CS559 Project Final Report
Chesss Knowledge Extraction From Games of Grandmasters

Doruk Çakmakçı
Computer Engineering Department
Bilkent University
Ankara, Turkey 06800
ID: 21502293

Furkan Özden
Computer Engineering Department
Bilkent University
Ankara, Turkey 06800
ID: 21502397

Abstract*—The game of chess has been a great challenge for the AI community since the beginning of AI research. Starting with the controversial win of the DeepBlue chess program against Garry Kasparov in 1997, AI research took the problem of playing chess at super-human level as a challange and continued to develop on the field. Recently, deep reinforcement learning paradigms are applied to the problem and got very astonishing results. However, most of the deep learning architectures applied to the problem suffers from interpretability issues from our perspective to the problem. However, generally that is not considered to be one of the goals of the challenge and frequently overlooked. In this work, we wanted to address on this issue and possibly propose an appropriate strategy regarding the side goal of interpretability. With this architecture, we try and investigate the playing characteristics intrinsic to the game of chess. Source codes to our model can be found at: *github.com/dorukcakmakci/circumspectus

Introduction

During the last decades, the game of chess has been a major topic of interest to research on artificial intelligence and computer science. Researchers mainly tried to understand the intrinsic decision-making process behind playing chess and a vast majority of work have been made towards this goal. In early years of research on this area, human-level decision making process is first successfully approximated, even surpassed, by rule-based systems such as Stockfish. While rule-based systems are perfected for this task over the years, availability of vast amount of computing power and chess data, turned the tide towards using data-driven approaches, such as deep learning [1], to the same problem. Deep learning induced a phenomena of using of representation learning based approach towards a problem instead of hand-extracted features, given a large amount of data. From another perspective, deep reinforcement learning enabled chess playing computer programs to be trained by playing chess against itself instead of utilizing a large dataset of predefined chess game strategies, as in rule-based systems [2]. When chess playing automatons utilizing rule-based systems are challenged against deep learning based approaches, it is seen that the latter is at least as successful as the former. Hence, deep learning based approaches are able to understand intrinsic properties of chess games and underlying decision-making process given an appropriate representation of chess positions and a suitable training environment.

Given that deep learning approaches are able to understand intrinsics of chess at least as needed for playing chess games, we wanted to analyze if deep learning is suitable for discriminating between different playing styles. For this task we used 12 grandmasters some of which were coined by the NY Times as ”swashbuckling attacker” or ”turning defense into attack”. Furthermore, instead of blindly classifying these grandmasters by their games, we wanted to develop a a system which can also project its inferences back to the chess domain. In brief, for this project, we aimed to utilize deep learning to classsify games of grandmasters and infer a set of characteristic moves that comprise the playing style of the given player.

Design

Chess games can be described as a sequence of board positions in which white and black sides move a piece in an alternating fashion (e.g. first white side moves their piece then black side moves a piece). We have designed a system which takes a sequence of board positions as input, then produces a probabilities for classifying the game as played by each 12 grandmasters (i.e. 12 probabilities). As an intermediate step the system produces an attention signal over the board position sequence to induce a relative importance over board positions (e.g. attention). We used games of grandmsters in which grandmasters are white players in order to avoid confusion regarding move ordering. Note that chess games consists of positions that are described in FEN format (i.e. a string representation of positions). Each chess position is converted to bitmap representation (a tensor of size of size $8 \times 8 \times 7$ describing a single board position). Then chess positions belonging to the same game are given to the system as a sequence.

In order to solve the mentioned task, we have designed a two stage system which utilizes two distinct neural networks

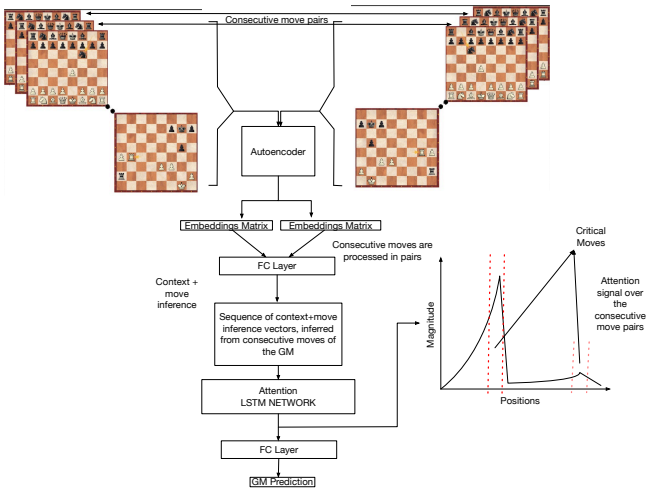


Figure 1: System Design

that work together:

- An autoencoder to embed any given chess position to a lower dimensional space. Position embeddings are expected to preserve critical information regarding the original position while being able to perform lossless reconstruction.
- An Attention-LSTM based neural network that processes sequence of chess position embeddings to classify the grandmaster that plays the sequence of positions while producing an attention signal over the sequence of positions to induce relative importance to board positions.

Overall system design can be seen in Figure 1.

Autoencoder Network

Autoencoder network contains a encoder composed of fully-connected layers, to project input chess positions to a lower dimensional space, followed by a decoder also composed of fully-connected, which tries to reconstruct the original chess position from the lower dimensional projection. Two parts are trained jointly and no sparsity is induced on lower-level representation. The network can be seen in Figure 2. The network uses a flattened bitmap representation of chess board positions as input. Bitmap representation of a board position is a $8 \times 8 \times 7$ sized tensor which holds information for reconstructing the board position in FEN notation. We have also experimented with a convolutional autoencoder architecture for chess board reconstruction. Applying convolutions on the first two dimensions of bitmap representation is observed to be a valid approach, but we continued with fully-connected version since we achieved a lower loss value.

Attention-based LSTM Network

In order to design an input format for our attention-LSTM based model, we have tried two different approaches.

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 448)               0
_________________________________________________________________
dense_1 (Dense)              (None, 300)               134700
_________________________________________________________________
dense_2 (Dense)              (None, 200)               60200
_________________________________________________________________
dense_3 (Dense)              (None, 150)               30150
_________________________________________________________________
dense_4 (Dense)              (None, 42)                6342
_________________________________________________________________
dense_5 (Dense)              (None, 150)               6450
_________________________________________________________________
dense_6 (Dense)              (None, 200)               30200
_________________________________________________________________
dense_7 (Dense)              (None, 300)               60300
_________________________________________________________________
dense_8 (Dense)              (None, 448)               134848
=================================================================
Total params: 463,190
Trainable params: 463,190
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Autoencoder Model

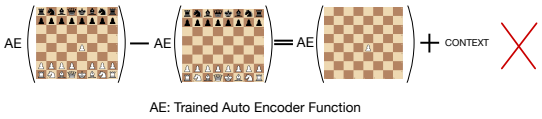


Figure 3: Our first and unsuccessful attempt for defining an input format for our attention-LSTM based model

First we expected the projected chess position embedding space to have the subtraction property where difference between two consecutive board embeddings give chess move + context of the move. This approach, shown in Figure 3, did not fulfill our expectations. Then we have tried direct subtraction of consecutive boards in order to represent any given move. However that did not work as well. Since we could not represent move information extracted from two consecutive boards with subtraction of board encodings, we tried to inserting a neural network instead of subtraction operation. We expect this neural network to learn to combine the consecutive boards in a way such that it learns to extract move information with its context. This approach is visualized in Figure 4. We concluded that a sub-network trained jointly with Attention-LSTM is able to learn an useful mapping from two consecutive board positions. Our overall model can be seen in Figure 5. Having represented sequence of chess moves with the approach mentioned, we have constructed the attention-LSTM network using many-to-one attention. The attention mechanism utilizes general attention variant of scoring mechanisms proposed by Luong et al. [3]. Computations involved in attention mechanism is

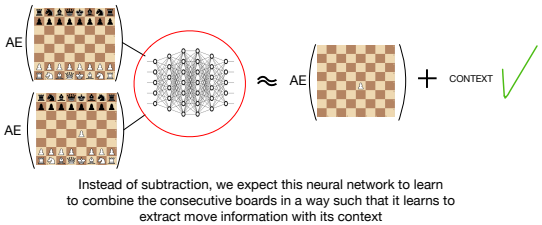
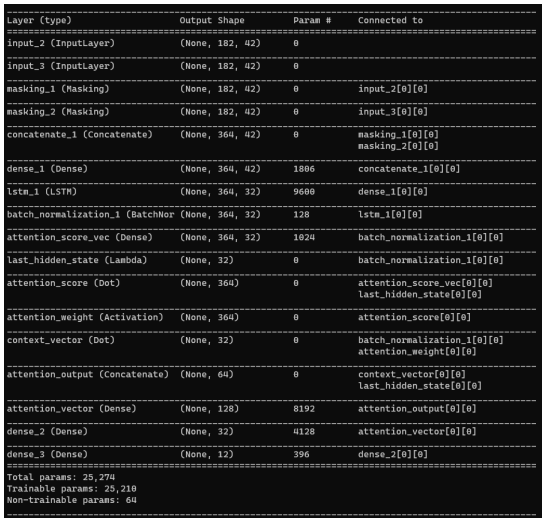


Figure 4: Our second and successful attempt for defining an input



Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 182, 42)	0	
input_3 (InputLayer)	(None, 182, 42)	0	
masking_1 (Masking)	(None, 182, 42)	0	input_2[0][0]
masking_2 (Masking)	(None, 182, 42)	0	input_3[0][0]
concatenate_1 (Concatenate)	(None, 364, 42)	0	masking_1[0][0] masking_2[0][0]
dense_1 (Dense)	(None, 364, 42)	1806	concatenate_1[0][0]
lstm_1 (LSTM)	(None, 364, 32)	9600	dense_1[0][0]
batch_normalization_1 (BatchNor	(None, 364, 32)	128	lstm_1[0][0]
attention_score_vec (Dense)	(None, 364, 32)	1024	batch_normalization_1[0][0]
last_hidden_state (Lambda)	(None, 32)	0	batch_normalization_1[0][0]
attention_score (Dot)	(None, 364)	0	attention_score_vec[0][0] last_hidden_state[0][0]
attention_weight (Activation)	(None, 364)	0	attention_score[0][0]
context_vector (Dot)	(None, 32)	0	batch_normalization_1[0][0] attention_weight[0][0]
attention_output (Concatenate)	(None, 64)	0	context_vector[0][0] last_hidden_state[0][0]
attention_vector (Dense)	(None, 128)	8192	attention_output[0][0]
dense_2 (Dense)	(None, 32)	4128	attention_vector[0][0]
dense_3 (Dense)	(None, 12)	396	dense_2[0][0]

Total params: 25,274
Trainable params: 25,210
Non-trainable params: 64

Figure 5: Attention-LSTM based architecture

as follows: Attention weights are computed over outputs of LSTM using softmax and general scoring function, as given in Equation (1). Attention weights per output is computed as in Equation (2). Context vector is generated as a weighted combination of attention weights and LSTM outputs, as given in Equation (3). Finally, attention vector is computed as in Equation (4). We used a multiplicative attention mechanism to directly induce an importance over all timestep outputs generated by LSTM. The LSTM layer processes move+context representations sequentially and predicts the GM that played the game. The attention signal generated describes the most important moves in that game that characterizes the playing style of that GM.

$$score(h_t, \bar{h}_s) = h_t^T W \bar{h}_s \tag{1}$$

$$\alpha_{ts} = \frac{exp(score(h_t, \bar{h}_s))}{\sum_{s'=1}^{S} exp(score(h_t, \bar{h}_{s'}))} \tag{2}$$

$$c_t = \sum_s \alpha_{ts} \bar{h}_s \tag{3}$$

$$a_t = f(c_t, h_t) = tanh(W_c[c_t; h_t]) \tag{4}$$

Dataset

We have used two different datasets for this project due to two different problems we anticipated that might arise:

- Dataset with chess games of grandmasters [4] has around 25000 games. This many data is not enough to train an autoencoder which is designed to embed chess positions to a lower dimensional space where information about all critical positional advantages are preserved.
- The data to train the autoencoder model must only be valid positions since we expect our autoencoder model to generalize well and perform lossless chess position reconstructions. Hence, we have used a much larger dataset [5] which contains around $2,000,000$ chess positions.

Data Preprocessing

Chess games from both datasets mentioned above are in FEN format [6]. Conventionally, computer chess has used bitboard representations [7] for these FEN formats in order to be able to represent any given chess position with an integer vector. Generally, bitboard representations of chess boards are sparse vectors, with very high dimensionality (e.g. 773d, 448d). These vectors are only conditional if-else embeddings of the positions and can not be considered as a feature extraction of the chess positions.
All chess positions from both datasets are (i) converted to sparse bitboard vector representations; (ii) used a bitboard converter algorithm from Python-Chess package to represent each position in 448 dimensional bitboard vectors, note that we do not take care of castling rights and *en passant* [8] to work with lower dimensional bitboard vectors for ease of autoencoder training.

Dataset for Autoencoder Neural Network Training

The dataset [5] consists of 2,282,519 chess positions in FEN notation. We first convert the data to bitboard notation. Then, divide the dataset to training and test partitions of sizes 2,000,000 and 282,519 respectively for autoencoder training.

Dataset for Attention-LSTM based Neural Network Training

The dataset [4] contains 21587 games from 12 grandmasters. Grandmasters play the white side in 11263 of the games in the dataset. We only use games in which grandmasters are white players in order to avoid confusion regarding move ordering. These grandmasters and corresponding number of white-sided games are: Alekhine (1377), Anand (942), Botvinnik (1328), Capablanca (154), Carlsen (1171), Caruana (545), Fischer (990), Kasparov (1289), Morphy (431), Nakamura (530), Polgar (1795) and Tal (711). It can be seen that the dataset is imbalanced.

Experimental Setup

In order to perform our experiments we have randomly divided the datasets mentioned above to training, validation and test datasets. As mentioned above, we have used a larger dataset of chess positions for autoencoder training. We randomly sampled $2,048,000$ instances without replacement as the training dataset and used the remaining part as the test dataset for autoencoder training. For attention-LSTM model training, we only used the games in which grandmasters played the white side. Training, validation and test datasets are splitted randomly and in a stratified manner using 70%,

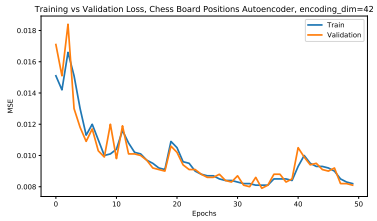


Figure 6: Loss graph of for the training of our autoencoder model

20% and 10% of the dataset respectively. The autoencoder model is trained for 50 epochs on GPU using mean-squared error loss, tanh activation function and Adam optimizer are used. On the other hand, attention-LSTM based model is trained for 15 epochs on GPU using weighted categorical cross entropy as loss, ReLU as activation and Adam as optimizer. Also, a masking layer is used for dealing with variable sized chess games. A batch normalization layer is used between LSTM outputs attention mechanism.

All considered deep learning models are implemented in Python language using Keras framework. All models are trained and tested on a Dell XPS 9570 with an 8th-gen Intel Core i7-8750H processor, 16GB RAM and an Nvidia GeForce GTX 1050 Ti Max-Q GPU(4GB). For a detailed explanation about task division per frameworks and libraries see Appendix A. Also for a division of project work among teammates, see Appendix B.

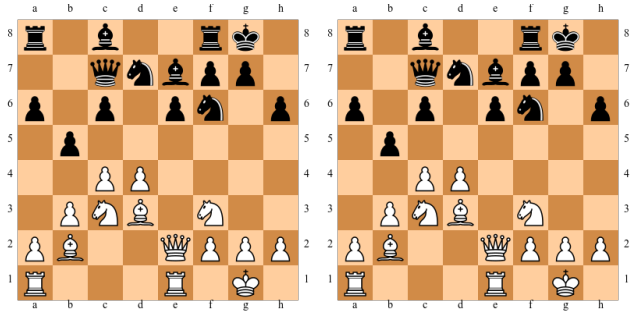
Results

Chess Position Embedding with Autoencoder Network

The autoencoder model in Figure 2 is trained for chess board embedding. Loss graph corresponding to the training can be seen in Figure 6. The model is able to: (i) apply lossless reconstruction to $\sim 80\%$ of test dataset (see Figure 7); and (ii) apply an almost precise reconstruction to $\sim 15\%$ of test dataset (see Figure 8).

Chess Grandmaster Classification with Attention-LSTM Network

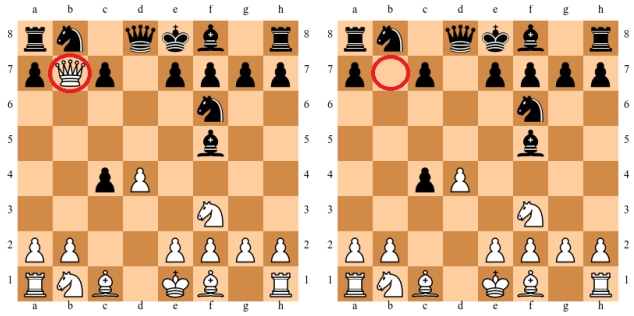
The attention-LSTM based model is trained for classifying games of 12 chess grandmasters. Loss graph corresponding to our training can be seen in Figure 9. Our architecture reaches an accuracy of 27.2% which we think is really impressive since there is an immense amount of variation in playing styles and possible chess games. We think that our model has learned some intrinsics about playing style of grandmasters given that a dummy predictor (i.e. predicting randomly) would reach an accuracy of $\sim 8\%$. Our main findings through examination of the games from the test set are as follows:



(a) Original Position
FEN: r1b2rk1/2qnbpp1/p1p1pn1p/8/1pPP4/1PNB1N2/PB2QPPP/R3R1K1 w

(b) Reconstructed Position
FEN: r1b2rk1/2qnbpp1/p1p1pn1p/8/1pPP4/1PNB1N2/PB2QPPP/R3R1K1 w

Figure 7: Perfect Reconstruction with the autoencoder network



(a) Original Position
FEN: rn1qkb1r/pQp1pppp/5n2/5b2/2pP4/5N2/PP2PPPP/RNB1KB1R b

(b) Reconstructed Position
FEN: rn1qkb1r/p1p1pppp/5n2/5b2/2pP4/5N2/PP2PPPP/RNB1KB1R b

Figure 8: Almost Perfect Reconstruction with the autoencoder network

- Model can discriminate between obvious playing styles: (i) Aggressive playing; and (ii) Defensive playing.
- Attention signals are significant almost always after the first 8 moves. This is expected since the first 8-10 moves are almost all theoretical moves and they are called openings in chess. During openning phase of a game, playing styles of chess grandmasters can not be reflected, they put their style after openings as expected.

Interpreting Attention Signals

According to many other GMs and chess authorities, GM Mikhail Tal is believed to be the most aggressive chess players of all time [9]. Also, GM Garry Kasparov is known to be the greatest defensive player of all time [10]. Even NY Times contains articles about Tal and Kasparov, which coins their playing styles as aggressive and defensive respectively. Some parts of these articles are given in Figures 10 and 11,

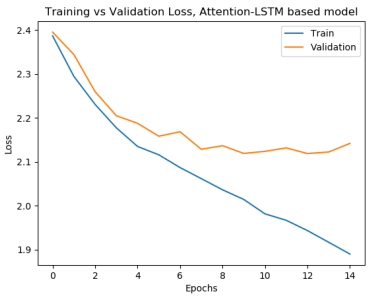


Figure 9: Loss graph of for the training of our Attention LSTM model

see Appendix C. We see that the most obvious difference between playing styles of GMs is aggressiveness. Our model mostly discriminates between the playing styles and most of the correct predictions belong to either Mikhail Tal or Garry Kasparov who are known to be the greatest of two edges. Furthermore, in a surprising manner, the attention weights from the games of these two players. For example, the move shown in Figure 12, see Appendix C, by Mikhail Tal is a very aggressive move sacrificing a piece (bishop) for a pawn in order to get positional advantage. This move, very Tal like move was captured by the model with the highest attention weight. Another example of surprising intrinsics captured by attention weights is given in Figure 13 of Appendix C. In this move, Kasparov demonstrates his ability to turn defense into attack (check the NY Times article 11 given in Appendix C). Notice how the pawn on A3 is attacked by black Queen. Kasparov not only defends his pawn by the move shown, he also attacks to the poorly defended black pawn on A5. Thus, this move turns the defensive gameplay into an attack Kasparov style.

Comparison with the State-of-the-art Methods

During our literature analysis we could find any other proposed methodology to classify GMs and do inference on hte move sequences about the playing characteristics. However, we found one project on predicting professional players' next move with deep learning (i.e. it could be any professional player). They achieved an accuracy of 28% [11].

Appendix A (Use of Software Libraries)

For chess game data processing we mainly used python-chess library [12]. Some preprocessing scripts are taken from an article [11]. For generating sequences of board positions associated with variable length games, we also used some functionality in python-chess library (i.e. chess games are parsed from corresponding PGN file, an initial board representation is initialized and the game is played from start to end while saving the board positions). Also, pandas [13] library is used to neatly access meta-data about the games provided in the dataset.

For the deep learning models, Keras [14] library is used. The Functional API is used to develop each model. Also for aggregating attention to our model, we first implemented dot-product attention mechanism ourselves. Then we found out that there is simple and neat attention layer implementation on GitHub [15] with over 2K stars. We continued with this neat implementation and used general attention scheme proposed by Luong et al. [3].

Appendix B (Contributions)

Contributions of each group member is given below:

- Furkan Özden: Designed and implemented the model. Wrote the scripts regarding preprocessing of the chess data. Wrote the required reports for the project.
- Doruk Çakmakçı: Helped the design. Also, implemented the attention part of the model. Wrote the result producing and analysis scripts for the project. Wrote the required reports of the project.

Appendix C (Supplementary Information)

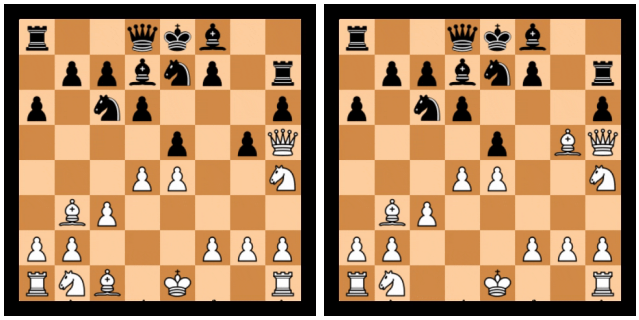
Figure 10: A segment of NY Times article devoted to Tal, which highlights his chess playing characteristics as aggressive

Figure 11: The headline of NY Times article devoted to Kasparov, which highlights his chess playing characteristics as defensive

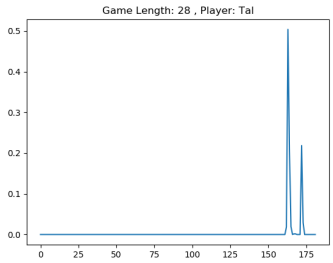
References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.



(a) Board position before move (b) Board position after move



(c) The attention signal associated with this game

Figure 12: A sample interpretation of attention signal generated by the model. In this case, Tal made an aggressive move by sacrificing his rook.

[3] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[4] L. Renyu, "Chess games from 12 top players," Jan 2020. [Online]. Available: https://www.kaggle.com/liury123/chess-game-from-12-top-players

[5] A. Stöckl, "Fics chess games," Aug 2018. [Online]. Available: https://www.kaggle.com/astoeckl/fics-chess-games

[6] "Forsyth edwards notation," Aug 2019. [Online]. Available: https://en.wikipedia.org/wiki/Forsyth\OT1\textendashEdwards_Notation

[7] "Bitboard," Mar 2020. [Online]. Available: https://en.wikipedia.org/wiki/Bitboard

[8] "En passant," Feb 2020. [Online]. Available: https://en.wikipedia.org/wiki/En_passant

[9] Robert, "Mikhail tal, a chess grandmaster known for his daring, dies at 55," Jun 1992. [Online]. Available: https://www.nytimes.com/1992/06/29/us/mikhail-tal-a-chess-grandmaster-known-for-his-daring-dies-at-55.html

[10] R. Byrne, "Kasparov's style: Turning a defense into an attack," Aug 1988. [Online]. Available: https://www.nytimes.com/1988/08/14/style/chess-kasparov-s-style-turning-a-defense-into-an-attack.html

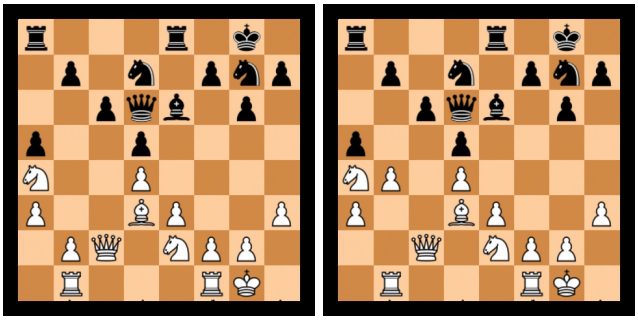
[11] A. Stöckl, Apr 2019. [Online]. Available: https://medium.com/datadriveninvestor/reconstructing-chess-positions-f195fd5944e

[12] Niklasf, "niklasf/python-chess," Jun 2020. [Online]. Available: https://github.com/niklasf/python-chess

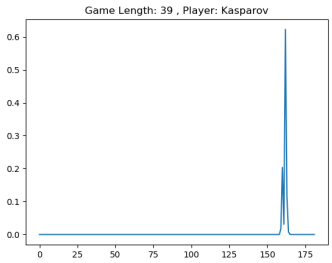
[13] "pandas." [Online]. Available: https://pandas.pydata.org/

[14] K. Team, "Simple. flexible. powerful." [Online]. Available: https://keras.io/

[15] Philipperemy, "philipperemy/keras-attention-mechanism," May 2020. [Online]. Available: https://github.com/philipperemy/keras-attention-mechanism



(a) Board position before move (b) Board position after move



(c) The attention signal associated with this game

Figure 13: Another sample interpretation of attention signal generated by the model. In this case, Kasparov turned defense into attack by moving his pawn to a tile where he performed an attack while defending the played pawn.