Frankfurt University of Applied Sciences
Fachbereich 2: Informatik und Ingenieurwissenschaften
Informatik (B.Sc.)

Generating Chess Commentary using Machine Learning

Abstract

This paper deals with the question of how machine learning can be used to create a comprehensive analysis of chess games, which can be used to generate textual, human-understandable, commentary. In particular, we will look at what is needed to represent a chess board that can be used by the neural network to plan and compare moves in order to make an appropriate evaluation of a game of chess. Based on this, we will then explore how the neural network can convert the evaluation into natural language that humans can understand.

Lecturer: Konstantin Ernst
Course: Künstliche Intelligenz und wissenschaftliches Arbeiten
Winter Semester 22/23

Submitted by:
Max Semdner
Matrikelnr.: 1294899
max.semdner@stud.fra-uas.de

Declaration of authorship

I hereby certify that the following project report was written entirely by me and is based on my work unless otherwise indicated. I am aware of the University's regulations regarding plagiarism, including the following actions in the event of a violation. Any form of use of outside work is identified where appropriate and noted in the sources.

Max Semdner

Approved: _____

Date: _____

Contents

1 Introduction

In the mid-20th century, computer chess experienced its first breakthroughs thanks to the work of scientists like Alan Turing, Claude Shannon and John von Neumann. Alan Turing, the pioneer of artificial intelligence, was convinced that games were an ideal model system for machine learning.[1] This prediction has come true, and machine learning have proven to be an essential part of many chess engines today. In particular, recent projects such as AlphaZero, developed by DeepMind, show how efficient programs which use neural networks are in analyzing board games compared to traditionally used algorithms such as alpha-beta search.[2] Although chess engines have become a powerful tool, they have a lack of transparency regarding the moves they perform. Therefore, professional chess players and commentators are often needed to explain the intention of these moves. This dependence on human chess commentators can be a disadvantage, since moves found by computers can sometimes be misinterpreted or not understood at all. Especially for non-professional chess players these often seem incomprehensible. In what follows, I will explore the question of how this intransparency can be overcome using a neural network to create a virtual chess commentator that uses a built-in chess engine to translate the engine's intentions into a language that humans can understand.

2 Generating Chess Commentary

As described in the next section, each chess engine must meet certain requirements. Based on the requirements, context is generated. Context is a set of properties that apply to the current position on the chess board. These properties are used by the generation models to make comprehensive statements about the moves.

2.1 Chess Engine

2.1.1 Requirements

Any chess engine must meet a number of requirements in order to function. These requirements include the *representation of the chessboard*, the *search of the possible moves* and the *evaluation of the position*. These requirements can be implemented in a number of ways. Certain implementation options have become widely accepted. In the following, one implementation procedure is presented in detail for each requirement. These are state of the art implementations that have been in use for many years or have achieved great success in the recent past.

2.1.2 Board Representation

Since the computer cannot work with a physical chess board and pieces, these must be converted into a form in which the board, pieces, and position can be interpreted by the computer and later used for the input layer of the neural network. The most common form of representation is the data structures bitboards. A bitboard is implemented as an 8×8 array, which is the size of a chessboard. Each array element corresponds to a square on the board. A bitboard is created for each type of piece (pawn, knight, bishop, rook, queen and king) of a given color (black and white). This gives a total number of

[1] Cf. Levy et al. 1982, p. 44-45
[2] See Silver et al. 2018, p. 1

12 bitboards. Finally, the squares on which the figures are placed must be marked on the respective boards. This can be done in binary, where 0 means the square is empty and 1 means the square is not empty.

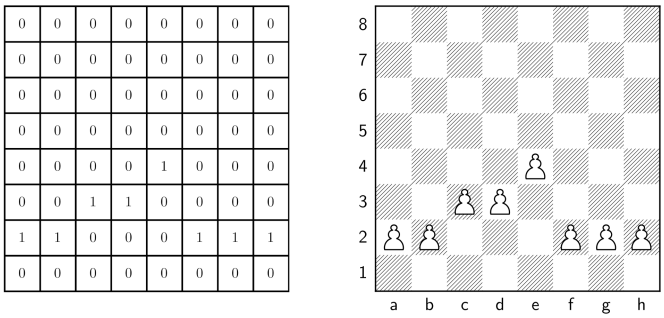

Figure 1: Representation of bitboard with white pawns (left) and the corresponding chessboard (right)

Through the usage of the logical operations, such as AND, OR, NOT the moves can be calculated. An advantage of the logical operations is, that be done quite fast by the processor. Furthermore, with x64 processors the position can be stored in one piece in the memory, as a bit string, since this is exactly 64 bits long due to the number of squares. Another advantage is that due to its simple representation, it can be used as input for the input layer in the neural network without the need to convert it into an understandable format.

2.1.3 Move Search and Position Evaluation

A challenge for both humans and computers is to find the best possible move. In fact, chess is considered unsolved, i.e. it is not known if there is an optimal strategy that always leads to victory, for either sides. The objective of a good chess engine is therefore to find the best move based on its computational capabilities. One factor to consider is the depth of analysis. Thus, each move must be considered not only in terms of the current state of the board, but also what effect it will have on subsequent best moves and positions. In order to find the best move, two tasks must be achieved, one is to find the legal, possible moves in the current and the following positions, and the other is the evaluation of these positions.

To implement these two tasks, many systems such as DeepBlue and earlier versions of Stockfish use human-defined evaluation functions and game tree search algorithms. The evaluation function receives the board as input and evaluates how good the obtained position is. Game tree search algorithms, such as MinMax, are then used to search for all possible moves (Usually, up to a certain depth, e.g. 15 moves). Each move leads to a new position. Since there is no information about these new positions yet, they need to be evaluated by the evaluation function. In the end, the path that has received the highest value from the evaluation function is chosen. Improvements such as alpha-beta pruning remove paths that are known not to yield high values of the evaluation. The problem with this method is that the evaluation functions have to be written by hand with the help of chess experts and constantly refined to achieve an optimal result.

To overcome this problem, and to get the best results from the engine for the commentator, machine learning is used. Silver et al. (2018) presented an algorithm that uses Monte Carlo tree search (MCTS) and a neural network. The neural network is used to evaluate an input position. It outputs two pieces of information: A vector with the move probabilities of all next legal moves and, a value, which tells what the player's chances of

winning are for the given position. MTCS is used to search deeper, i.e., to search many possible move paths and then select the path with the best neural network evaluation.

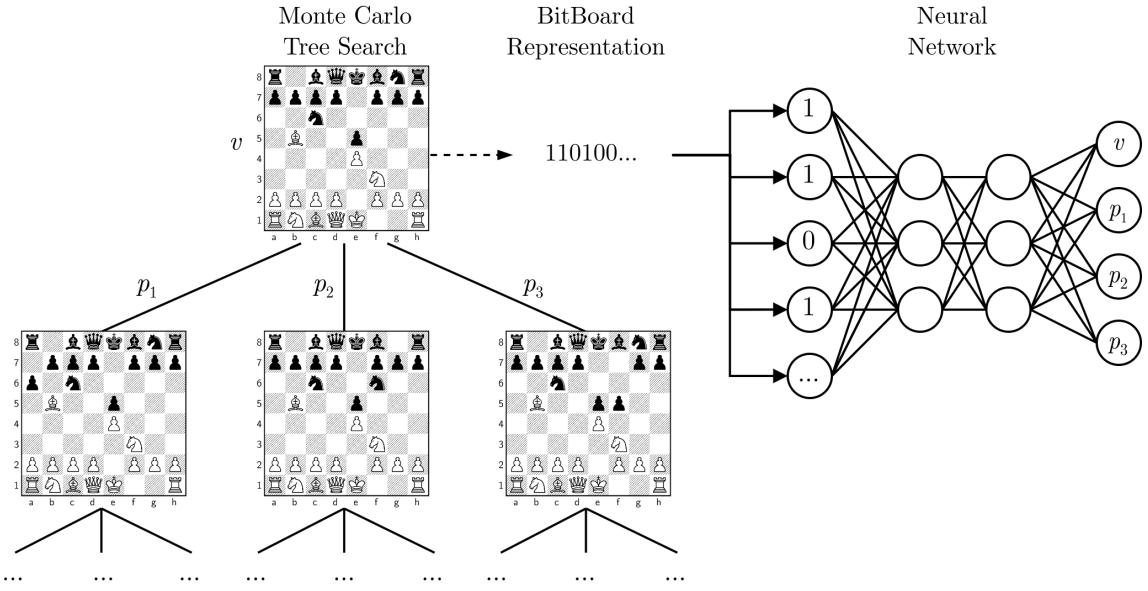

Figure 2: Simplified example of move search and position evaluation using MCTS and a neural network. (p_n move probabilities, v winning chance value)

In order for the neural network to deliver the best possible results, it must be trained with data. Instead of relying on existing data sets, Silver et al. (2018) used the approach of generating their own data sets through self-play. Initially a program exists that only knows the rules of the game, uses an untrained neural network for evaluation and the MCTS algorithm for move selection. The program executes the following four steps to train the neural network: (1) The program plays against itself by searching for several moves in advance through the MTCS, letting the neural network evaluate these moves for move probability and position strength, and finally choosing the path with the best evaluation. Every game is recorded move by move. At the end of each match, the final position of the sides are assigned lost (-1), drawn (0) or won (+1) to create a dataset.[3] (2) The neural network is cloned and the parameters are adjusted "to minimize the error between the predicted result [...] and the game result [...]" and train the network with the generated data set. (3) Let the new program play against the previous one. (4) The winning program is selected and the process is repeated from step 1. As Silver et al. (2018) showed, this method was able to outperform Stockfish, the strongest engine at this time, after only 4 hours.

2.2 Chess Commentator

2.2.1 General Approach

The chess commentator has the task to convert certain information, which it receives from the chess engine described before, into human understandable comments. Such a task falls into the domain of *sequence-to-sequence* processing. The idea of sequence-to-sequence models are that an input of a certain length is mapped to an output of a certain length, where input and output length can be different. For such tasks, Encoder-Decoder architectures has achieved great success in the past. The architecture consists of three

[3]See Silver et. al p. 2

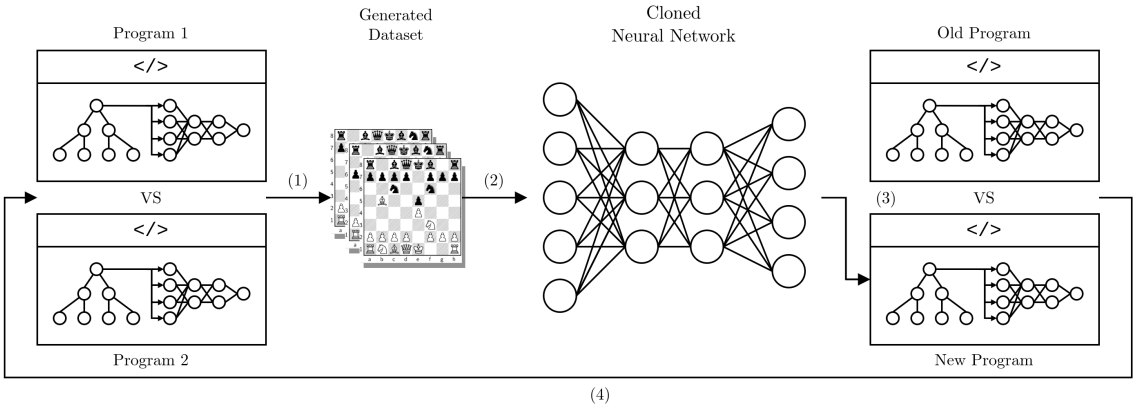

Figure 3: Steps of traning the neural network through self-play

parts, an encoder and a decoder, which are two *bidirectional recurrent neural networks (BRNN) using long short-term memory (LSTM) state cells*, and a context vector lying between them.

Recurrent neural networks (RNN) are a special type of neural network designed to process sequential inputs and recognize patterns in them. Due to an internal memory, RNNs are able to remember and reuse certain characteristics of the input. In general, an RNN works as follows: (1) recive an input (2) let the neurons generate an output, (3) copy that output (4) take the copied output and the new input and return to (1). Thus, the new RNN input consists of the output generated in the past and the new sequence part added, allowing the network to build a deep understanding of the sequence. An extension of an recurrent neural networks are bidirectional recurrent neural networks. While the neurons of RNNs always pass the generated output to the neuron in the immediate future, in BRNNs there is another level of neurons that can pass the output to past neurons. By including information from both the past and the future, an even deeper understanding of the sequence can be built. When (Sutskever et al., 2014) first introducted their sequence to sequence model they used RNN with long short-term memory state cells. Those are a special neuronal architecture of an RNN, which is why it is also simply referred to as LSTM or BLSTM (in case of bidirectional LSTMs). LSTM use a special kind of neurons, called state cells, which solve problems of RNNs and BRNNs that make it difficult to train them.

Glossary

DeepBlue First chess engine to win against a world chess champion. 4

Stockfish Most use free and open source Sachach engine. 4

References

(2018). *FIDE LAWS of CHESS*. International Chess Federation.

Czech, J., P. Korus, and K. Kersting (2021, May). Improving alphazero using monte-carlo graph search. In *Proceedings of the International Conference on Automated Planning and Scheduling, 31*, pp. 103–111. arXiv.

Hochreiter, S. and J. Schmidhuber (1997, 11). Long Short-Term Memory. *Neural Computation 9*(8), 1735–1780.

Jhamtani, H., V. Gangal, E. Hovy, G. Neubig, and T. Berg-Kirkpatrick (2018, July). Learning to generate move-by-move commentary for chess games from large-scale social forum data. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia, pp. 1661–1671. Association for Computational Linguistics.

Levy, D. and M. Newborn (1982). *All About Chess and Computers*. Springer Berlin, Heidelberg.

Silver, D., 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 (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Volume 362, pp. 1140–1144.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural networks. In *Proc. NIPS*, Montreal, CA.

Zang, H., Z. Yu, and X. Wan (2019, July). Automated chess commentator powered by neural chess engine. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, pp. 5952–5961. Association for Computational Linguistics.