Exploring Chess Variants with AlphaZero

Conner Hettinger
hetti031@morris.umn.edu
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

Abstract

AlphaZero is a reinforcement learning algorithm that is capable of learning chess, shogi, and go beyond human levels of play without any game data. In "Assessing game balance with alphazero: Exploring alternative rule sets in chess," Nenad Tomasev et al. has AlphaZero learn different versions of chess and develop its own high level strategies. They use this method to analyze 9 different variations of chess where they look to find how small changes in the original rule set can change the flow of the game. This paper aims to showcase a few of the 9 variants covered, how AlphaZero is capable of learning them, and how those variants influenced the flow of the game.

1 Introduction

In the realm of video games and board games there is a constant battle for developers between balancing a game to make it fun, fair, and competitive. One board game has been around for centuries and has been under the critical eyes of millions of players. That game is chess. Chess also happens to be the target of many engines that can play chess far beyond any human ability. With these resources, the dynamics of chess can be researched far beyond what is possible for newer board games, since most board games have not been around for that long and do not have computers designed to play it.

AlphaZero is a chess AI that can learn the rules of chess and surpass any human player by training on self-play games. This means we can see how an artificial intelligence develops its strategy to become the best player in the world at any version of chess, without help of humans. With this information, we can find how different rules can influence different types of play.

In this paper, section 2 provides background information that is useful to understand ideas about chess, what a chess variant is, and relevant machine learning algorithms. Section 3 discusses the methods used in the study. Finally, section 4 is the results and section 5 is the conclusion.

2 Background

This section provides the background on the topics that are required to understand what is happening in this research paper. First, section 2.1 gives some intermediate chess concepts that are expanded on later in the paper. Then, section 2.2 talks about the details of artificial intelligence approaches that are relevant to the research described in the rest of the paper.

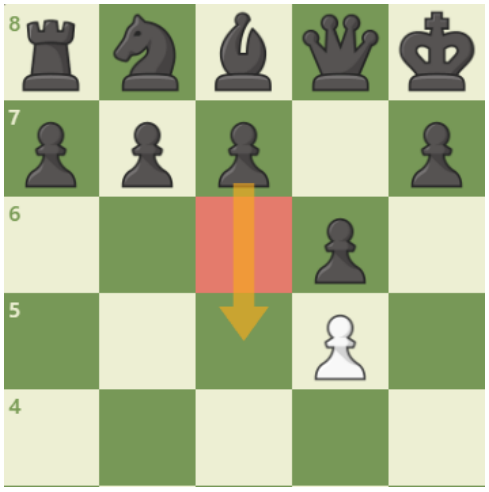


Figure 1. In this image, the orange arrow shows black making their pawn jump two squares and the red square shows where the white pawn can move to take the black pawn.

2.1 General chess concepts

2.1.1 Special moves. Most people know all of the basic moves that each piece can make in chess. However, for this study, it is helpful to know some of the special moves that can be made. In chess, there are a few moves that can only happen once in specific scenarios. One of those special moves has to do with how pawns can take other pawns. Typically, when a pawn takes a piece, the opposing piece has to be in a square diagonally one space in front of the pawn. However, if an opposing pawn passes a pawn in a single move, then the pawn can capture the opposing pawn as if it were in a square diagonally in front it. This is called *En Passant* and is the only move in the game where a piece captures another piece by moving to an empty square in a position relative to the opposing piece [4].

2.1.2 Pawn Structure. Pawns are not typically regarded as the most powerful piece in chess. It is true, however, that pawns are essential to the flow of the game and the evaluation of any given position. Pawn structure refers to the position of a player's pawns, not taking into account any other piece. Due to the lack of mobility of pawns, there are

not a lot of variations of how pawn structures look in the beginning of the game. Regardless of variability, the pawn structure provides a lot of information about the state of the game. For example, if there are a lot of pawns stuck in the middle of the board, the game is considered to be closed, which means there isn't a lot of room for pieces to jump across the board. In a closed state, we might determine that the knight is a more valuable piece than the bishop since knights are capable of hopping over other pieces and vice versa if the position is open. Pawn structure also heavily influences what kind of strategies are available to either player. For example, if the pawn structure is weak around an opponent's king, it might be easier to initiate an attack. [3]

2.1.3 Wins, losses, and draws. Chess is a game where draws can be more common than victories depending on the level of play. Qiyu Zhou in [11], has found that high level games typically end in a draw. The database used in this article has only high quality games, due to most recorded games being part of major tournaments like national championships, world championships and qualifiers. From this database, out of 78,468 games played in the past 47 years, 53% of games have ended in a draw. 28.9% of games are won by white and 18.0% of games are won by black. [11] From this information it is easy to see that at high levels of play draws are common. As it turns out, the higher the ranking the higher the amount of draws. This can be seen at the level of artificial intelligence. For example, when AlphaZero was pitted against another chess engine, Stockfish in [9], 639 games out of a 1000 that ended in a draw.

2.2 Chess artificial intelligence

2.2.1 Monte Carlo Tree Search. One of the main parts of AlphaZero is an algorithm called Monte Carlo Tree Search. The Monte Carlo Tree Search is a family of algorithms that relies on two concepts. First it needs to be able to guess the value of any given action from the current state of the game that it is looking at. Second it needs to be able to use the estimated values to choose actions based on some policy, or strategy. The tree looks like a normal tree structure where it starts at a root node that has edges that point downward to child nodes and those child nodes have child nodes. Each node represents a specific position in chess and each edge represents an action that can be taken from that position. For example, the root node could be the standard starting position in chess and an edge could be moving a pawn forward. Each layer in the tree also represents a players turn. So after an action happens the next position has a list of edges that corresponds to the other player's possible moves. [1]

There are four main steps to generate the tree in the MCTS: selection, expansion, simulation, and back propagation. Selection is a recursively applied function that uses some policy to select child nodes until it reaches a node that is expandable. The node is considered expandable if it doesn't represent

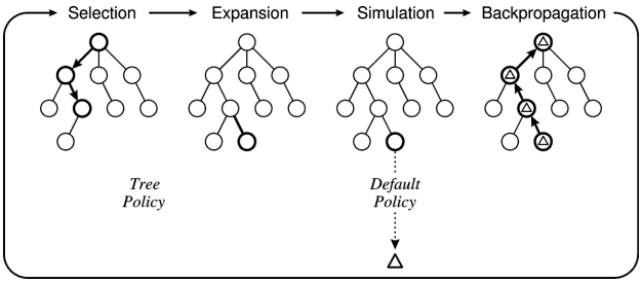


Figure 2. The image depicts one iteration of MTCS. Taken from [1].

a state of the game that is the end of the game and it has unvisited children. Then expansion happens when selection finds an appropriate node. Expansion adds one or more child nodes to the tree, which comes from the current available actions. Then simulation is ran on the newly expanded node. Simulation uses a default policy to continually select nodes until it reaches an end-of-the-game position. The simplest form of default policy is to select completely random actions until it reaches the end of the game. Then it takes the end game value it gives it to back propagation. Finally, back propagation takes the value of the simulated node and takes it all the way back up the tree, readjusting each node value according to the new result. For example, if the end result found was a win, each node traveled to get to that win would increase in value for the nodes of the corresponding winner. [1]

These four steps repeat over and over again until a terminal node is reached or the total budget for computation is reached. Once this is completed, there is another mechanism that helps choose what node the MCTS actually chooses. The two typical cases is that the mechanism chooses the node with the highest value or it chooses the node that got selected the most. The highest value typically dictates a method of exploitation, which is looking for the fastest way to win. For a two player game, every time the current player's turn is swapped the strategy for choosing an action is reversed. So if it's deciding the opponents move, the opponent would try to choose the move that has the lowest possible value. The highest selection rate typically dictates a method of exploration, which is looking for routes with more game play. The strategy for the opponent is chosen in this case and for now can still be thought of as choosing the lowest valued action. [1]

2.2.2 Deep Neural Networks. A neural network is a web of nodes that feed input into one another. In the web, nodes are organized into layers where every node in one layer feeds into every other node in the next layer. The connection between nodes is called an edge. The network starts with an input layer, then goes into a hidden layer with one or more layers of nodes, then finally ends with an output

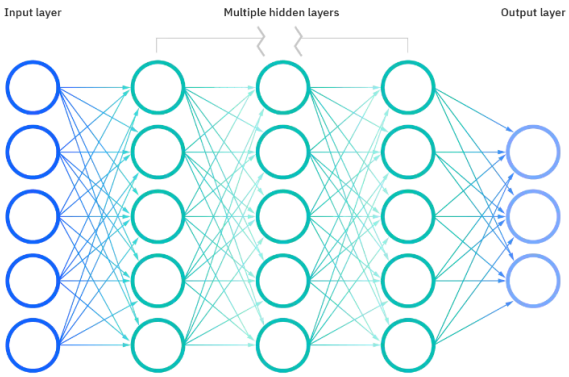


Figure 3. This is an example setup of a feed-forward neural network. The arrows between nodes are the edges and in this example all the information is flowing to the right. [6]

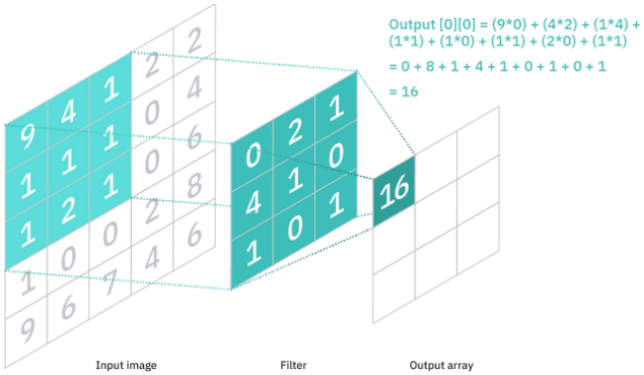


Figure 4. This is an image of the convolutional layer in a convolutional network (in the second step of calculating the first position of the output array, there is a missing (1*1).) [5].

layer. The output layer is usually pretty small, containing very few nodes figure 3. The input layer takes some series of numbers as the input where the numbers represent features of the data being analyzed. For example, if one were to input the features of a fish, they could input 47 for weight, 1 for breathing underwater, etc. Any feature that is represented as a yes or no question is represented with a 1 or a 0. The information in each input node is then sent to the first layer in the in hidden layer. Every edge contains some weight that gets multiplied to the number coming from the input node and the receiving node also contains some number called a bias that gets added to the total of the multiplication. So each node in the first layer of the hidden layer would be equal to the summation of all input nodes multiplied by their respective weight plus the node bias. In the hidden layer the sending of information happens a similar way. To start, the information in each node gets sent into a chosen activation function. For example, a very common activation function is a function called Rectified Linear Unit (ReLU). ReLU maps the value to itself if it is positive and 0 if it is negative. For example, 47 would become 47 and -47 would become 0. This is a common example of an activation function, however there are different types. This sending of information continues to move forward down the hidden layers until it reaches the output layer. The output layer can then be used to determine the result. [6]

For example, lets look at figure 3. There are five nodes in each layer except for the output layer where there are three. The top node of the input layer will have some numeric value that represents some feature in the data to be processed. That value then gets sent to all five nodes of the layer next to it. Each node receives a different value from the first input node because each edge has a different weight being multiplied to the value. Then each input node repeats this step. Once the first layer of the hidden layer receives and sums all the values from the input layer, it will add a bias. Then each node of the hidden layer will send its value to the activation function. Assuming we're using ReLU, all values will stay the same or turn into 0 depending on if it's original value was positive or negative. This process repeats until information gets sent to the output layer. The output layer will have three values that correspond to three categories. In this example, we could think of the first node representing a cat, second a dog, and third a fish. Whichever node has the highest value, called the confidence level, is the resulting category. [6]

Typically a neural network starts with random weights and biases. So it can be assumed at the start the neural network isn't capable of categorizing or processing any information accurately. In order to increase accuracy it uses something called a cost function. The cost function gives a value that represents the inaccuracy as a number, which is typically the difference. The neural network then adjusts the weights according to the cost value, the goal being to reduce the cost. So in order to have an accurate neural network, there needs to be some pre-processing to give it a chance to decrease the cost. This is usually done in training. Training can usually only happen when the network already knows the answer to the data it is trying to guess, since the only way for the weights to get adjusted based on the correct answer is to have an undeniable correct answer. [6]

This type of neural network is called a feed-forward network. It is one of the simplest networks and is typically used for categorization. If a feed-forward network has more than three layers then it is called a deep neural network. There are also other types of neural networks that are better for different types of inputs. For example, recurrent networks are used for natural language processing and convolutional neural networks are used for processing images or computer vision. Convolutional networks is what AlphaZero uses to process the chess board [10]. [5]

Convolutional networks keep the same idea where feature data is being sent through layers and manipulated by multiplying them by weights and adding biases. However, convolutional networks are specifically designed to take in

multidimensional data, such as an image or a chessboard. The image may be viewed as width*height*3, where 3 refers to the values of the three RGB channels, and a chessboard is 8*8*n, where n is the amount of information for each of the squares. The n in the chess example is auxiliary information like castling rights, where the pieces are, etc. A convolutional network has three types of layers: the convolutional, pooling, and fully connected layers.

The convolutional layer is where most of the computation occurs and can be broken into three distinct parts, as shown in figure 4. There is the input image, filter, and output array. As I mentioned before, the input image would be the individual pixels and the corresponding RGB values. Then there is a filter which is usually a 3 by 3 matrix of weights. The filter is typically smaller than the image, and looks at 3 by 3 sections of the image. The filter gets multiplied to the corresponding sections and each result is summed and put into a corresponding position in the output matrix. Then the filter sweeps across the entire input image until it has visited each section. Since the output isn't connected to every spot in the input image, convolutional layers are considered to be "partially connected". After each convolution operation, ReLU gets applied to the output matrix. Each convolution operation can be followed by more convolution operations. This helps to break down the input into separate parts. For example, if we were to attempt to determine if an image is a bicycle, then we could think of every individual piece of the bicycle as a lower-level pattern in the network. We would break it into the wheel, seat, gears, handlebars, etc.. Inevitably the image gets broken down into numerical values, making it possible for the network to process relevant patterns.

After the convolutional layer is the pooling layer. The pooling layer is responsible for decreasing the complexity of the data that comes out of the convolutional layer. The pooling layer uses a filter similar to that of the convolutional layer, however instead of a matrix of weights it uses some aggregation function. The more common function is called max pooling. Max pooling moves across the input and selects each position with the maximum value to send to the output array. This ensures that the neural network is taking into account the most important features of the data that it is trying to process.

The last layer is the fully connected layer. The fully connected layer is very similar to the feed-forward network where all the nodes from the pooling layer are connected to each node in the fully connected layer, making it the only layer that is not partially connected to the output layer. This last layer then uses another activation function called softmax to map the values between 0 and 1 onto multiple nodes. Finally the convolutional network can use this information to make a decision. This decision can also be thought of like the previous network, where each node in the output layer has some corresponding category or guess. [5]

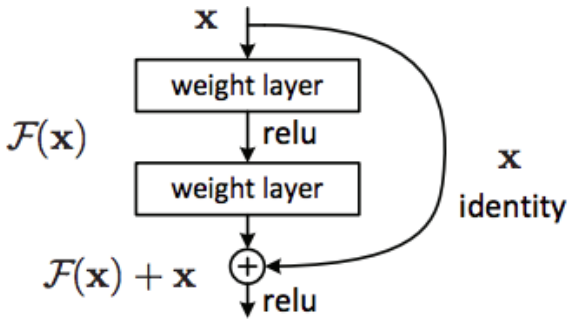


Figure 5. This is an example of a Residual Network block [5].

The main issue with convolutional networks is that it takes a lot of computational power to perform. It has been shown that there is a dropoff where increasing the complexity no longer is worth the amount of computation it takes. However, there is a different way to add complexity without adding a lot of computation. A Residual Neural network (ResNet) is basically a convolutional network with an extra feed-forward step as shown in 4. This shows a single layer of convolution, where x is the output of the previous convolutional layer and the function F is ReLU. F(x) + x gets sent to the next layer. [8]

2.2.3 AlphaZero. AlphaZero is the chess engine chosen by the researchers for this study because it can learn a new rule-set on its own without any human gameplay. AlphaZero uses both a deep neural network (DNN) and Monte Carlo Tree Search (MCTS). The neural network is used to evaluate each state of the board and the MCTS is used to navigate states [7]. AlphaZero learns move probabilities and value estimates from self-play games, which each self-play game is used to guide further evaluations of moves [9].

AlphaZero's neural network is a residual convolutional network with 19 residual blocks and 1 convolutional block, where a residual block consists of two 3x3 convolutions both followed by ReLu and a convolutional block has one convolutional layer followed by a ReLU. The input is an 8x8x(14h+7) multidimensional vector, where h is the number of turns. For example if we were to consider one turn, then h=1 and we would only consider the current position of the board and the vector would be 8x8x21. The first 12 8x8 channels are meant for keeping track of the board positions. It is the followed by channels representing the number of repetitions, the side to play, castling rights, irreversible move counter and total move counter. For every h there is another position of the board that gets inputted.

AlphaZero's nerual network starts with a random set of parameters that get trained by reinforcement learning from self-play games. Each game in training is played by running the MCTS from the current position. The move selection to the next position can either be chosen greedily or proportionally. Proportionally means prioritizing moves that were

evaluated highly by the neural network and greedily means to choose a move that has been visited a lot by the MCTS. The network outputs a vector of move probabilities along with the expected outcome of the game in the given position. Then once the end of the game is reached the network parameters are adjusted according to the terminal value of the last position using some cost function. So the weights in each position will get adjusted towards what the actual outcome was at the end of the game. For example, if the game ends in a loss then AlphaZero will use the -1 value to check against the expected outcome in each position. So if the position right before the lost was closer to 1, then the networks weights will get adjusted to give an output close to -1 in that position. This process happens hundreds of thousands of times before AlphaZero starts to become an optimal player. [9]

3 Methods

In the paper [10], Nenad Tomasev et al. explore ten chess variants and write about their results. In this section, I describe how the authors conducted their study, including what variants they used, how they set up AlphaZero for training, and what data they collected.

3.1 Chess Variants

The goal of choosing chess variants is to keep the game as close to classical chess as possible, while trying to create a new space for AlphaZero to learn. In the process, the researchers are watching for AlphaZero to uncover new openings, middle game strategies, and end game strategies.

In some cases, there needs to be a couple of alterations to the rules rather than just one big change. This is due to new rules conflicting with old rules. For their study, Tomasev et al. only consider changes to 2 rules when the main change also requires a change to the 50 move rule (if there are 50 unique moves without a pawn move or capture, the game results in a draw). There are no rule changes that involve changing the board, the pieces used, or the arrangement of the pieces. A list of chess variations is shown in Table 1. [10]

Variant	Rule Change
Pawn one square	Pawns can only move by one square
Stalemate=win	Forcing stalemate is a win rather than a draw
Torpedo	Pawns can move by 1 or 2 squares anywhere on the board. En passant can consequently happen anywhere on the board.
Pawn-sideways	Pawns can also move laterally by one square. Captures are unchanged, diagonally upwards Additionally, pawn moves do not count toward 50 move rule
Self-capture	It is possible to capture one's own pieces

Table 1. List of Chess Variants adapted from [7]. More variants can also be found there.

3.2 Training

For each rule alteration listed in Table1, AlphaZero starts from scratch with the same set of weights each time. "The models were trained for 1 million training steps, with a batch size of 4096 and allowing for an average 0.12 samples per position from self-play games. In order to encourage exploration during training, a small amount of noise was injected in the prior move probabilities before search." The batch size is the number of samples processed before the network's parameters gets updated [2]. Noise is something added to the dataset being trained on to prevent the neural network from completely memorizing it. Here it is added to prevent AlphaZero from always attempting to pick the same moves, hence preventing memorization. In the first turns of each self-play match, diversity is promoted by stochastic move selection by picking final moves proportional to the MCTS visit counts. [7]

For each chess variant, the researchers trained AlphaZero on 10,000 self-play games played at 1 second per move and 1,000 self-play games played at 1 minute per move as shown in figure 6(a) and figure 6(b). [7]

4 Results

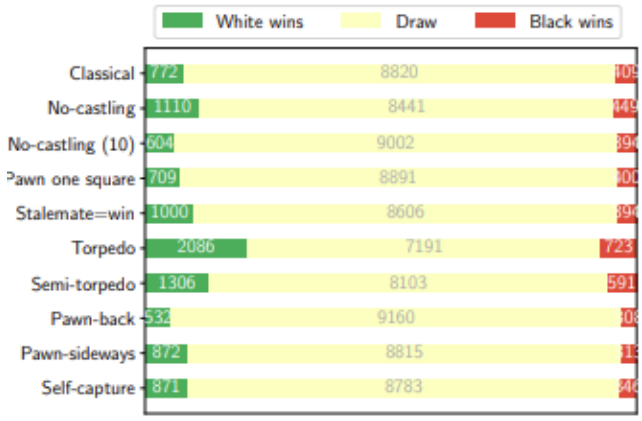
4.1 Utilization of special moves

Some of the chess variants allow for AlphaZero to choose between some new moves. It isn't clear just by the win rates how often AlphaZero chooses to utilize these special moves. Table 2 shows how often the special moves were used in the test data. More information on this can be found in [9].

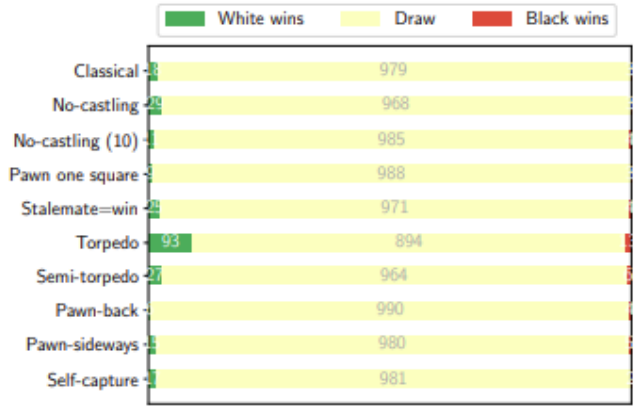
As can be seen in Table 2, self-capture chess had really low usage compared to the other variants. 86.9% of those self-captures are pawns with descending percentages depending on how valuable the piece is. It was also found that in some cases AlphaZero would capture its own piece because it was incentivised to explore rather than exploit. So sometimes

Variant	% of games	% of moves
Torpedo	94%	2.4%
Pawn-sideways	99.6%	11.4%
Self-capture	52.5%	0.7%

Table 2. A list of how often AlphaZero used the special ruleset moves given to it in the variations with special moves.



(a) 1 second per move



(b) 1 minute per move

Figure 6. The game outcomes of 10,000 AlphaZero games played at 1 second per move (a) and 1,000 AlphaZero games played at 1 minute per move (b) for each different chess variant [7].

instead of getting a checkmate in a single move, it would choose to take the longer route to a checkmate.

4.2 Analysis

The researchers' assessment heavily relied on the expertise of the chess grand master (GM) Vladimir Kramnik, a previous chess champion. The patterns for each variant is characterized to help give insights into each variant on what type of players might enjoy which variant. The following is a short summary of the key takeaways that Kramnik saw in each variant. [10]

Pawn one square chess is a slow variation. Kramnik mentions in could be a good training tool, since the slower build of structure gives better insight on how pawn structure is helpful overall. It also becomes hard to initiate quick attacks since the pawns can no longer "jump" from their original position. Overall this reduces the decisiveness of the game, therefore creating more draws. [10]

Stalemate=win chess did not impact the beginning or middle of the game in any interesting way. Being as stalemates typically occur towards the end, this variation only really changed the evaluation of the end game. Therefore, this chess variant is deemed to be not very useful and not super interesting since the overall effect of game play and strategy is minor. [10]

Torpedo chess makes the game a lot more decisive and adds value to pawns that have no opposing pawns in front of them. Pawns become a lot harder to stop since the rule essentially halves the amount of moves before promotion for every pawn. This creates new patterns in all stages of the game and creates more opportunities for attacking. Overall the game becomes more decisive and strategic. [10]

Pawn-sideways becomes very complex very fast for those who are experienced in looking at pawn structures normal to classical chess. Being able to move pawns sideways makes it harder for players to force weaknesses in their opponent's pawn structure. Both players inevitably have very fluid pawn structures and each position becomes hard to evaluate. Due to the complexity, this variation of chess is heavily dependent on deep calculation of the position. In AlphaZero's games a lot of strategies are developed that are not possible in classical chess. This variation is very tactical and complex. [10]

Self-capture chess is an entertaining variant, since the only new move provides another way of sacrificing pieces to gain some form of positional advantage. However, not every game involves self-capture since it is not possible in most positions to sacrifice a piece for an advantage. Only about half of the games use this strategy, and most of the time it is used to open up a position that is being blocked by pawns. [10]

5 Conclusion

AlphaZero allows testing rule variations without involving any human subject. As shown in this paper, variations of chess can have both drastic and small changes to the flow of the game. For example, self-capture chess had a surprising low usage rate as well as low influence on the game, whereas pawn-sideways chess was the most complex variation of chess. Being as AlphaZero is already capable of learning chess, shogi, and go, it is not too far reaching to say that AlphaZero could be adapted for other board games as well. There are some obstacles to be overcome for multiplayer games that have more than 2 players and single player games. Going one step further, it would be amazing to see an implementation of this for video games or similar complicated real-time rule sets. Overall this research can be helpful for the chess community as is, but hopefully in the future it can be used for other kinds of games as well.

References

[1] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[2] J. Brownlee. Difference between a batch and an epoch in a neural network, 2022.

[3] chess.com. Pawn structure in chess - chess terms. https://www.chess.com/terms/pawn-structure.

[4] chess.com. Special chess moves: Chess terms. https://www.chess.com/terms/special-chess-moves.

[5] I. C. Education. Convolutional neural networks, 2020.

[6] I. C. Education. Neural networks, 2020.

[7] T. McGrath, A. Kapishnikov, N. Tomaev, A. Pearce, D. Hassabis, B. Kim, U. Paquet, and V. Kramnik. Acquisition of chess knowledge in alphazero. *ArXiv*, abs/2111.09259, 2021.

[8] C. Shorten. Introduction to resnets, 2019.

[9] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[10] N. Tomasev, U. Paquet, D. Hassabis, and V. Kramnik. Assessing game balance with alphazero: Exploring alternative rule sets in chess. *CoRR*, abs/2009.04374, 2020.

[11] Q. Zhou. Has the number of draws in chess increased?, 2018.