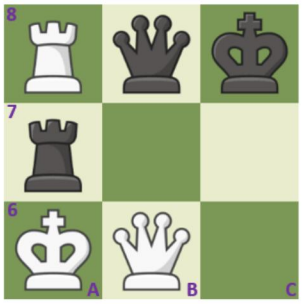
Additionally, to maintain consistency and streamline the learning process, the neural network estimates all positions from the perspective of "White to move." This strategic choice ensures uniformity during both training and inference, simplifying the network's decision-making process.

When it is Black's turn, the board is flipped along the 7^{th} rank, and the White/Black encoding is swapped (the tensor is multiplied by -1). This represents Black's turn as a "White to move" position. Special care is taken with the network's policy head output to map move probabilities from this manufactured "White to move" scenario back to Black's actual move options.

4) Action Space Encoding: In AlphaZeroNano, the action space is encoded as a structured 2D array (Fig. 8), providing a concise representation of legal moves within the game. This encoding scheme employs a straightforward matrix where each column signifies a specific piece type, and each row denotes a move paired with a direction.

This 2D array effectively encapsulates the potential moves available for each piece type, forming a comprehensive catalog of the permissible actions within the 3x3 chess environment. Though all pieces are treated as if they were a Queen in this array, the simplicity of the encoding enables it to be easily passed as neural network output.

This action encoding serves a dual purpose within the AlphaZeroNano framework. While used to encode move probabilities from the neural network's policy head, the formulation is also used to encode legal moves and perform illegal move masking. This adaptability highlights the elegance of the ap-



Rook			Queen			King		
1	0	0	0	-1	0	0	0	-1
-1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0	0

Fig. 7: 2D Slices of State Encoding

	Rook	Queen	King
North 1	0	0	0
North 2	0	0	0
South 1	1	0	0
South 2	0	0	0
East 1	0	0	0
East 2	0	0	0
West 1	0	0	0
West 2	0	0	0
NE 1	0	0	0
NE 2	0	0	0
SE 1	0	0	0
SE 2	0	0	0
NW 1	0	1	0
NW 2	0	0	0
SW 1	0	0	0
SW 2	0	0	0

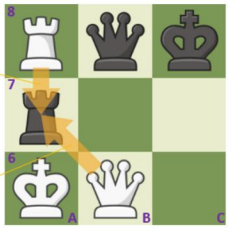


Fig. 8: Legal Moves Encoding For White

proach, as it streamlines the integration of move probabilities into the broader learning framework.

When the neural network is used to evaluate a position for Black, the North-South, NE-SE, and NW-SW rows are swapped to convert from the manufacture "White to move" position back to Black's legal moves.

B. Tree Search Algorithm

The tree search algorithm employed by AlphaZeroNano departs from standard MDP Upper Confidence Tree (UCT) Monte Carlo Tree Search (MCTS) in two ways: Minimax and Policy Estimation Influenced UCT.

1) Minimax MCTS: Given that 3x3 chess is fundamentally a game rather than a Markov Decision Process (MDP), the traditional MCTS optimization framework is not directly applicable. Instead of the conventional approach of maximizing the quantity $Q(s,a)+UCT_{bonus}(s,a)$ for each selected action, AlphaZeroNano employs a Minimax-inspired strategy.

When selecting an action as White, the algorithm maximizes $Q(s,a) + UCT_{bonus}(s,a)$ and when selecting an action as

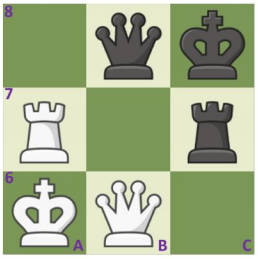


Fig. 9: First Starting Position For Self-Play, White to Move

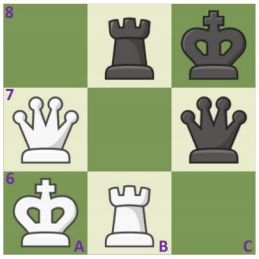


Fig. 10: Second Starting Position For Self-Play, Black to Move

Black, it minimizes $Q(s,a) - UCT_{bonus}(s,a)$ This modification ensures that each simulated action is oriented towards maximizing the advantage for the action taker and minimizing the advantage for the opponent, aligning with the principles of zero-sum games.

2) Policy Estimation Influenced UCT (PUCT): To incorporate the neural network's policy estimation into the exploration strategy, AlphaZeroNano introduces a modified UCT expression. Unlike traditional UCT, PUCT considers the policy estimation $p(a|s)$ when selecting an action during the search process. This adaptation allows the neural network to guide exploration, leveraging its learned insights to influence the selection of promising moves.

$$UCT_{bonus}(s,a) = p(a|s)c\sqrt{\frac{\log N(s)}{N(s,a)}} \tag{1}$$

VII. Neural Network Training

Similar to AlphaZero [9], AlphaZeroNano undergoes training through a process of self-play. The algorithm engages in games against itself, initiating play from two starting positions Fig. 9 and Fig. 10, both of which are theoretically drawn. During this self-play, 'experience tuples' are gathered as each move is played, which contain:

- An encoding of the board state.
- An encoding of the probability associated with each legal move, as determined by the tree search algorithm.
- The eventual outcome of the game, indicated by a binary label: 1 for a win by White, -1 for a win by Black, or 0 for a draw.

Effectively, the neural network learns a function that maps a given board state to the outcomes predicted by the tree search

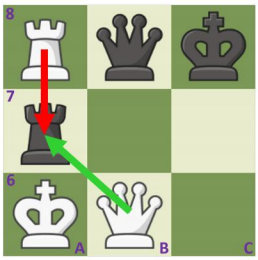


Fig. 11: White to Move and Win Evaluation Position With Legal Moves Colored as Described in Section VIII-B

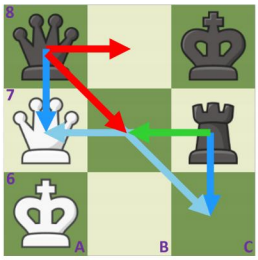


Fig. 12: Black to Move and Win Evaluation Position With Legal Moves Colored as Described in Section VIII-B

algorithm. These experience tuples are stored in an experience buffer, forming the basis for the iterative training process, reminiscent of the principles of Deep Q-Learning.

AlphaZeroNano was trained with up to 100 parallel self-play processes in between training steps [11]. The overhead associated with starting parallel processes in python, combined with the relatively short duration of 3x3 games, led to approximately 10-30 parallel processes being utilized concurrently at any given time.

Two distinct training setups were tested, each with varying buffer sizes and batch sizes:

1) **Buffer size:** 1×10^5, **Batch size:** 1×10^3, trained for approximately 50 hours.
2) **Buffer size:** 1×10^6, **Batch size:** 1×10^4, trained for approximately 22 hours.

These training configurations were explored to assess the impact of buffer size and batch size on the learning efficiency and overall performance of AlphaZeroNano.

VIII. AlphaZeroNano Evaluation

Training the neural network is the most crucial portion of employing any AlphaZero-style algorithm. Due to 3x3 Chess being solved from any position, rather than having AlphaZeroNano play against a benchmark opponent, its neural network was simply evaluated on four boards. These boards encompassed the two starting positions (Figs 9 and 10) which are theoretical draws, as well as positions resulting in each a win for White (Fig. 11) and Black (Fig. 12). The evaluation process involved tracking the neural network's assessments over the course of self-play games.

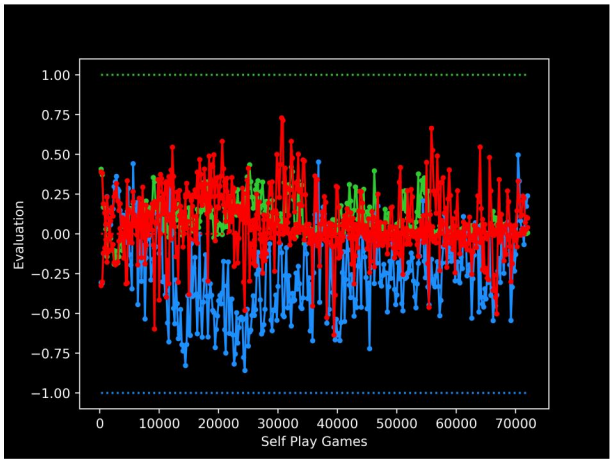


Fig. 13: Value Head Evaluation, Small Buffer and Batch Size

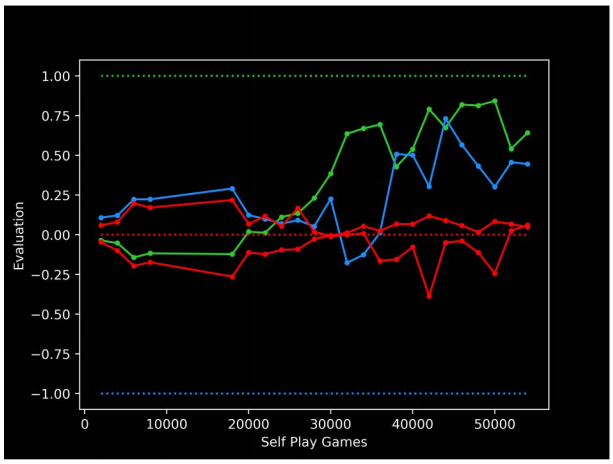


Fig. 14: Value Head Evaluation, Large Buffer and Batch Size

A. Value Head Evaluation

The value estimate of each of the four boards was tracked (Figs 13 and 14). Draw boards were plotted in red, the win for White in green, and the win for Black in blue. The desired evaluation for each board is plotted with a dotted line.

In the value head training, some promise is evident, but significant room for improvement remains. In both training attempts, the network demonstrates an ability to differentiate between the three distinct position types around the midway point of its training; however, there is a noticeable loss of clarity regarding the winning strategy for the Black position as training progresses. It's worth noting that non-monotonic training behavior is not uncommon in deep learning, and this observation might be an artifact of insufficient training duration. Further insights into potential enhancements for AlphaZeroNano's performance are discussed in Section IX.

B. Policy Head Evaluation

For the positions that are wins for White and Black, the probability of playing each action was tracked (Figs 15, 16, 17, and 18). Red moves correspond to moves that will result

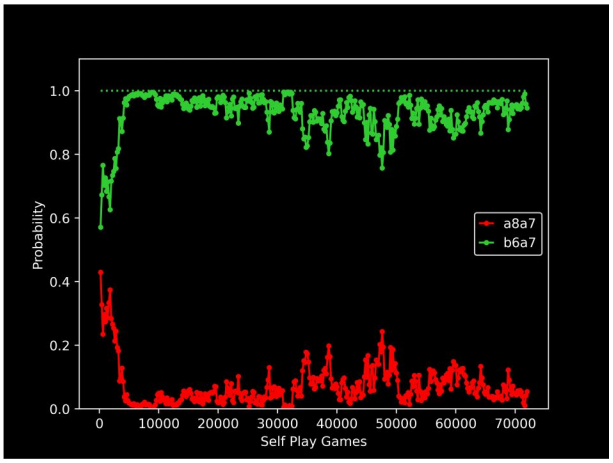


Fig. 15: Policy Head Evaluation, Small Buffer and Batch Size, White to Move and Win

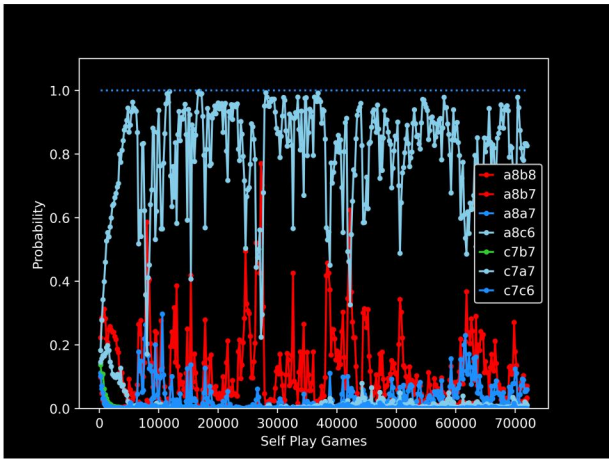


Fig. 16: Policy Head Evaluation, Small Buffer and Batch Size, Black to Move and Win

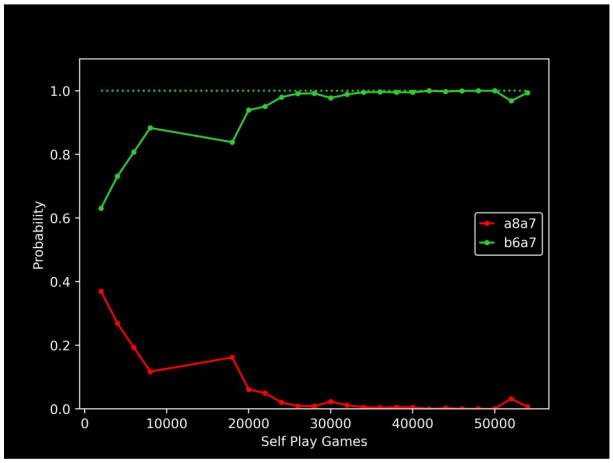


Fig. 17: Policy Head Evaluation, Large Buffer and Batch Size, White to Move and Win

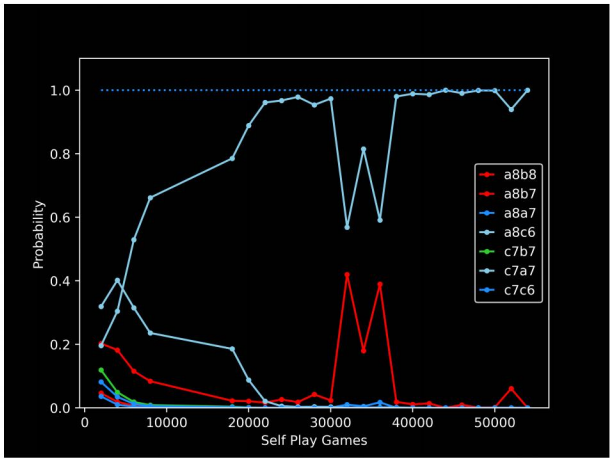


Fig. 18: Policy Head Evaluation, Large Buffer and Batch Size, Black to Move and Win

in a draw if perfect play is observed after the move, green moves will result in a win for White, light blue moves will result in a win for Black, and blue moves are an immediate win for Black.

The training results for the policy head exhibit more promise compared to the value head. In each position, the neural network quickly selects a correct move with a high probability. It's worth noting that in the scenario where Black has the opportunity to move and win, the network consistently opts for a move that secures a Black victory, but not the checkmate-in-one. This tendency may stem from the absence of a discount factor in the tree search algorithm, causing AlphaZeroNano to lack a sense of urgency. While this choice doesn't compromise the accuracy of AlphaZero's moves, it contrasts with the behavior of some flagship Chess AIs that prioritize moves leading to the quickest possible victory.

IX. Issues and Possible Improvements

Optimizing the performance of AlphaZeroNano is crucial for achieving its full potential in 3x3 chess. Several avenues for improvement are worth exploring.

A. Extended Training Time

Consideration should be given to extending the duration of the training phase. A more extended training period might allow the neural network to converge more effectively, potentially refining its strategic understanding and decision-making capabilities; however, for extended training time to be a practical next step, the framework laid out would require significant optimization.

B. Diverse Neural Network Architectures

Experimentation with different neural network architectures could yield insights into more effective configurations tailored to the unique challenges of 3x3 chess. Exploring variations in layer structures, activation functions, or network depths may enhance the network's learning capacity.

C. Adjusting UCT Exploration Parameters

Fine-tuning the parameters governing the Upper Confidence Tree (UCT) exploration strategy may influence the balance between exploration and exploitation during the tree search. Modifying these parameters could lead to more efficient and informed decision-making.

D. Reducing Dependency on python-chess

The current dependence on the python-chess module introduces overhead and speed limitations. Creating a specialized chess module designed for AlphaZeroNano could streamline the process, reducing unnecessary computations and enhancing overall efficiency.

E. Transition to C++ or Julia

Considering a shift from Python to a language like C++ or Julia for the core implementation might offer performance benefits. These languages are known for their efficiency, and the transition could mitigate the overhead associated with certain tasks, contributing to faster execution and training.

This may become essential when exploring network architectures and UCT exploration parameters, as the ability to effectively evaluate each permutation is dependent on the efficiency of the training algorithm and the computational resources available.

X. Conclusion

This study has successfully established a comprehensive framework and methodology for the implementation of AlphaZero in the context of MP3x3 chess. The explicit design of the framework lays a foundation that can be extended to the complexities of full chess.

The neural network training, conducted through extensive self-play sessions, exhibited promising results. The network demonstrated the capacity to learn strategic nuances and make informed decisions; however, there is certainly room for improvement. The identified issues, ranging from training dynamics to dependencies on external modules, provide valuable insights into the intricacies of adapting such algorithms to specific scenarios.

In conclusion, this work contributes a robust independent blueprint for AlphaZero in chess. The encountered challenges serve as stepping stones for future refinements, urging the exploration of novel strategies and optimizations. As the algorithm evolves, addressing these challenges will be pivotal in unlocking the full capabilities of AlphaZero.

XI. Housekeeping

The author grants permission for this report to be posted publicly.

References

[1] ”Python Chess,” https://python-chess.readthedocs.io/en/latest/
[2] “3x3 Chess,” kirr.homeunix.org/3x3-chess/
[3] “Stockfish,” https://stockfishchess.org/
[4] ”Rules of Chess,” https://en.wikipedia.org/wiki/Rules_of_chess
[5] ”Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm” Silver D., Hubert T., Schrittwieser J.
[6] ”Deep Blue” Campbell M., Joseph Hoane Jr. A., Hsu F.
[7] ”Chess AI: Competing Paradigms for Machine Intelligence” Maharaj S., Polson N., Turk A.
[8] ”DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess” David O., Netanyahu N., Wolf L.
[9] David Silver et al. , ”A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. Science362,1140-1144(2018).
[10] ”A0C: Alpha Zero in Continuous Action Space” Moerland T., Broekens J. , Plaat A., Jonker C.
[11] ”BetaZero: Belief-State Planning for Long-Horizon POMDPs using Learned Approximations,” Robert J. Moss and Anthony Corso and Jef Caers and Mykel J. Kochenderfer
[12] https://www.tensorflow.org/

1 Final Project Report

- 0 pts Correct