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

Generating Chess Commentary using Machine Learning

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

Abstract

This scientific seminar paper deals with the question how machine learning can be used to generate comments on chess games. On the one hand it is shown how a chess engine can be built to analyze games and on the other hand how the information provided by the engine can be used by a virtual chess commentator to generate comments. The engine is built based on a neural network and a Monte-Carlo Tree Search algorithm and can be trained by self-play. The generation is based on an encoder-decoder model that uses a bidirectional short-term memory architecture. For a total of five different categories, comments are generated by corresponding generation models, which together form the virtual chess commentator.

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.[1] Alan Turing, the pioneer of artificial intelligence, was convinced that games are an ideal model system for machine learning.[2] 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, have shown how efficient programs which use neural networks are in analyzing board games compared to traditionally used algorithms such as alpha-beta search.[3] 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, most moves played by professional players as well as computers are incomprehensible, since they do not have the appropriate experience. To solve this problem, in the following the question "how machine learning can be used to generate annotations for chess games" will be discussed. For this the construction of a chess engine is presented, which corresponds in its play and analysis strength to the today's engines. After a brief introduction on how to analyze the moves in depth, the construction of a virtual chess commentator is described, which uses the information gained by the analysis of the chess engine to create detailed comments on a given position and move.

2 Generating Chess Commentary

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*.[4] 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 way of representing positions and movements of chess pieces is the data structure bitboards. A bitboard is implemented as an 8×8 bit array, which is the size of a chessboard.[5] Each array element corresponds to a square on the board. A bitboard is created for each type

[1] Cf. Keen (2009), p. 3
[2] Cf. Levy and Newborn (1982), pp. 44-45
[3] Cf. Silver et al. (2018), p. 1
[4] Cf. Klein (2022), pp. 75-76
[5] Cf. Boskovic et al. (2005), p. 360

of piece (pawn, knight, bishop, rook, queen and king) of a given color (black and white). Further 4 board for "2 casteling choices of each player"[6] are generated. This gives a total number of 16 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.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	1	1	0	0	0	0
1	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0

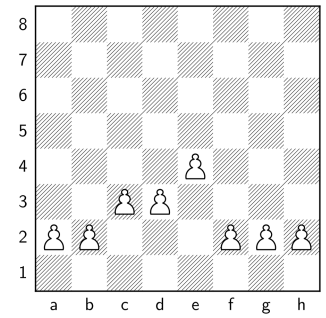

Figure 1: Representation of bitboard with white pawns (left) and the corresponding chessboard (right)

Bitboards can also be used to represent possible movements of chess pieces. This is achieved by performing certain logical operations on the bitboard. An advantage of the logical operations is, that be done quite fast by the processor.[7] 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.[8] Another advantage is that due to its simple representation, it can be used as input for the input layer in the neural network. The network could then be trained to evaluate specific chess positions and predict moves.

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.[9] 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 MiniMax, are then used to search for all possible moves (Usually, up to a certain depth, e.g. 15 moves).[10] 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.[11] The

[6] Zang et al. (2019)
[7] Cf. Boskovic et al. (2005), p. 360
[8] Cf. Segundo and Galán (2005), p. 67
[9] Cf. Allis (1994), pp. 7-8
[10] Cf. Klein (2022) pp. 75-76
[11] Cf. Keen (2009), p. 7

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.[12]

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.[13] MTCS is used to search deeper, i.e., to search many possible move paths and then select the path with the best evaluation. To select the next move MCTS selects "each valid move, play a number of random games"[14], generates statistics for these randomly selected paths and compares them. The best evaluated move of a path is played. The statistical evaluation for each path can be generated by the evaluations made by the neural network at each node of the search tree.[15]

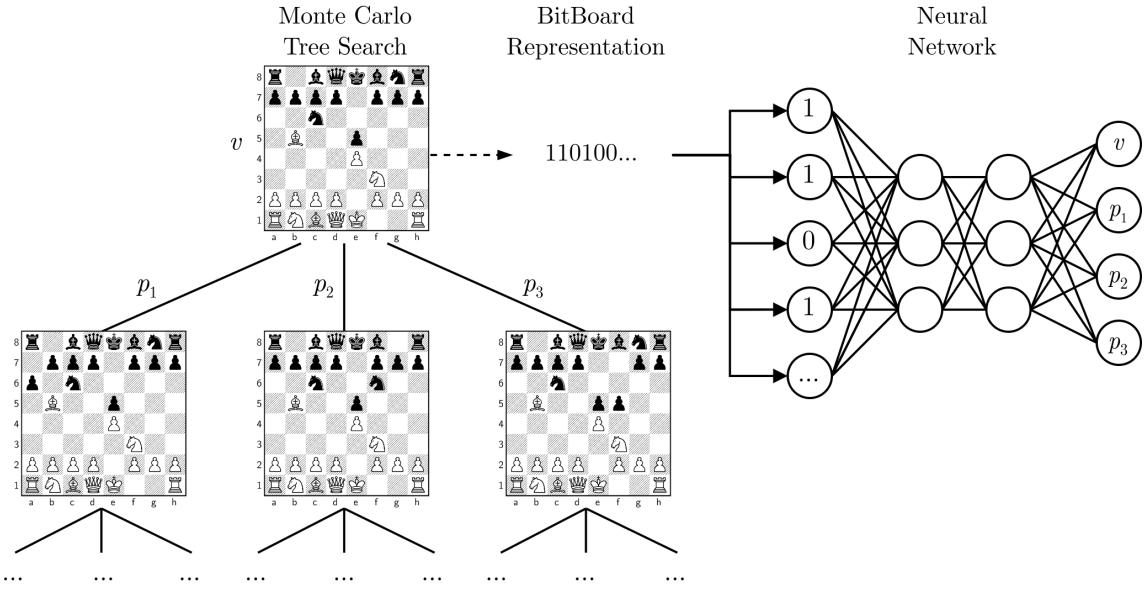

Figure 2: Simplified example of move search and position evaluation using MCTS and a neural network. (p_n move probabilities, v winning chance value)

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.[16] (2) The neural network is cloned and the parameters are adjusted "to minimize the error

[12]Cf. Silver et al. (2018), p. 1
[13]Cf. Silver et al. (2018), p. 1
[14]Klein (2022), pp. 106-107
[15]Cf. Klein (2022), pp. 100
[16]Cf. Silver et al. (2018) p. 2

between the predicted result [...] and the game result [...]"[17] 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.

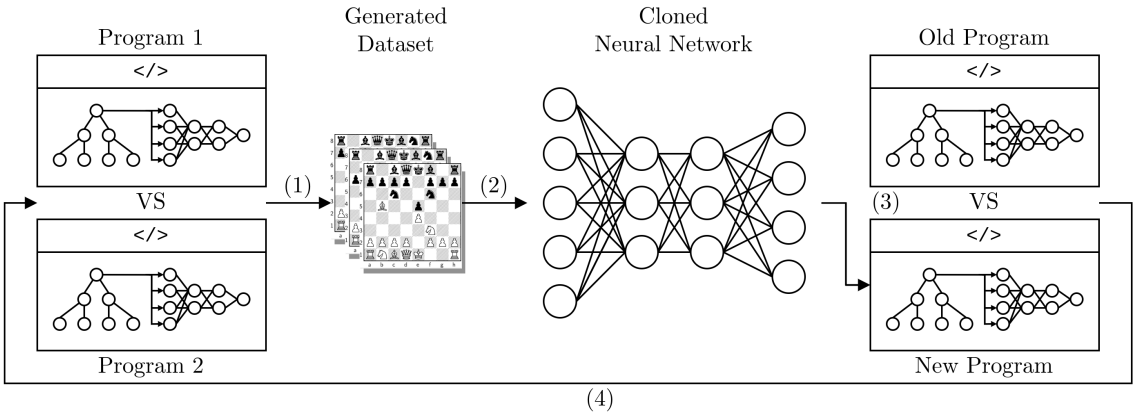

Figure 3: Steps of traning the neural network through self-play

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.[18] For such tasks, encoder-decoder architectures with attention has achieved great success in the past. An architecture proposed by Zang et al. (2019) consists of two parts, an encoder and a decoder, which are two *bidirectional recurrent neural networks (Bi-RNN) using long short-term memory (LSTM) state cells*.

Recurrent neural networks (RNN) are neural networks designed to process sequences of data. RNNs process the data sequences in a time-dependent manner by putting the generated outputs back into the network along with new inputs.[19] This allows the network to remember ongoing patterns in the data and respond to those patterns. State and time are related in recurrent networks because the reused output and the newly added input adjust the values in the RNN accordingly.[20] The state in the network, therefore, represents the values of the neurons in the network at a given time. Bidirectional recurrent neural networks are a special type of recurrent networks that consider both the previous and subsequent input data, unlike traditional RNNs that consider only the previous input data.[21] This is done by splitting "the state neurons of a regular RNN in a part that is responsible for the positive time direction (forward states) and a part for the negative time direction (backward states)"[22]. This enables them to remember and respond to ongoing

[17] Cf. Silver et al. (2018) p. 2
[18] Cf. Sutskever et al. (2014), p. 1
[19] Cf. Lipton et al. (2015), p. 2
[20] Cf. Lipton et al. (2015), p. 8
[21] Cf. Salehinejad et al. (2018), p. 1
[22] Schuster and Paliwal (1997), p. 2674

patterns in both directions along the sequence. One problem of RNNs and Bi-RNNs is the processing of long input sequences, where the so-called "vanishing gradient problem" can occur, whereby the network is no longer able to correctly consider the information from the past. This can be solved by recurrent networks that use a long short-term memory architecture. LSTM cells are special neurons that allow the network to remember and respond to ongoing patterns over long periods of time.[23] This makes them particularly well suited for processing sequences where patterns persist over an extended period of time.

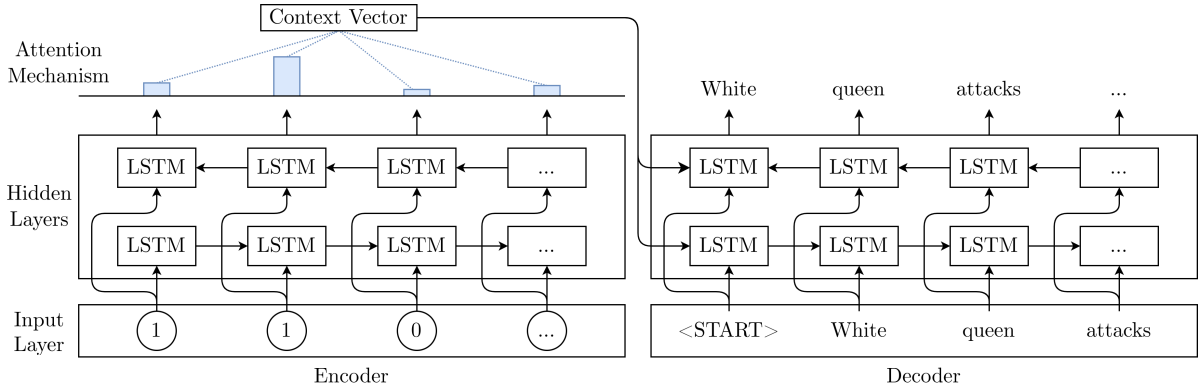

Figure 4: Chess Commentator Encoder-Decoder Architecture with Attention Mechanism (Note: Figure based on Jhamtani et al. (2018) Figure 4)

The encoder-decoder architecture used for generating chess commentary makes use of the described Bi-LSTMs.[24] It's a model used in machine translation and other areas of artificial intelligence. It consists of two main components: the encoder and the decoder. The encoder is responsible for converting the input into an appropriate representation (e.g., a vector of numbers). The decoder uses this representation to produce the desired output.[25] In the case of the chess commentary translation the encoder receives a sequence of data, generated by the chess engine (position and move), processes the data and encodes it into a "fixed-length vector representation"[26]. This vector are the last states of the encoder. The vector can either be used as initialization for the LSTM cells of the decoder or as additional input for each step of the generation of the output.[27] One problem that affects the quality of the decoder's translation, is the length of the sequences. The sequence stored in the vector tends to dilute over time, resulting in poor translations. To solve it, an attention mechanisms can be used. The attention mechanism allows the decoder to "selectively focusing on parts of the source"[28] while producing the desired output. This is done while processing the data in the encoder, which encodes only the most important information in the vector, called the context vector. It can be said that the context vector contains a summary of the sequence. This allows the decoder to focus only on the information relevant for translation. If the decoder now wants to generate comments, it receives a start symbol as input (represented by `<START>` in Figure 4). This start symbol is used to indicate the start of the output sequence. The first output is the first word of the translation. The output is additionally used as new input in the

[23] Cf. Lipton et al. (2015), p. 17
[24] Cf. Zang et al. (2019), p. 2
[25] Cf. Cho et al. (2014), p. 1
[26] Cho et al. (2014), p. 1
[27] Cf. Mohajerin and Waslander (2017), pp. 1-2
[28] Luong et al. (2015), p. 1

next step, generates an output and which is used as the new input again. This procedure is repeated until an end symbol is reached, which signals the end of the sequence. The generated output sequence is the translation, so here the chess comments.

2.2.2 Generation Models

For a given chess position and a played move, the commentator should create comments on different categories. Those categories are the *description of the current move*, the *description of the move quality*, the *comparison of moves*, the *description of the move planning* and *contextual game information*.[29;30]

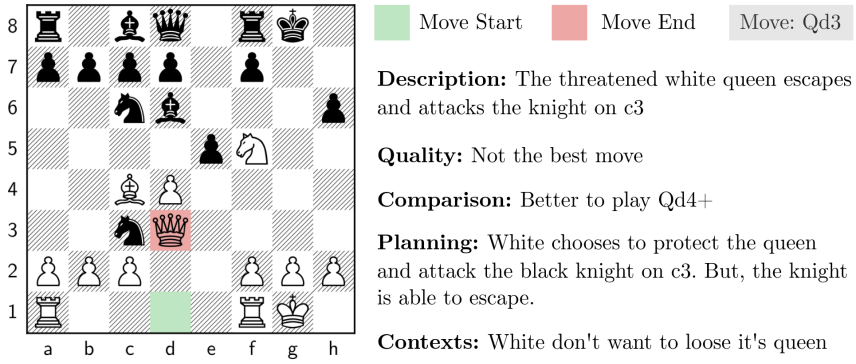

Figure 5: Chess Commentary Example (Note: Figure based on Zang et al. (2019) Figure 1)

Since the procedure for creating comments varies from category to category, it is practically impossible to do this with a single commentator, as simplified described above. Therefore, in the following it will be spoken of five individual ones, each for one of the categories. In addition, instead of categories, it is now spoken of generation models, since these are used to generate comments. These generation models come with two challenges that can have a strong impact on the quality of the generated description: (1) the features on the basis of which the comments for the aspects are generated, (2) the number of moves and the resulting position considered. To overcome the first challenge Jhamtani et al. (2018) used "discrete information (threats, game evaluation scores, etc.)"[31]. How Zang et al. (2019) show, those features information did not provide the best results. Therefore, they have used features provided directly by the internal chess engine. These are the *state of the board before the move*, *start square of the move*, *end square of the move*, *piece on the start square*, *piece on the end square*, *promotion state* and *checking state*. The pieces on the starting square and on the ending square can be different, since the pawns can be replaced by another piece when they reach the opponent's last rank.[32] In this case the promotion state would be set to 1. The advantage of using these features is that they can be easily read from the information provided by the chess engine described in the previous section. These features can be summarized under the terms position and move, which is why in the following only those two terms are used to described all the features. The second challenge can be overcome by distinguishing between generation models that need only one move (description) and generation models that need multiple moves (comparision, planning and context) so that the description is accurate. Depending on the model, they implement either a single-move encoder (SME)

[29]Cf. Jhamtani et al. (2018), p. 1663
[30]Cf. Zang et al. (2019), pp. 3-4
[31]Zang et al. (2019), p. 4
[32]Cf. Levy et al. (1989) p. 10

or a multi-move encoder (MME). An execption is the quality model, which doesn't need any move to create description and therefore implements a completly different encoder. These encoders differ in their training so that they can precisely store different amounts of information in the context vector.

The process of encoding and decoding can be represented as a function. f_{Decode} is the decoder, f_{SME} the single move encoder, f_{MME} the multi move encoder and f_{Encoder} the encoder of the quality model. Encoding and decoding are mapped to an output C. This corresponds to the respective comment of a model. This representation helps to understand what information is needed for the translation of the respective translation models. The description model should provide a move explanation for a position b_0 and a move m_0 played in this position. The process of encoding and decoding can be formulated as $f_{\text{Decoder}}(f_{\text{SME}}(b_0, m_0)) \rightarrow C_{\text{Description}}$[33]. The quality model should describe how good a move was. Therefore it needs three information on basis of which the the commentary is generated. Those are the board before the move, the board after the move and the winning rate difference. For every position evaluated by the chess engines neural network it outputs a value which describes the winning chances of a player (denoted with v in Figure 2). The winning rate difference is the difference between the value before and after the move. This can be formulated as $f_{\text{Decoder}}(f_{\text{Encoder}}(b_0, b_1, v_1 - v_0)) \rightarrow C_{\text{Quality}}$[34]. The comparision model compares two moves. This requires the move played and the resulting position, as well as a move to compare and it's resulting position. If a player plays a move m_0 and it is not the best move, the engine returns the best move m_1 for comparison, as well as the resulting positions b_1 and b_2. If the best move has already been played, the second best move is used for comparison. Since this involves two moves and positions, the multi-move encoder is used. This can be formulated as $f_{\text{Decoder}}(f_{\text{MME}}(b_1, m_0, b_2, m_1)) \rightarrow C_{\text{Comparision}}$[35]. To generate comments for the depth analysis, the planning model receives probable moves played after the current move as well as the resulting positions. $f_{\text{Planning}}(f_{\text{MME}}((b_2, m_1), (b_3, m_2), (b_4, m_3), ...)) \rightarrow C_{\text{Planning}}$[36]. Lastly, comments are to be generated for the entire situation. The planning model can simply be extended by the played move, i.e. $f_{\text{Planning}}(f_{\text{MME}}((b_1, m_0), (b_2, m_1), (b_3, m_2), (b_4, m_3), ...)) \rightarrow C_{\text{Planning}}$[37].

Finally, the five generation models still need to be trained. For this purpose, Jhamtani et al. (2018) compiled a dataset of 11,578 annotated chess games, further used by (Zang et al., 2019). they divided the dataset into three sub-datasets for training, validation and testing, in the ratio 7:1:2. In both cases, training with the data set and the generation models described above showed good results.

3 Conclusion

[33]Cf. (Zang et al., 2019), p.3
[34]Cf. (Zang et al., 2019), p.4
[35]Cf. (Zang et al., 2019), p.4
[36]Cf. (Zang et al., 2019), p.4
[37]Cf. (Zang et al., 2019), p.4

Glossary

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

Stockfish Most use free and open source Sachach engine. 4

Acronyms

Bi-LSTM Bidirectional Long short-term memory.

Bi-RNN Bidirectional Recurrent Neural Network.

LSTM Long short-term memory.

MCTS Monte Carlo tree search.

MME Multi-Move Encoder.

RNN Recurrent Neural Network.

SME Single-Move Encoder.

References

Allis, L. V. (1994, July). *Searching for Solutions in Games and Artificial Intelligence*. Ph. D. thesis, University of Limburg.

Boskovic, B., S. Greiner, J. Brest, and V. Zumer (2005). The representation of chess game. In *27th International Conference on Information Technology Interfaces, 2005.*, pp. 359–364.

Cho, K., B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation.

Cho, K., B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR abs/1406.1078*.

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.

Keen, B. (2009, November). A history of computer chess. In *ECM3401*, pp. 1–14.

Klein, D. (2022). Neural networks for chess.

Levy, D., D. Broughton, and M. Taylor (1989). The sex algorithm in computer chess. pp. 10–21. ICCA Journal, Vol. 12, No. 1.

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

Lipton, Z. C., J. Berkowitz, and C. Elkan (2015). A critical review of recurrent neural networks for sequence learning.

Luong, M.-T., H. Pham, and C. D. Manning (2015). Effective approaches to attention-based neural machine translation.

Mohajerin, N. and S. L. Waslander (2017). State initialization for recurrent neural network modeling of time-series data. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2330–2337.

Salehinejad, H., S. Sankar, J. Barfett, E. Colak, and S. Valaee (2018). Recent advances in recurrent neural networks.

Schuster, M. and K. K. Paliwal (1997). Bidirectional recurrent neural networks. *IEEE Trans. Signal Process. 45*, 2673–2681.

Segundo, P. S. and R. Galán (2005). Bitboards: A new approach. In M. H. Hamza (Ed.), *IASTED International Conference on Artificial Intelligence and Applications, part of the 23rd Multi-Conference on Applied Informatics, Innsbruck, Austria, February 14-16, 2005*, pp. 394–399. IASTED/ACTA Press.

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.

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: _____