**AlphaZero: The New Best Chess Engine**
**Mattia Danese**
**Introduction to Machine Learning (COMP 135)**

In December of 2017, DeepMind Technologies developed AlphaZero, an AI able to play the age-old Chinese board game Go, Chess, and Shogi ("Japanese Chess"). AlphaZero is an extension of the previously developed AlphaGoZero AI (in October 2017), which was only able to play Go, thus AlphaZero has a lot of similar characteristics to AlphaGoZero. The architecture (deep neural network) and algorithm (Monte Carlo search tree) of AlphaZero stems from those of AlphaGoZero. Having "Zero" in the name comes from the fact that, like AlphaGoZero, AlphaZero has no prior knowledge of any game apart from the respective rules (it trains just from self-play and not from a database of already-played games with known outcomes). In the most basic terms, AlphaZero uses its neural network to find a set of best next moves that result in the highest winning percentages, and then further explores these moves in a game tree to find the ultimate best move [1]. Additionally, it is worth mentioning that DeepMind Technologies is owned by Google, allowing AlphaZero to train using 5,000 custom-made optimized tensor processing units (TPUs) [2]. In this paper, there will be a focus on the ability of AlphaZero to learn and play Chess; however, these analyses are consistent for the other games and can be applied to them with minor deviations.

In order to gauge the power of AlphaZero, it is important to acknowledge its achievements and statistics before diving down into its implementation. Since AlphaZero had no prior exposure to move/ending tables, it started its first training games (against itself) by making random moves. Two hours into training, AlphaZero already started to perform better than human grandmasters; four hours into training, it garnered a higher ELO (Chess) rating than the (then) best Chess engine in the world, Stockfish. 44 million games later, over the course of nine hours,

AlphaZero became the undisputed best Chess engine in the world, winning 28 games, drawing 72 games, and losing 0 games against Stockfish in a 100-game tournament [1,2].

This tournament, having a time limit of one minute per move, spurned some controversy, though, as some people claimed AlphaZero was operating on Google servers and therefore had an advantage since Stockfish was designed to run a laptop. This resulted in another tournament being held in early 2018 between these two Chess engines, but this time they played 1,000 games where each game had a total time limit of three hours (per engine) with 15 seconds added to an engine's total time after each move. As expected from the previous tournament, AlphaZero won 155 games, drew 839 games, and lost 6 games against Stockfish, proving yet again AlphaZero is the best Chess engine in the world [3]. This made headlines across the Chess community, prompting many comments from prominent figures. One such figure, an International Master named Levy Rozman, made a YouTube video analyzing a couple of games from this tournament that Googled released. During the video, Rozman states, "one thing about AlphaZero is it's incredible ability to just leave material in the hands of its opponent…[it doesn't] need immediate benefit." The International Master goes on to say, "AlphaZero begins playing like an absolute savage…[and] baits Stockfish into pushing its pawn ". Comments like these, from an experienced Chess player, bring to light the true power of AlphaZero as they make it seem as if AlphaZero was going up against a complete amateur, and not what used to be the best Chess engine in the world [5].

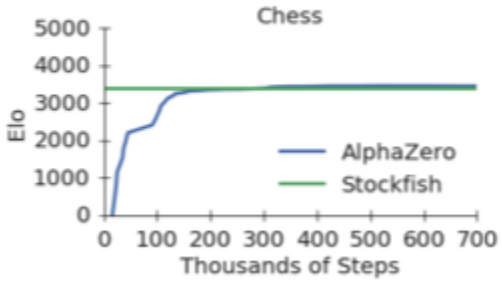
It is quite surprising that AlphaZero was able to beat Stockfish after a relatively little amount of training time, especially since Stockfish's implementation (an Efficiently Updatable Neural Network and an Alpha-Beta search tree) is very comparable to that of AlphaZero. What is even more fascinating is that AlphaZero examines about 60,000 positions per second while

Stockfish examines a striking 60 million positions per second [4]. This goes to show just how more efficient AlphaZero's searching algorithm is compared to a traditional Chess engine like Stockfish.

As previously mentioned, the first of two main components of AlphaZero is a feed-forward neural network. A feed-forward neural network is a combination of one or more layers of nodes (neurons). The first layer of the neural network is the input layer, containing one neuron for each feature of the data. The last layer of the neural network is the output layer, containing one neuron for each class or possibility of output. In between these two layers are one or more hidden layers, containing neurons that connect to the layer above and the layer below. The hidden layers are also what give the neural network the shape of an acyclic graph. Usually, the neurons in the output and hidden layers connect to *every* neuron in the immediate previous layer. In these connections from neuron *a* to neuron *b*, the neuron closest to the output layer, *a*, stores a specific weight which is applied to all input sent by *b* and received by *a*. This weighted input is then applied to the neuron's assigned function (independent from the function of all other neurons) and sent to all the connected neurons in the next sequential layer. The neural network learns through a process called back-propagation, where training data is sent through the network and the error of the output neurons, relative to the correct output, are calculated. The error is then used to adjust the weights for each neural connection going from the output neurons to the input neurons (direction is backwards). This procedure is repeated until error is minimized and falls below some predetermined threshold [7].

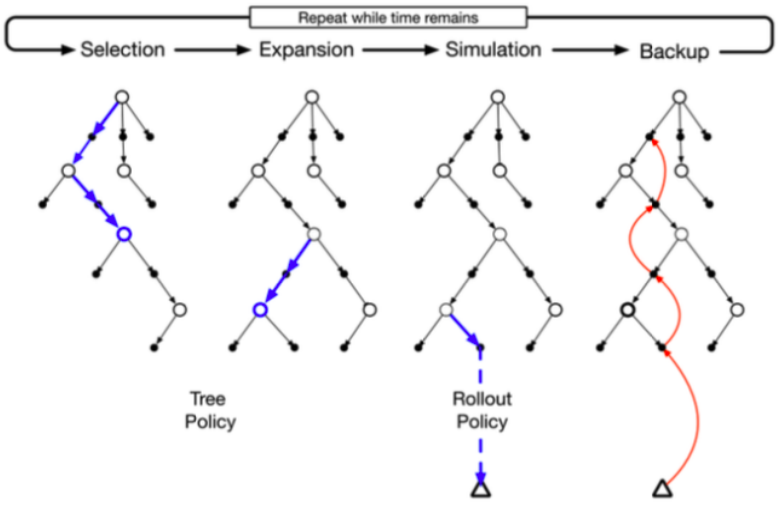
In terms of AlphaZero, "its neural network takes the board position *s* as an input and outputs a vector of move probabilities **p** with components $p_a = Pr(a|s)$ for each action *a*, and a scalar value *v* estimating the expected outcome *z* from position *s*, $v \approx E[z|s]$." AlphaZero then

uses these best move probabilities to *learn*; in other words, AlphaZero minimizes the associated error of each move relative to the actual outcome (known at the end of the current game), and uses the finalized probabilities to guide its tree search in future games. The training procedure of AlphaZero started with randomly initialized parameters and continued for 700,000 steps, with a *mini-batch* size of 4,096. Traditionally, a neural network has predetermined training data that it can train on, but AlphaZero did not. AlphaZero learned entirely from "self-play reinforcement learning"; it started out picking random moves and evaluated the strength and winning percentage of next moves through its (at first minal) knowledge of already played moves and the game rules. This, as one might expect, resulted in AlphaZero performing very poorly at the beginning of its training, but its growth was exponential and reached the level of Stockfish at just around 250,000 steps. Below is a graph that shows AlphaZero's ELO (Chess) rating in relation to the amount of training steps taken [2].




The second main component of AlphaZero is the Monte Carlo search tree (MCTS) algorithm. AlphaZero uses a game tree because Chess, along with Go and Shogi, "can be represented as a tree of possibilities," but specifically uses the MCTS algorithm to search the game tree because the tree gets very large very quickly [1]. In particular, an MCTS uses Monte Carlo simulation—a model used to predict probability of different outcomes when a random variable is present—to hone in on more promising nodes in the tree, and thus avoiding the

(impractical) need to visit every node. In an MCTS there are four steps which are repeated some number of times (in theory infinitely): selection, expansion, simulation, and backup. The first step, selection, deals with the balance between exploration and exploitation (the tendency to pick the best known move or to pick an unexplored move and see if it leads to a better result). Each node in the tree has a Upper Confidence Bound (UCB) value and the *leaf* node (a next move) with the greatest UCB, out of the set of possible best next nodes provided by the neural network, is selected. In order to keep a balance between exploration and exploitation, the UCB of each node is dependent on how many times that node is visited (more visits results in a lower UCB). In the second step, expansion, an unexplored node of the chosen *leaf* is simply explored at random (a random next move is executed). In the third step, simulation, one or more simulations of the game at this state are carried out with reward accumulations for each respective simulation. In the final step, backup, the accumulated rewards of the simulations are used to update the UCB values of nodes in the tree [6]. To further promote a balance between exploration and exploitation, sometimes the MCTS selection step picks a *leaf* node that the neural network thinks is not very promising, just in case it, in fact, turns out to be [1]. Below is a graphic showing this four-step process in action [6].



The beauty of how AlphaZero works is how both components, the neural network and the MCTS, work together. As stated above, AlphaZero's neural network outputs estimated probabilities of winning and which next moves are best. The probabilities of best moves are then used by the MCTS to focus on a particular part of the tree. For example, "if the network guesses that 'knight-takes-bishop' is likely to be a good move…then the M.C.T.S. will devote more of its time to exploring the consequences of that move." Moreover, in the early training stages of the neural network, the ability to look ahead provided by the MCTS algorithm, especially towards the end of the game, was very useful in initially tuning the weights of the network and starting the snowball effect of learning. In essence, the neural network helped the MCTS algorithm know where to search in the tree which, in turn, helped train and fine-tune the neural network [2].

All in all, the inner workings of AlphaZero are greatly complex and intricate, but, at a high-level, AlphaZero remarkably mimics how a human would play Chess. A complete novice player at Chess with no experience at all, just like AlphaZero, would most likely make random moves, see what happens, and then either repeat them in future games if they were good or not if they were bad. The components of AlphaZero even emulate the thought process of human players: the game tree is analogous to a player knowing the possible moves they can make on a given turn, the neural network is analogous to a player analyzing the strength and consequences of available moves, and the MCTS is analogous to a player focusing on the most beneficial moves and seeing what would happen in future turns, ultimately picking the move that they deem has the best chance of success (reward).

## References


[1] Somers, James. *How the Artificial-Intelligence Program AlphaZero Mastered Its Games*. The New Yorker, 28 Dec. 2018, www.newyorker.com/science/elements/how-the-artificial-intelligence-program-alphazero-mastered-its-games.

[2] Silver, David, et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm."

[3] Company, Sudo Null. "AlphaZero Again Beat Stockfish in a Match of 1000 Games." *SudoNull*, sudonull.com/post/7278-AlphaZero-again-beat-Stockfish-in-a-match-of-1000-games.

[4] Pete (User). "AlphaZero Crushes Stockfish In New 1,000-Game Match." *Chess.com*, Chess.com, 17 Apr. 2019, www.chess.com/news/view/updated-alphazero-crushes-stockfish-in-new-1-000-game-match.

[5] GothamChess, director. *How AlphaZero Completely CRUSHED Stockfish*. *YouTube*, 13 May 2021, youtu.be/8dT6CR9_6l4.

[6] Wang, Benjamin. "Monte Carlo Tree Search: An Introduction." *Medium*, Towards Data Science, 11 Jan. 2021, towardsdatascience.com/monte-carlo-tree-search-an-introduction-503d8c04e168.

[7] Goodfellow, Ian, et al. *Deep Learning*. The MIT Press, 2017.