RESEARCH ARTICLE OPEN ACCESS

A Survey of Deep Learning on Chess

Fenil Mehta[1], Hrishikesh Raipure[2], Shubham Shirsat[3], Shashank Bhatnagar[4], Prof. Bailappa Bhovi[5]
[1,2,3,4,5]*(Department of Computer Engineering, International Institute of Information Technology, Pune, India)*
[1]*(Email id: fenilgmehta@gmail.com)*
[2]*(Email id:hrishikesh.raipure64@gmail.com)*
[3]*(Email id:shubhamshirsat00@gmail.com)*
[4]*(Email id: shashank.bhatnagar27@gmail.com)*
[5]*(Email id: bailappab@isquareit.edu.in)*

ABSTRACT
This paper describes the various approaches taken using deep learning algorithms for the purpose of developing a chess program that can perform better than any of the previously developed hard coded game specific engines. Some of the discussed models outperform the current top chess engines or expert human players, some models have achieved very high performance, setting new benchmarks in the domain, while some of these models implement a new architecture or methodology and achieve average performance.
Keywords **-** Artificial Intelligence, Chess Engines, Lookahead, Neural Network.

Date of Submission: 14-04-2020 Date of Acceptance: 28-04-2020

I. INTRODUCTION

One of the most popular applications of artificial intelligence is developing a generalized algorithm or a methodology which can work on any board game, instead of having to develop game-specific algorithms. The idea to develop such applications arises as humans can improve in any game after playing for a few times. This ability of identifying repetitive patterns from given data has inspired many researchers to work on deep learning's application on games, while also aiming for artificial general intelligence.

The objective here is to examine the previous research about developing chess engines using deep learning. The papers used here go through the development of very simple network architectures to highly complex networks with a wide variety of features and also many hand tuned features and information unique to the domain. We attempt figuring out the working of the architectures and various search algorithms which lead to an improvement in the model's performance.

The papers surveyed discuss the properties of the different architectures used, their advantages and disadvantages, and their performance relative to some given benchmark. The choice of this architecture is in turn, dependent on the choice of board game selected, the target engine or minimum score to achieve, or to reach a certain level of performance that was previously unachievable in that context. Some of the architectures considered are Multi-dimensional Recurrent Neural Network (MDRNN), Multi-layer Perceptron (MLP), and Convolutional Neural Networks (CNN).

The papers surveyed also discuss the importance of choosing an efficient searching technique. For a system to have as low time complexity as possible, choosing an efficient searching algorithm is important. Some of the searching techniques mentioned are Alpha-Beta search, Quiescence search, and the Monte-Carlo Tree Search (MCTS). The model's performance is hugely affected by the searching algorithm.

The input representation also has to be considered because some architectures work with only certain input formats. For example, A Convolutional Neural Network works better with an array representation of input. But having to process every image into a multidimensional array is time consuming and will deter the speed of execution. A flat array encoded with bits can improve the performance as it will be much easier to perform computations on.

II. LITERATURE SURVEY

[1] This paper introduces NeuroChess, which trains itself from the end results of played games. NeuroChess uses evaluation functions based on the Artificial Neural Network (ANN). It makes use of inductive neural network learning, temporal differencing (TD), and a variant of explanation-based learning, called explanation-based neural

network (EBNN). It uses TD to construct neural network evaluation functions. The games are analyzed in terms of a chess model using EBNN. Temporal Differencing was used to find an evaluation function V, which scored chess boards according to their usefulness from the perspective of one side. For learning the training pattern, TD transforms the entire chess game denoted by a sequence of chess boards. The final board values i.e. $V(S_{final})$ will be 1 if S_{final} is a win for white, 0 if S_{final} is a draw and -1 if S_{final} is a loss for white. A separate neural network, called the chess model M, represents domain-specific knowledge, which captures temporal dependencies of chess board features. EBNN uses knowledge learned from previously played expert-level games for the purpose of analysis. Both, the target values' values and slopes contributed for the functions to fit more accurately. Hence, to achieve this, the Tangent-Prop algorithm was used.

[2] The various board games have either a fixed or a flexible size. Increasing board size leads to exponential increase in the branching factor. Game engines have increased time and space complexity because of such scalability issues. Hence, to overcome such an issue, this paper makes use of MDRNN (Multi-dimensional Recurrent Neural Networks). Atari-Go and Gomoku, the variations of Go with simpler rules were chosen to develop the model on. MDRNN is implemented along both dimensions, resulting in a large but simple feedforward neural network. To keep the architecture free from domain-specific knowledge, evolutionary methods are used and to not depend on any particular algorithm. Given an equal number of hidden units, MDRNN performed better than MLP when assigned random weights and averaged over 100 games. As the board size increased, the performance of MDRNN got better, and deteriorated for MLP. It was found out that the correlations were always positive and high for all board sizes except for Gomoku with board size 5x5 which had a score of 0 in majority of the networks. The proportionality p was greater than 0.5, representing good scalability. Training the architecture using coevolution was done to make results independent of the biased fitness measure. The results showed that the performance of the architecture wasn't affected significantly by the size of the board. In the final experiment for scalability in trained networks, it was found that the correlation was low but p was high.

[3] For finding the optimal path to go from one board state to another, neural networks are used. It was found that, when using neural networks, pruning of costly branches was done at a greater scale. This contributed to the development of hybrid AI game-tree search systems. Here, they have experimented with two different neural classifier architectures:

I. The first MLP architecture consisted of one hidden layer, having 30 neurons in it.
II. The second MLP architecture had two hidden layers, with 30 and 20 neurons respectively.

The input is a vector of the set {-1, 0, 1} where each square is represented by five values of the mentioned set where negative numbers represent presence of black piece and positive numbers represent presence of white piece. Squares with no pieces are represented by zeros.

The proposed approach is geometrically oriented and relies on calculating Manhattan distance between the potential target square and a particular, arbitrarily chosen predefined square.

[4] Devising various approaches for evaluation function that can set weights to the chess engine's neural network has become a popular research topic in recent years. In this paper, the authors have described a methodology for calculating each chess piece's positional value. This chess engine uses Alpha-Beta searching algorithm with iterative deepening, stabilization of positions through the Quiescence algorithm, hash tables and move generator through the 0x88 hexadecimal method. The evaluation function that gives heuristic value of position for either side (black or white piece) is given by:

$$f = \sum_{i=1}^{r} m_i + \sum_{i=1}^{q} (c_i \times p_i)$$

m_i is the material value of the piece i and it is static, c_i is the adjustment of the weight pi ($c_i = 0.5 \times m_i$), $p_i$is the positional value of the piece i and it is dynamic. pi ∈ [0, 1].

This paper integrates a genetic algorithm known as an evolutionary algorithm that helps change the weight of the deep neural network. "Initialize population" module assigns initial random weights to the neural network, features of the position obtained in module feature extraction, "Play tournament" module coordinates a tournament between n virtual players. The first n/2 virtual players with the highest points are chosen by the "Selection" module. These players then generate the rest of the n/2 players by undergoing mutation through the "Mutation" module. After the generation process, the evolutionary algorithm begins execution. It is expected that by adding the number of inputs to the neural networks, the positional value of the chess pieces will be assessed in a more precise manner and, consequently, the strength of the chess engine will be increased.

[5] This thesis used machine learning in order to create a new chess engine called Giraffe. The Giraffe program introduced in this work makes use of the TDLeaf(□) algorithm. The implementation took positions as inputs and for each

position a sequence of numbers was given as output that worked as a signature. The position similarity is the reason for humans' high search efficiency. Here the unwanted searching could be avoided if humans can make out the equally efficient moves. The average branching factor of the search trees was drastically reduced by this. The Giraffe engine gives the probabilities of all the moves by a particular chess piece that can take place. The main drawback of this engine is its low search speed, which is caused by the low hit rate. The main focus of this thesis was to evaluate the position accurately else deeper and wider searches were required to compensate. This chess engine played very well in the start and end of the game because this chess engine focused on the tricky moves rather than considering the far moves and it also understood the complicated moves which is difficult for human players to understand.

[6] This thesis has shown how trained deep value neural networks are able to play chess as high level human chess players play, without looking ahead more than one move. In this thesis, performances of Multilayer Perceptron and Convolutional Neural Network are examined. Data sets containing a collection of games represented in Portable Game Notation (PGN) are parsed and different board representations are created. These board representations are used as inputs to the Multilayer Perceptron and Convolutional Neural Network architecture. To assign every chess board position a fractional centipawn (cp) value as a label, Stockfish, one of the powerful chess engines used so that classification and regression experiment can be performed. Different ANN architectures have been trained to approximate Stockfish's evaluation function as precisely as possible. This neural network based chess program employs Mini-Max search and Alpha-Beta pruning algorithm for choosing next move.

The conducted experiments showed that MLPs performed better than CNNs. Given these results, the proposed system was still behind the strongest existing chess engines and the top human players.

[7] In this work, the DeepChess program is introduced, the performance of which is on par with grandmaster-level players. The architecture of DeepChess uses a deep neural network. No domain-specific knowledge is imparted in the model manually. Deep neural network training is performed in two sections, one is unmonitored pre-training and the other is supervised training. Identification of high-level features is accomplished by adopting unsupervised training by the model. Supervised training enables the model to be able to compare two chess board positions and select the more favorable one. The training depends completely on datasets of a few million chess games.

DeepChess assigns scores to the potential positions and this score represents how good the given position is. This model receives two positions as input and learns to predict which position is better after comparing both the positions. For training, pairs consisting of two moves are given as input. These pairs contain one move from a game where White is the winner, and another move from a game in which Black wins. DeepChess can be said to have an aggressive style, often sacrificing pieces to benefit in the long term.

Already trained Pos2Vec DBN used as initial weights for this DeepChess model's supervised network and during the training phase the whole network as well as Pos2Vec parts was modified. The Falcon chess engine was used as the baseline of the experiment. Falcon is a chess program which plays at the grandmaster level.

This style of playing chess is very similar to that of human grandmasters. Hence the DeepChess has an adventurous playing style with frequent positional sacrifices.

[8] In this project, an intelligent chess board is made which is very useful for the beginners to understand the rules of chess. The intelligent system proposed here helps the users to know all the correct positions of each piece on the chessboard. A back propagation neural network was used for this project as the training algorithm. The architecture described in here, works with the chess pieces according to their attributes without needing computer devices. The aim here is to find the next piece to be moved, given the current board state and also to obtain the available positions where this piece can move on.

Implementation of the hardware component of this project done by using the FPGA cart because of its simplicity, efficiency and ability to reset.

System uses LDR sensors to detect and determine which piece of chess to be delivered. The proposed system that replaces the proper position of each piece on the chessboard consists of 64 laser diodes, 64 pressure buttons, 64 LEDs.

[9] This paper introduces AlphaZero, a general reinforcement learning algorithm. This engine is applicable to Chess, Shogi and Go. The deep neural network takes input as the board position s and gives a vector of move probabilities p as output, having components pa = Pr(a|s) for every action a anda scalar value v evaluating the outcome z from position s. MCTS was used to get output as a vector having probability distribution over the moves. The parameters are arbitrarily initialized in the beginning and then trained by self-play reinforcement learning. For tuning of the hyperparameters, the optimization technique used was Bayesian optimization. Stockfish, Elmo and AlphaGo Zero were used for comparison with

AlphaZero. Here, 100 games were played to get the outcomes. These games had a time restriction of 1 minute for each move. Both AlphaZero and its predecessor worked on a single machine with 4 TPUs.The chess and shogi engines, Stockfish and Elmo respectively, played with the configuration of 64 threads and 1 GB hash size. They played with this configuration at their strongest level. AlphaZero won against all the opponents; losing zero games to Stockfish, eight games to Elmo and forty to AlphaGo Zero. It was observed that AlphaZero searches just 80k positions per second in chess and 40k in shogi, compared to the 70 million for Stockfish and 35 million for Elmo.

The algorithm of AlphaZero differs in many ways from that of AlphaGo Zero. AlphaGo Zero worked by evaluating and optimizing the winning probability, given that the outcome was binary. AlphaZero on the other hand, evaluated and optimized the expected outcome, based on the number of draws obtained and other outcomes.

It was observed that AlphaZero follows a human-like approach for searching as it focused on the most promising variations through its deep neural network.

[10] This paper considers various approaches using different combinations of architectures and input representations for training ANNs to evaluate chess positions. A dataset of around 3,000,000 different chess positions played by highly skilled chess players was taken and labelled with the evaluation function of Stockfish, one of the strongest existing chess engines.

They have compared the results of MLP and CNN for differently normalized four different datasets for different notations of chess boards, namely Bitmap and Algebraic notation. The outputs have shown that MLP is relatively better than CNN for all the datasets taken and both the board representations and have shown that Bitmap notation gives better results while on the other hand algebraic notation giving more information to the network adversely affects the results.

On testing the best MLP architecture with the Kaufman Test, the optimal move was played only twice - in position 3 and position 6. And in positions 12, 14, 15, 19, 22 and 23 losing moves were chosen by the ANN. While for the other positions, moves with maximum cp value difference of 1.5 were chosen as compared to the optimal move.

[11] Presents CrazyAra which is a Convolutional Neural Network based on supervised training for the chess variant crazyhouse. They have used networks with Monte-Carlo Tree Search to predict the game moves. There is only a smaller dataset of lesser quality than Go and chess, however, the results obtained are promising. They have used a more compact input board presentation by making the state fully Markovian, removed the history component as opposed to the AlphaZero and performed rescaling/normalization for better performance. To lower the changes of blunders, more sample efficient Monte-Carlo tree search has been used. Transposition tables, called Q-Values, are alluded to pick a move. These tables are used for sharing evaluations across multiple nodes.

III. CONCLUSION

On reviewing the previous works on using Deep Learning for chess, we find that there are numerous architectures that are suitable for developing a game engine which can outperform previously created programs or even defeat world champions. Certain search techniques, architectures and algorithms have been found to give promising results.

ANN have demonstrated huge reduction in the evaluation of low performing board positions and only highly advantageous positions are explored. However, hardcoded engines like Stockfish are able to explore millions of moves in one second and are able to find checkmate moves quickly in the game ending. ANN are able to play tactical games and have been able to win games but the computation power required is high as compared to hardcoded engines.

REFERENCES

[1]. S. Thrun, “Learning To Play the Game of Chess,” Proceedings of the 7th International Conference on Neural Information Processing Systems, 1994.

[2]. T. Schaul, J. Schmidhuber, “A Scalable Neural Network Architecture for Board Games,” Artificial Neural Networks – ICANN 2009 Lecture Notes in Computer Science, vol 5768, pp. 1005-1014, 2009.

[3]. C. Dendek and J. Mandziuk, "A Neural Network Classifier of Chess Moves," 2008 Eighth International Conference on Hybrid Intelligent Systems, Barcelona, 2008, pp. 338-343.

[4]. E. Vázquez-Fernández, C. A. Coello Coello and F. D. Sagols Troncoso, "Assessing the positional values of chess pieces by tuning neural networks' weights with an evolutionary algorithm," World Automation Congress 2012, Puerto Vallarta, Mexico, 2012, pp. 1-6.

[5]. M. Lai, “Giraffe: Using Deep Reinforcement Learning to Play Chess,” M.Sc. thesis, Dept. Computing., Imperial College London, London, 2015.

[6]. M. Sabatelli, “Learning to Play Chess with Minimal Lookahead and Deep Value Neural

Networks," M.Sc. thesis, University of Groningen, Amsterdam, 2017.

[7]. O. E. David, N. S. Netanyahu, and L. Wolf, "DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess," Artificial Neural Networks and Machine Learning – ICANN 2016 Lecture Notes in Computer Science, pp. 88–96, 2016.

[8]. A. H. Omran, Y. M. Abid and H. Kadhim, "Design of artificial neural networks system for intelligent chessboard," 2017 4th IEEE International Conference on Engineering Technologies and Applied Sciences (ICETAS), Salmabad, 2017, pp. 1-7.

[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. (Dec. 2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. [Online]. Available: https://arxiv.org/abs/1712.01815

[10]. M. Sabatelli, F. Bidoia, V. Codreanu, and M. Wiering, "Learning to Evaluate Chess Positions with Deep Neural Networks and Limited Lookahead," Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods, 2018.

[11]. J. Czech, M. Willig, A. Beyer, K. Kersting, and J. Fürnkranz. (Aug. 2019). Learning to Play the Chess Variant CrazyHouse Above World Champion Level with Deep Neural Networks and Human Data. [Online]. Available: https://arxiv.org/abs/1908.06660