

Patrik Hammersborg

Explainable AI approaches for deep reinforcement learning agents in a high performance chess environment

Master's thesis in Computer Science

Supervisor: Inga Strümke

May 2023



Norwegian University of
Science and Technology

Patrik Hammersborg

Explainable AI approaches for deep reinforcement learning agents in a high performance chess environment

Master's thesis in Computer Science

Supervisor: Inga Strümke

May 2023

Norwegian University of Science and Technology

Faculty of Information Technology and Electrical Engineering

Department of Computer Science



Norwegian University of
Science and Technology

Abstract

How does one best explain the learned knowledge of chess-playing neural network models? With strong neural network models quickly becoming the state of the art within the domain of chess, it has become increasingly interesting to investigate what such models have learned. This thesis aims at presenting a less computationally expensive alternative to training strong neural network models for chess, by training chess-playing agents through self-play using reinforcement learning. The thesis also presents the application of existing explanation methods to the resulting models, and the development of specialised methods for explaining chess-playing neural network models.

While there are existing implementations of standard 8x8 chess environments and large open-source versions of chess-playing agents available, most explanatory methods are quite computationally expensive to perform on large models. Additionally, having access to all parts of a complete training pipeline greatly facilitates creating explanatory methods that require training any given model from scratch. The required computational savings presented in this work are mainly achieved by operating on smaller variants of chess, which is made possible by implementing a customisable and high-performance chess environment. This environment is then coupled with an effective procedure for deep reinforcement learning (DRL) through Monte Carlo Tree Search. The resulting pipeline is then used for training of strong neural network models for 4x5 and 6x6 variants of chess.

Models produced by the custom training pipeline are then explained using various modern methods from the field of Explainable Artificial Intelligence (XAI). This includes searching for domain knowledge accrued by the models through concept detection probing, the results of which can be compared to other established results for larger models.

This work also presents novel explanatory methods for neural network based chess agents. Firstly, a method for visualising the model's comprehension of a given concept through concept maximisation is presented. Secondly, a method for providing global explanations for a given model through observing the correlations between probed concepts and predicted game outcome is described. This also allows for analysis of the selected 4x5 and 6x6 chess variants. Thirdly, the work presents an approach for gauging the information content in intermediary layers using binarised neural networks. Finally, alternative adaptations of the well-studied method groups of counterfactual explanations and saliency maps to the game of chess are presented.

Sammendrag

Hvordan kan man forklare kunnskapen som sjakkspillende nevralt nettverk opparbeider seg? Dominansen av sterke nevralt nettverksmodeller innenfor sjakk har gjort det stadig mer interessant å undersøke hva slags sjakkmessig forståelse slike modeller fremstiller. Denne oppgaven legger derfor frem et mindre beregningstungt alternativ for å produsere kompetente sjakkspillende nevralt nettverksmodeller, ved å trene slike modeller gjennom “reinforcement learning” ved selvspill. Oppgaven presenterer også bruk av eksisterende forklaringsmetoder på de resulterende modellene, samt utvikling av spesialiserte metoder til bruk for å forklare sjakkspillende nevralt nettverk.

Selv om det finnes tilgjengelige implementasjoner av standard 8x8-sjakkmiljø og sjakkspillende nevralt nettverksmodeller, så er eksisterende forklaringsmetoder relativt beregningstunge ved bruk på store modeller. Dessuten vil det å ha tilgang til den fullstendige treningsprosedyren gjøre det mulig å skape forklaringsmetoder som krever opptrening av de aktuelle modellene fra bunnen av. Den nødvendige reduksjonen av beregningskrav blir hovedsakelig oppnådd ved å produsere et tilpassingsdyktig sjakkmiljø med høy simuleringsytelse. Dette miljøet kobles sammen med en effektiv treningsprosedyre som trenes ved “deep reinforcement learning” ved Monte Carlo tresøk. Den implementerte treningsprosedyren brukes deretter for å trene kompetente nevralt nettverksmodeller på 4x5- og 6x6-varianter av sjakk.

Modeller produsert av den nevnte treningsprosedyren kan deretter forklares ved hjelp av moderne forklaringsmetoder fra forklarbar kunstig intelligens som fagfelt. Dette innebærer blant annet å søke etter modellenes opparbeidede domenekunnskap gjennom konseptdeteksjon, der resultatene fra denne metoden kan sammenlignes med etablerte resultater for større modeller.

Denne oppgaven presenterer også nye forklaringsmetoder for sjakkspillende nevralt nettverksmodeller. For det første presenteres det en metode som visualiserer modellenes forståelse av et gitt konsept gjennom konseptmaksimering. For det andre presenteres det en fremgangsmåte som kan gi globale forklaringer for en gitt modell ved å se på korrelasjoner mellom oppdagede konsepter og predikert partiutfall, noe som også tillater en analyse av de valgte 4x5- og 6x6-sjakkvariantene. For det tredje viser denne oppgaven en strategi for hvordan man kan observere informasjonsinnholdet i mellomlag i en gitt modell ved å bytte ut et antall av disse lagene med binære mellomlag. Til slutt presenteres det prosedyrer spesielt tilpasset sjakkmodeller som kan brukes til å fremlegge kontrafaktiske forklaringer og viktighetskart over vilkårlige stillinger.

Preface

The work in this thesis is partially based on preliminary project work in the fall of 2022. Some of the work contained in this thesis was also reworked and has been accepted for publication at the 22nd World Congress of the International Federation for Automation and Control (Hammersborg and Strümke, 2022).

I would like to thank my supervisor Inga Strümke for irreplaceable guidance and support both before, and throughout this thesis work.

My time working on this thesis was made infinitely better by Céline, my beloved.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Research questions	2
1.3	Thesis structure	3
2	Background	5
2.1	Chess	5
2.2	Minimax	6
2.3	Monte Carlo Tree Search	7
2.4	Neural networks	9
2.4.1	Fully connected neural networks	10
2.4.2	Convolutional neural networks	12
2.4.3	Residual neural networks	12
2.4.4	Binarized neural networks	13
2.5	Applying deep learning to tree search	14
2.6	Explainable AI	17
2.6.1	Feature importance attributions	17
2.6.2	Saliency maps	19
2.6.3	Concept detection	20
2.6.4	Counterfactuals	21
3	Methods	23
3.1	High performance chess environment	23
3.1.1	Bitboards	23
3.1.2	Precomputing possible attacks	24
3.2	Training models	26
3.2.1	Model architecture	28
3.2.2	Input/output structure	28
3.2.3	Training architecture	29
3.2.4	Availability	29

3.3	Concept detection	30
3.4	Backpropagating concepts	30
3.5	Binarized intermediate layers	32
3.6	Output correlations	33
3.7	Counterfactual explanations	34
3.8	Heatmap-based explanations	35
4	Results	39
4.1	Concept detection	39
4.2	Concept backpropagation	40
4.3	Binary intermediate layers	40
4.4	Correlations	40
4.5	Heatmap	40
4.6	Counterfactual explanations	41
5	Discussion	49
5.1	Concept detection	49
5.2	Concept maximisation	51
5.3	Shapley values in concept space	52
5.4	Correlations	54
5.5	The viability of counterfactual explanations for chess	55
5.6	Do methods need to be model agnostic?	56
5.7	Information content	56
5.8	What makes for a good explanation?	58
6	Conclusion	59
6.1	Contributions	59
6.2	Future work	60
6.2.1	Architectures for linear probing	60
6.2.2	Methodologies for concept backpropagation	60
6.2.3	Extending the alternative heatmap explanation	60
	Bibliography	61
	Appendices	65
A	A selection of edge-cases for calculating legal moves for a given chess position	67
B	Equation for maximising single probe-output	67
C	Collapsing a n-layered feed-forward, fully connected neural network with linear layers into a single layer	68

D	GradCAM applied to positions used for validating the proposed heatmap method	70
---	--	----

List of Figures

2.1	The starting position for standard chess on an 8x8 board.	6
2.2	Fully connected, 3-layered neural net models with randomly initialised weights and biases, describing a mapping from $[-1, 1]$ to \mathbb{R}^1 . (a) shows such models with a linear activation function, while (b) shows such models with the nonlinear <code>gelu</code> -activation function, $f(x) = x \cdot \frac{1}{2}[1 + \text{erf}(x/\sqrt{2})]$ (Hendrycks and Gimpel, 2016b). Observe that the networks with linear activation functions (Fig. 2.2a) only produce linear mappings, even when multi-layered.	11
2.3	The Heaviside step function (a), and its derivative (b).	14
2.4	A histogram of a bimodally distributed variable. Observe that the mean falls between the two sharp peaks of the distribution.	18
2.5	The main architecture of concept detection, operating on an arbitrary model. Here, the linear probe is inserted at layer L_3	20
3.1	An illustration of representing parts of a chess state using bitboards. This also shows how board sizes smaller than 8x8 (equaling 64 bits in total) fit into an unsigned 64-bit integer.	24
3.2	The possible moves for a rook in the lower-right corner, which can only move in a straight line.	24
3.3	The possible moves, highlighted in blue, for (a) the pawn, (b) the knight, and (c) the king.	25
3.4	The possible moves for (a) the bishop, (b) the rook, and (c) the queen. Diagonal moves are highlighted in red, and straight-line moves in blue. The main observation is that the queen has the movement of a combined rook and bishop.	25
3.5	The two chess-variants used: (a) Silverman 4x5 and (b) Los Alamos 6x6.	27
3.6	The architecture of the 6x6-ResNet model. The structure of the 4x5 model is similar, but without skip-connections and only three initial convolution layers.	27

3.7	An illustration of the binary masks described in Sec. 3.4. In the shown figure, s^- is represented as a 6x6x1 matrix, meaning that it masks over all pieces on the board, while both s and s^+ are represented as 6x6x12-matrices. This means that each combination of piece color and type is given its own plane in s^-	31
3.8	GradCAM applied to two positions of 4x5 chess. Observe that the explanation method only highlights a single piece, which is the piece to be moved for the preferred move.	35
3.9	A set of illustrations showing the masking module produces a reductive mask during training (a), and how that reductive mask is applied to the given training sample (b).	36
4.1	Concepts modelled by the 4x5 chess agent, showing the model's ability to detect (a) whether the player to move has a material advantage, (b) whether the opponent is currently presenting a mate-threat, (c) whether the opponent's queen is under threat, (d) whether the player to move is in check, (e) whether it has a double-pawn, (f) whether the opponent has a double-pawn, (g) whether both players contest an open file on the board, and (h) whether the player to move's queen is threatened, and (i) being a sanity check performed on a data set of random labels.	42
4.2	Concepts modelled by the 6x6 chess agent, showing the model's ability to detect (a) whether the player to move has a material advantage, (b) whether the opponent is currently presenting a mate-threat, (c) whether the opponent's queen is under threat, (d) whether the player to move is in check, (e) whether it has a double-pawn, (f) whether the opponent has a double-pawn, (g) whether both players contest an open file on the board, and (h) whether the player to move's queen is threatened, and (i) being a sanity check performed on a data set of random labels.	43
4.3	Backpropagation of concepts performed on the concepts (a), (b) <code>has_own_double_pawn</code> , (c), (d) <code>in_check</code> , (e), (f) <code>threat_my_queen</code> , and (g), (h) <code>threat_opponent_queen</code>	44
4.4	The first six channels of binary activation patterns for a given position.	45
4.5	Correlation results between concept probes and Q-output using the (a), (b) <code>dcor</code> , (c), (d) <code>r2</code> , and (e), (f) <code>hsic</code> -correlation metric for the 4x5 and the 6x6 model.	46
4.6	Positions with an applied heatmap. The opacity of each piece shows the learned importance per piece for the given position.	47

4.7	Generated counterfactuals for a game of 4x5 chess. All positions are sampled sequentially from the same game, with a random move selection process up to the first position shown in (a).	48
5.1	A typical position occurring under optimal play in the presented 4x5 chess variant. Observe that none of the pawns for either player have any legal moves.	50
5.2	An example position simplified with a heatmap (a), and the same position (b), with no simplifying heatmap.	57
A.1	A position where White's rook (highlighted in red) is pinned to its own king. The moves highlighted in blue are not legal, even though they correspond to the rook's standard pattern of movement. . . .	68
D.1	Positions with an applied saliency map generated by applying the GradCAM method described in Sec. 2.6.2. The opacity of each piece shows the learned importance for each piece for the given position. .	70

List of Tables

- 2.1 A list of the available pieces in chess, and their established material evaluation. The evaluation is given in the units of “pawns”, where the value of a singular pawn is used as the baseline. 7
- 3.1 Description of each input channel of a chess-representation adapted for neural networks. 28
- 3.2 Concept functions used in Sec. 2.6.3. 30

List of Algorithms

1	Backpropagation for neural networks by gradient descent for a set of input-output pairs I, O , loss function $E(\cdot, \cdot)$, model M with intermediate layers L_1, L_2, \dots, L_n with corresponding parameters $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_n$, learning rate (hyperparameter) η and some performance measure $P(\cdot)$ to be applied to the given model with accompanying cutoff-threshold T	10
2	Full training procedure for a MCTS-based game agent, with n search-iterations per move.	16
3	GradCAM for a CNN-based classifier M taking s as input, with last convolutional layer L with filter dimensions (m, n) , and l filters . . .	20
4	Concept backpropagation for a single input sample s , partial model $M_I(\cdot)$ as a map from the input space of s to the activation space of an intermediate layer I in M , pre-trained concept probe P on the activation space I , pre-trained legality-checker $C(\cdot)$ that evaluates the legality of any given position, and a threshold T as to define adequate maximisation of the given concept.	32
5	Search strategy for finding specified binary masks C_i in the binary activation channels B_i with n activation channels in total, for a given input sample I_i , and for a given model M that produces B_i given I_i . This finds the indices of all channels that always match the given binary mask.	33

Chapter 1

Introduction

1.1 Motivation

Historically, chess has been used as a domain for testing and developing methods within Artificial Intelligence (AI). This is evidenced through the development of leading chess engines, such as Stockfish (Romstad et al., 2022), and through general AI methods that learn through self-play, such as AlphaZero (Silver et al., 2018). Chess is also an example of a game where top programs consistently outperform leading human players. Many would claim that this fact was first made evident through the match between IBM’s Deep Blue and current world champion Garry Kasparov, as documented in Campbell et al. (2002). As chess programs, Stockfish and Deep Blue are functionally quite similar, as they are both based on human defined heuristics and extensive tree search. During the last years, however, other approaches with particularly different points of departure have been presented. Among the most prominent of these is AlphaZero (Silver et al., 2018), which is trained using deep reinforcement learning (DRL) and self-play, meaning that the model improves by playing against itself.

While traditional chess engines are strong players, the playing style of DRL-based chess programs might be considered more interesting to humans: It can be argued that DRL-based programs such as AlphaZero have a more human approach to the gameplay, in the sense that the model is intended to learn heuristics by exploration and experimentation. Since AlphaZero reaches superhuman performance, examining and explaining these heuristics could therefore give human-like insights into the game of chess. These might include novel approaches to certain aspects of the game, or reimaginations of already established domain knowledge.

Strategies for explaining learned heuristics in DRL chess models, such as those

presented in McGrath et al. (2021), often carry a significant computational cost. Additionally, the models themselves are exceedingly difficult to produce, requiring vast amounts of self-play games to be performed. This leaves both the methods for creating the agents themselves, and the methods explaining them, out of reach for many research groups. In order to be able to explore new ways of explaining these heuristics, as well as reproducing those shown in McGrath et al. (2021), lower-complexity alternatives must be made available. This would include the ability to create agents that play lower-complexity variants of chess, while remaining adequately complex to inhibit advanced heuristics for play. This thesis therefore presents an alternative for training DRL-based chess agents on smaller chess variants, and an open-source implementation of the concept detection strategy as described in Silver et al. (2018). The thesis also intends to explore and show alternative explanatory methods for extracting learned domain knowledge from trained chess models. Here, the proposed chess environment is essential for being able to create and train the models which these methods are to be applied to.

1.2 Research questions

Goal To explore methods from Explainable Artificial Intelligence (XAI) with regards to how DRL-based state-of-the-art chess models reason when playing chess.

As presented in Sec. 1.1, RL-based chess models are quickly becoming strong adversaries. It would therefore be very interesting to gain insight as to what such models have learned in order to achieve their playing strength.

Research question 1 Is it possible to create an environment with adequate tooling as to recreate AlphaZero (Silver et al., 2018), and their presented XAI-methods (McGrath et al., 2021) with smaller computational requirements?

Since most of the work presented in Silver et al. (2018) and McGrath et al. (2021) is closed-source, and exceedingly computationally demanding, a viable alternative must be presented. While there are some open-source implementations of the pipelines presented in Silver et al. (2018), it is still preferable to create a custom pipeline for this work. This is mainly because creates a possibility to work with smaller chess variants, which in turn makes it possible to re-train models from scratch in the cases where this might be necessary. Additionally, applying existing explanatory methods to very large models, such as AlphaZero, also has a non-trivial computational cost, which is heavily reduced by being able to use smaller models.

Research question 2 Do there exist alternative methods to those presented in McGrath et al. (2021) in order to explain DRL-based chess models?

If one is able to train DRL-based chess models from scratch, and has access to relevant tooling in order to interface with these, this opens up several interesting possibilities with regards to providing novel explanatory methods for the given model. Ideally, this could lead to novel explanation methods beyond those described in McGrath et al. (2021).

1.3 Thesis structure

This thesis is divided into six chapters. Chapter 1 presents an introduction to the point of intersection between chess and AI.

Chapter 2 presents relevant background information and motivates the work and results in this thesis. This includes a brief overview of chess as a game in Sec. 2.1, algorithms used in traditional chess engines in Sec. 2.2 and Monte Carlo Methods and Monte Carlo Tree Search in Sec. 2.3. Additionally, this section provides an introduction to relevant deep learning methodology and neural networks in Sec. 2.4, and how these are applied to MCTS in Sec. 2.5. Chapter 2 also provides an introduction into relevant XAI methods, such as feature importance attributions (Sec. 2.6.1), saliency maps (Sec. 2.6.2), concept detection (Sec. 2.6.3), and counterfactual explanations (Sec. 2.6.4).

Chapter 3 presents the methodology used to answer the research questions posed in Sec. 1.2. Secs. 3.1 and 3.2 include descriptions of the implemented high-performance chess environment, and training AlphaZero-like models using it, and Secs. 3.4 to 3.8 present the proposed explanatory strategies used in this thesis.

The results for each proposed explanatory method are shown in Chapter 4, and discussed in Chapter 5. Chapter 5 also discusses various aspects of each of the proposed methods.

Chapter 6 concludes the thesis, evaluates the proposed research questions, and suggests future work.

Chapter 2

Background

2.1 Chess

Chess is a well-studied turn-based game for two players. Canonical chess takes place on an 8x8 board, where both players possess an identical set of pieces. The goal of the game is to trap the opposing player's king, and the game is won by simultaneously trapping and attacking it – resulting in checkmate. All pieces have discrete movement rules, and the game is always commenced from a standard starting position, see Fig. 2.1.

As a subject of study for computerised game-playing methods, chess has several practical properties. It is a game of perfect information, meaning that no part of the game state is hidden from any of the players during play. Secondly, both its state space and action space are discrete, deterministic, and finite, meaning that it is possible to enumerate all legal actions in any given state. For conventional search methods, this means that it is possible to use the complete enumeration of all child-states to evaluate any given state. This also means that, given infinite computing power, it is in theory possible to provide an optimal strategy by evaluating all possible states, and calculating how these relate to one another. This is however far out of reach by current computational facilities.

Chess has also accrued a vast amount of expert knowledge, with many facets of the game having been studied in great detail. This has resulted in some aspects of the game being accepted as empirical truths, even if no theoretical proofs are available. Among such truths is the notion of the relative evaluation of each piece. These evaluations are shown in Table 2.1. The notion of such knowledge has proven to be quite relevant for computerised chess playing methods. While it can be used and implemented directly, it can also serve as a benchmark for methods that do

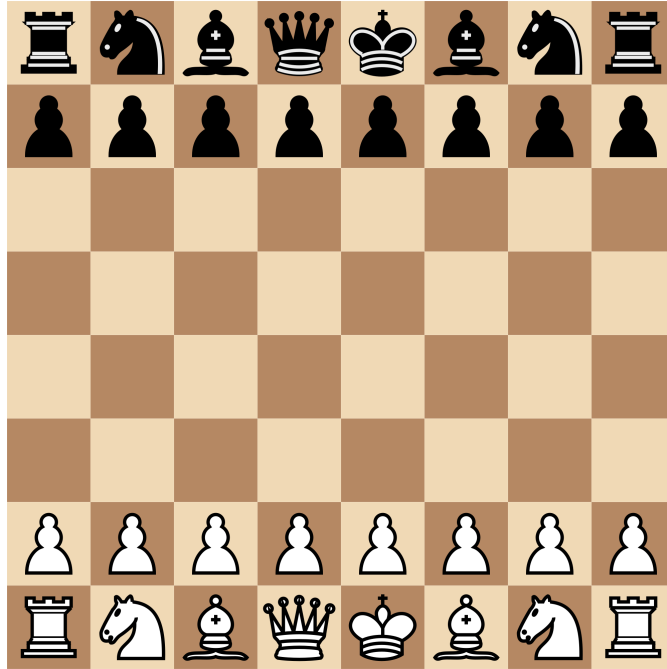


Figure 2.1: The starting position for standard chess on an 8x8 board.

not rely on any knowledge a priori.

2.2 Minimax

Minimax is a decision algorithm that can be applied to any zero-sum game with finite action and state spaces (von Neumann et al., 1944). It can be used to provide a game-value for a given state s by constructing a game tree based on the child nodes of s . For a two-player game with alternating turns, the value of s wrt. the player to move can be expressed as

$$v(s | p) = \max_{\{a\}} v(s_i | p^*), \quad (2.1)$$













where a is the action that leads from state s to s_i , p is the player to move, and p^* is the opposing player. Additionally,

$$v(s | p) = C, C \in \mathbb{R} \quad (2.2)$$

for terminal states. Since the game is zero-sum, it is assumed that

$$v(s | p) = -v(s | p^*). \quad (2.3)$$

Table 2.1: A list of the available pieces in chess, and their established material evaluation. The evaluation is given in the units of “pawns”, where the value of a singular pawn is used as the baseline.

Piece name	Piece graphics	Evaluation
King	 	-
Queen	 	9.0
Rook	 	5.0
Bishop	 	3.0
Knight	 	3.0
Pawn	 	1.0

Eqs. 2.1, 2.2, and 2.3 provide an exhaustive strategy in order to provide an evaluation of any state s . The practical interpretation of these equations is that a player p wishing to play optimally, will choose the action a from s that results in the highest value. Assuming that the opposing player p^* also wishes to play optimally, the value of each child-state s_i is then decided by p^* choosing its action to achieve the highest value from each possible s_i . This continues until the game is terminated, and assigned a value based on the result of the game, as described in Eq. 2.2.

For chess, however, a naive implementation of the minimax algorithm would not be sufficient. This is because it is currently unfeasible to continue this evaluation until reaching all terminal child-states for a given state s . To remedy this, Eq. 2.3 is altered to also apply to select non-terminal states. The constant C for a non-terminal state s is then estimated by applying a heuristic to the state. For chess, common heuristics include for example a weighted sum of the difference in the pieces held by each player, or more abstract concepts, such as the notion of piece mobility. There also exist several extensions to the standard minimax algorithm, such as Alpha-Beta pruning (Knuth and Moore, 1975), and the killer-move heuristic (Akl and Newborn, 1977), all used in state-of-the-art chess engines.

2.3 Monte Carlo Tree Search

For tree search contexts where it is computationally difficult to evaluate all possible child-states from a given state, it is useful to consider alternative strategies. One such strategy is Monte Carlo Tree Search (MCTS), first described in Coulom (2007). Given a state s , this tree search variant seeks to approximate the exhaus-

tive enumeration of the child-states of s with a procedure of gathering a representative sample of these child-states. MCTS is derived from the more general concept of Monte Carlo methods, which concern themselves with approximating results for a given process through large amounts of random sampling (Metropolis and Ulam, 1949). Such methods are useful when working with infinite or continuous action spaces, or in cases where the combined action and state space is so large that it is computationally difficult to work with.

Given a state s , and a set of possible actions $A = \{a_1, a_2, \dots, a_n\}$ leading to other child-states $\{s_1, s_2, \dots, s_n\}$, a general Monte Carlo method repeatedly samples actions from A , applies these to s , and regards the aggregate of the evaluations of the resulting states as an estimate of the true value of s . For MCTS, this sampling strategy is combined with a standard tree search procedure. Here, the tree search is guided by information gathered through the Monte Carlo sampling strategy.

For standard MCTS, for a given state s , the sampling is done by iteratively building a search-tree from the node representing s by maximising

$$Q(s) + U(s, a), \quad (2.4)$$

where $Q(s)$ is the expected value of the state s found incrementally through the MCTS-process, and $U(s, a)$ is a bias towards exploring action a in the state s . U can be any function wrt. s and a , but the polynomial upper confidence bound applied to trees (PUCT), first described in Rosin (2011), is commonly used. This is defined as

$$U(s, a) = c \cdot P(s, a) \cdot \frac{\sqrt{\sum_{a^*} N(s, a^*)}}{1 + N(s, a)}, \quad (2.5)$$

where $N(s, a)$ is a tally of how often the action a has been traversed from s in previous MCTS-iterations, a^* is a possible action from the state s , c is a tunable constant, and $P(s, a)$ is a function designating a prior probability towards performing action a in the state s .¹ In the cases where no such probability is known, this can be set to 1 for all pairs (s, a) .

The tree search method is performed iteratively. At the start of the routine, a search-tree with a single root node is created. Then, for each iteration, one traverses from the root node by maximising Eq. 2.4 until a leaf node is reached. An evaluation is then performed on the leaf node. One can think of this as being equal to “sampling” an outcome from the leaf node. The way this sampling is performed can vary, and it is common to perform a random playout from each such leaf node. This involves creating a copy of the state represented by the leaf-node, and then performing random actions from it until it represents a terminal

¹This can be seen as a way of incorporating expert knowledge into systems using MCTS.

state. This sampled outcome is then used to update $Q(s_i)$ for each state s_i on the path between the root node and the leaf node. The intention of using $Q(\cdot)$ is that $Q(s)$ should return the expected game-outcome of traversing to s . This sampling strategy then ensures that $Q(s)$ reflects the average sampled outcomes from all leaf-nodes which are reached by traversing through s . This path is also used to update $N(s, a)$, by incrementing the tally for all state-action pairs used to traverse from the root node to the leaf node. A concrete implementation of this is discussed in Sec. 2.5.

2.4 Neural networks

Neural networks are a general means for approximating any mapping from an input space I to an output space O . Given an adequately large set of samples of input-output pairs from the function defining the given map, $F : I \rightarrow O$, and a loss function $E(\cdot, \cdot)$ that quantifies some notion of “difference” between two output pairs (O_n, O_m) , a neural network model M aims to minimise the $E(O_n, M(I_n))$ for all such corresponding pairs. That is, it aims to approximate the mapping F , without access to any direct representation of F itself.

A neural network is comprised of layered mathematical structures. Each layer of a neural network serves as a mapping from some n -dimensional space, to some other m -dimensional space, where each such mapping is dependent on a large amount of parameters. One can define each such layer as a mapping

$$F_i : (L_{i-1}, \mathbf{w}_{i-1}) \rightarrow L_i \quad (2.6)$$

where each mapping from L_{i-1} to L_i is dependent on a set of parameters \mathbf{w}_{i-1} . In practice, these parameters are often implemented as n -dimensional, real-valued tensors.

When chaining these layers together, the output space of layer $i-1$ is equivalent to the input space of layer i . In order to serve as universal function approximators, it has been shown that these mappings need to be non-linear (Hornik, 1991). Empirical evidence for this can be seen in Fig. 2.2, and a more formal proof is outlined in Appendix C.

Finding a weight tensor \mathbf{w}_i for each layer L_i in a given neural network can be done using gradient descent. For a given sample-pair (I_n, O_n) , a model M , and a loss function comparing two sample-outputs $E(\cdot, \cdot)$, one computes the gradients of change for the last weight tensor with regards to the outputs. This represents how the elements in \mathbf{w}_n should be modified in order to minimise the loss for the given sample. This calculation is performed for each layer, where the gradients of

\mathbf{w}_i depends on the gradients of \mathbf{w}_{i+1} . Small increments of change are then applied for each sample pair, which in turn should produce weight tensors that minimise the given loss function. This is often done stochastically in mini-batches, where these calculations are performed simultaneously over a randomly sampled batch of input-output pairs. The process of backpropagation in order to approximately determine all \mathbf{w}_i is described in Algorithm 1.

Algorithm 1 Backpropagation for neural networks by gradient descent for a set of input-output pairs I, O , loss function $E(\cdot, \cdot)$, model M with intermediate layers L_1, L_2, \dots, L_n with corresponding parameters $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_n$, learning rate (hyperparameter) η and some performance measure $P(\cdot)$ to be applied to the given model with accompanying cutoff-threshold T .

```

while  $P(M) < T$  do
  for pair  $(I_i, O_i)$  in  $\text{shuffle}(I, O)$  do                                 $\triangleright$  Randomise pair ordering
     $e \leftarrow E(M(I_i), O_i)$ 
    for layer  $L_i$  in  $M$  do
       $G \leftarrow \frac{\delta \mathbf{w}_i}{\delta e}$   $\triangleright$  Find the gradient of the parameters of  $L_i$  wrt. the error
       $\mathbf{w}_i \leftarrow \mathbf{w}_i + \eta \cdot G$   $\triangleright$  Nudge the parameters to minimise the error
    end for
  end for
end while

```

2.4.1 Fully connected neural networks

The structure of each layer in a given neural net can be adapted to best fit to the task at hand. The most basic structure is the fully connected, feed-forward neural network, first described in Rosenblatt (1963), which works well for mappings with relatively small input-dimensions. In this case, each layer-mapping for an input vector x is defined as

$$F_i(x) = f(\mathbf{w}_i x + \mathbf{b}_i), \quad (2.7)$$

where f is a chosen non-linear function (often referred to as the activation-function of the given layer), and \mathbf{w}_i and \mathbf{b}_i are two-dimensional weight matrices. The row-size of \mathbf{b}_i and \mathbf{w}_i determines the output-dimensionality of F_i .

Fully connected neural networks can be applied to various learning tasks. For simpler cases, this can be done through adapting the last layer of the neural network, in conjunction with choosing a relevant error function. For single-variable regression, for example, this is done by choosing a row size of 1 for \mathbf{w}_{n-1} and \mathbf{b}_{n-1} . For classification tasks, specific activation functions are commonly used, for

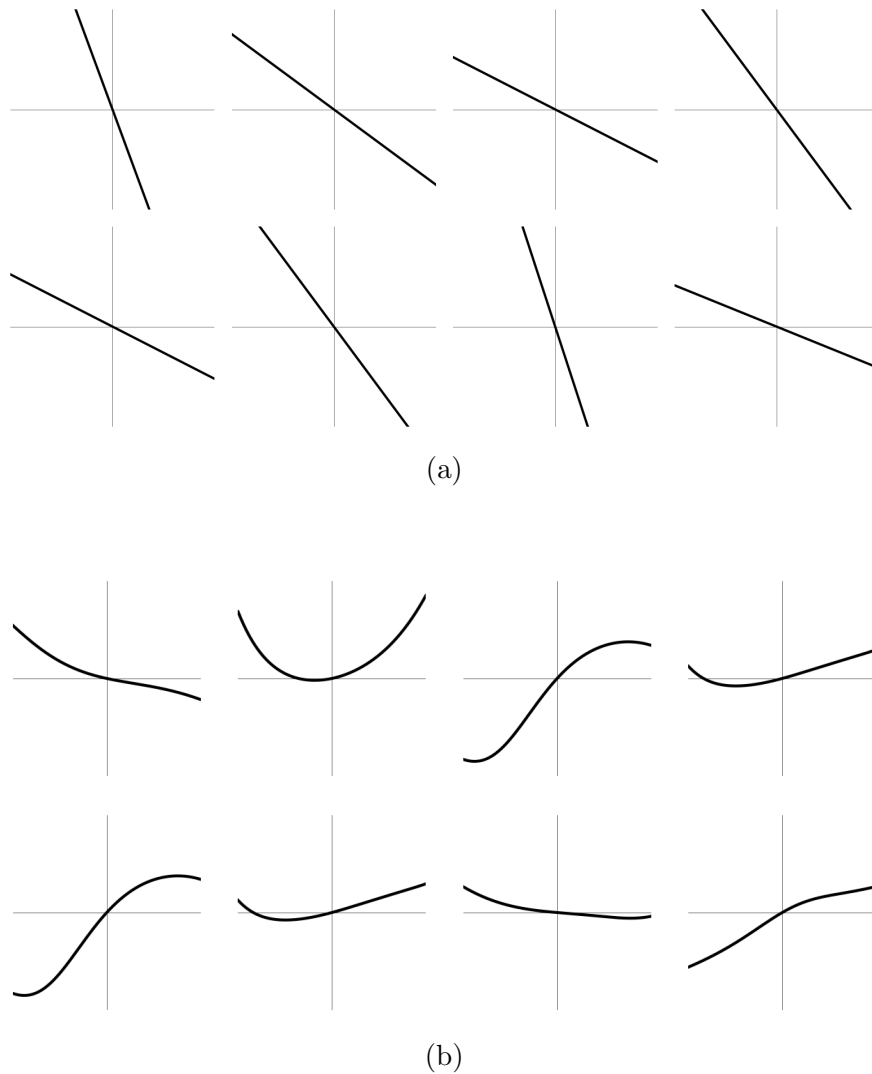


Figure 2.2: Fully connected, 3-layered neural net models with randomly initialised weights and biases, describing a mapping from $[-1, 1]$ to \mathbb{R}^1 . (a) shows such models with a linear activation function, while (b) shows such models with the nonlinear `gelu`-activation function, $f(x) = x \cdot \frac{1}{2}[1 + \operatorname{erf}(x/\sqrt{2})]$ (Hendrycks and Gimpel, 2016b). Observe that the networks with linear activation functions (Fig. 2.2a) only produce linear mappings, even when multi-layered.

instance the Softmax-function

$$S(x)_i = \frac{e^{x_i}}{\sum_{j=0}^n e^{x_j}}, \quad (2.8)$$

where $S(x)_i$ and x_i are respectively the i -th elements of the output vector and x . This function is used to normalise the n -dimensional output vector, so that $S(\cdot)$ can be interpreted as a probability distribution over n outcomes. In practice, this is used to allow the neural network to predict a probability distribution over the n classes.

2.4.2 Convolutional neural networks

There also exist layer structures that are specifically tailored for high-dimensional structured data, such as images, or game boards. One example of such layer structures are convolutional layers, and networks comprising such layers are colloquially called convolutional neural networks (CNNs) (LeCun et al., 1989). Here, each layer structure is defined as a convolution with a variable, fixed-size kernel over its input x . Mathematically, for a single kernel of size (W, W) , the convolutional operation from layer n to $n + 1$ with no activation function can be defined as

$$C_n^{i,j} = \left(\sum_{k=-\lfloor W/2 \rfloor}^{\lfloor W/2 \rfloor} \sum_{l=-\lfloor W/2 \rfloor}^{\lfloor W/2 \rfloor} \mathbf{w}_i^{i,j} \cdot C_{n-1}^{i+k,j+l} \right) + \mathbf{b}_i^{i,j}, \quad (2.9)$$

with the additional constraint that

$$C_{n-1}^{i+k,j+l} = 0 \quad (2.10)$$

for all instances of $C_n^{i+k,j+l}$ where $(i+k)$ or $(j+l)$ is outside the bounds of C_n .

In practical terms, CNNs allow for layer structures where the number of trainable parameters is not directly dependent on the size of the input. For fully connected layers, as described by Eq. 2.7, their setup quickly becomes infeasible when working with larger input sizes. CNNs do however assume some spatial relation between adjacent elements, meaning that it needs to make sense to group adjacent elements together. Therefore, CNNs are often used when dealing with images, or grid-like data, since they benefit from both reducing the computational complexity, and from being able to link bordering elements.

2.4.3 Residual neural networks

While advances in computing allows training of neural networks with great computational capacity and a high number of layers, the training procedure for such

models has often proved difficult. Various layer architectures have been proposed to remedy this, such as the residual network, or “ResNet”, architecture proposed by He et al. (2015). Here, the main proposal is adding skip-connections in parallel to each layer. In mathematical terms, this means introducing a direct dependency between the outputs of two layers i and $i - 1$, which adds an additional transfer of information from the output of layer $i - 1$, to the output of layer i . This can be described as

$$F_i(x) = P(f(\mathbf{w}_i x + \mathbf{b}_i), F_{i-1}(x)), \quad (2.11)$$

where $P(\cdot, \cdot)$ is a function that defines how to merge the outputs of layers i and $i - 1$. A standard choice for this function is to define P as being element-wise addition. For cases in which the dimensionalities of layers i and $i - 1$ do not match, P must be tailored to account for this.

The ResNet-architecture has empirically been proven to be effective for large and deep neural network models. It has been utilised in former state-of-the-art image classifier models (as described in He et al. (2015); Szegedy et al. (2016); Mahajan et al. (2018)), in addition to being used in many instances of the AlphaZero-family of models.

2.4.4 Binarized neural networks

Binarized neural networks, described in Courbariaux and Bengio (2016), are a proposed method for neural network layers that utilise both binary activation functions and binary weight matrices. This encompasses activation functions on the form

$$f(x) = \begin{cases} 1 & \text{if condition} \\ 0 & \text{otherwise.} \end{cases} \quad (2.12)$$

An example of such a function is the standard Heaviside function.

In practice, binarized layers can be thought of as a special case of normal neural network layers. However, extra care needs to be taken in order to make sure they are applicable under the backpropagation procedure described in Algorithm 1. This is because the derivative of any function on the form of Eq. 2.12 is 0 for most values (as seen in Fig. 2.3), thus making it unsuited for use within the backpropagation method. This is remedied by using a gradient estimation strategy, that replaces the direct reliance on the gradient of the layer with the given binary activation function. A standard candidate for gradient estimation is the straight-through estimator, first described in Bengio et al. (2013). The described gradient estimation strategy is here to ignore the activation function when calculating the gradient wrt. a given layer. That is, for a fully connected, standard neural network layer on the form $F_i(x) = f(w_{i-1}x + b)$ with f being some threshold function as

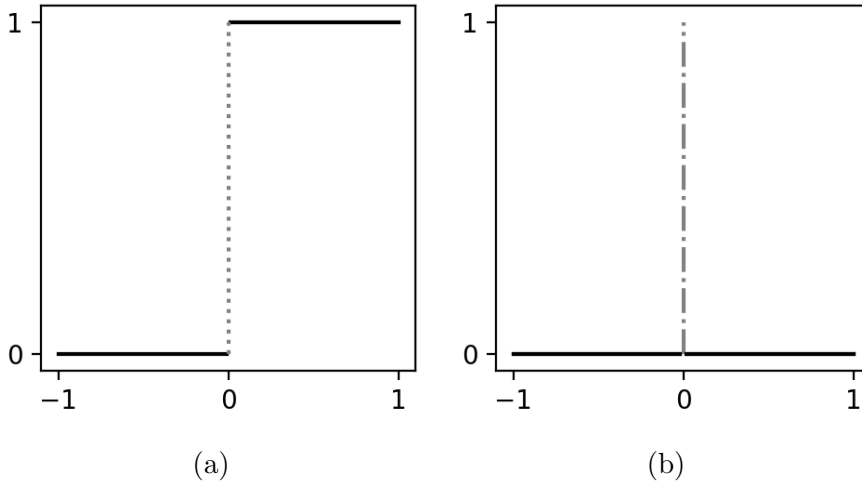


Figure 2.3: The Heaviside step function (a), and its derivative (b).

described in Eq. 2.12, the gradients of w_{i-1} wrt. the given error are calculated as $f(a) = a$. Even though this does not necessarily reflect the exact behaviour of F_i during backpropagation, it has been shown with strong empirical evidence that this is an adequate estimate (Courbariaux and Bengio, 2016).

Binarized neural networks are often used for their benefits when considering computational cost reduction. Such layers are able to substitute most required floating point operations required for calculating $F_i(x)$ with bitwise operations, which have faster implementations on most systems. Using binary weights also means that one is able to reduce the memory footprint of each trained model, drastically decreasing the system requirements for running such models.²

2.5 Applying deep learning to tree search

The chess-playing agent described in Silver et al. (2018) uses a variant of MCTS to achieve superhuman performance in a various set of two-player games. After each set of MCTS-iterations for a given root state s , it uses the tally provided by $N(s, a)$, see Eq. 2.5, to choose its candidate moves. Here, the main idea is that if a node is visited frequently on subsequent traversals, it is likely to represent a favourable state. The practical interpretation of this is that a given move a^* is favourable if it is repeatedly chosen in the search procedure, *and* if it has a high

²This can be viewed as an extension of the many quantisation strategies employed for being able to run large generative diffusion models (Shang et al., 2023) or large language models (Dettmers et al., 2022) on consumer hardware. In these cases, `int8`-quantisation is often used, usually applied after the model has been trained.

$Q(\cdot)$ -value. Here, a large $Q(\cdot)$ -value for a given state, having been sampled $N(s, a)$ times, means that s is valuable with a high certainty.

One of the main contributions presented in Silver et al. (2018) is a neural network model used to estimate many of the functions upon which the MCTS-setup relies. Firstly, it provides the expert probabilities for $P(\cdot, \cdot)$, as described in Eq. 2.5. Secondly, it replaces the outcome sampling by learning to directly estimate $Q(s)$ for any given state s .

The network is trained to produce $P(s, a)$ by approximating $N(s_i, a)$ over all state-action child pairs for a given state s with children s_1, s_2, \dots, s_n . In the same vein, the network is also trained to approximate $Q(s)$ by observing the outcomes of entire games played using the MCTS-procedure. In practice, for a given game-outcome O , one stores samples $(s, N(s, \cdot), O)$ after each complete cycle of the MCTS-algorithm, with the intention of predicting $(N(s, \cdot), O)$ from s . After a fixed number of self-play games, the neural network is then partially trained on the set of gathered samples, essentially creating a partial supervised learning task. The entire MCTS-training procedure is detailed in Algorithm 2.

While MCTS coupled with a neural network estimator has been proven superior to a similar system without a neural network approximator, it is also much more computationally demanding. Therefore, several improvements have been presented for MCTS that make larger computational setups more viable. One of these improvements is being able to benefit from root-parallelized MCTS, first described in Chaslot et al. (2008). This amounts to being able to play several games as described in Algorithm 2 in parallel. Additionally, larger amounts of graphical/tensor processing unit (GPU/TPU) computational resources can be efficiently utilised by implementing a strategy for virtual loss (Chaslot et al., 2008). Virtual loss is a traversal strategy that allows for queries to the $Q(\cdot)$ -function to be deferred. This is important in cases where one uses large neural network approximators, since each evaluation of $Q(\cdot)$ is provided by the given approximator. The virtual loss strategy is realised by replacing and propagating a placeholder value in all cases where the $Q(\cdot)$ -function is utilised. These deferred states are then evaluated in bulk when a sufficient number of states are gathered, replacing the placeholder values upon completion. This yields a higher GPU performance ratio than on-demand evaluation. However, the additional infrastructure required for deferring certain evaluations also incurs a significant overhead, and is therefore only relevant when the amount of GPU/TPU resources is considerable.

Algorithm 2 Full training procedure for a MCTS-based game agent, with n search-iterations per move.

```

 $B \leftarrow \{\}$ 
while training should continue do
   $E_s \leftarrow s_0$  ▷ Start a new self-play game
   $B_t \leftarrow \{\}$  ▷ Episode training buffer
  while  $E_s$  not terminal state do
     $T \leftarrow \text{tree}(E_s)$  ▷ Empty search-tree rooted in  $E_s$ 
    for  $i$  in  $1, 2, \dots, n$  do
       $c \leftarrow \text{root}(T)$ 
      while  $c$  is not leaf do ▷ Traverse to preferred child in  $T$ 
         $c \leftarrow \text{argmax } Q(c_i) + U(c_i)$  over all children  $c_i$  of  $c$ 
      end while
       $O \leftarrow Q(c)$  ▷ Evaluate leaf using neural network approximator
       $T \leftarrow T \cup \text{all children of } c$ 
      for  $c_p$  along path from  $c$  to  $\text{root}(T)$  do
         $Q(c_p) \leftarrow \text{update}(Q(c_p), O)$ 
         $N(c_p, \cdot) \leftarrow N(c_p, \cdot) + 1$ 
      end for
    end for
     $B \leftarrow B \cup (E_i, N(E_i, \cdot))$ 
     $E' \leftarrow \text{argmax } N(E_i, \cdot)$  over all children  $E_i$  of  $\text{root}(T)$ 
     $E^s \leftarrow E'$ 
  end while
   $O_g \leftarrow \text{outcome}(E_s)$ 
   $B_t \leftarrow B_t \sqcup O_g$  ▷ Add the outcome to all instances in episode-buffer
   $B \leftarrow B_t \cup B$ 
  Train NN-approx,  $Q(\cdot), P(\cdot, \cdot)$  on subset of  $B$  ▷ Can be skipped for most iterations
end while

```

2.6 Explainable AI

Explainable AI is a subfield of AI that seeks to “provide transparency to predictions made by machine learning models”, as formulated by (Miller, 2017). Explainability is desirable when considering neural network models: While neural network models are very strong learners, their learned function mappings are opaque, requiring auxiliary techniques in order to explain their inner workings, or the outputs for a given input. In the following, a relevant selection of explanation methods is presented.

2.6.1 Feature importance attributions

The Shapley value, first described in Shapley (1951), is a solution concept from cooperative game theory. For a given cooperative game, with N distinct participants, S denoting any sub-collection, or *subcoalition*, of these players, and $v(S)$ denoting a characteristic function describing the value of a coalition, the Shapley value for the i -th player can be calculated as

$$\varphi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} (v(S \cup \{i\}) - v(S)). \quad (2.13)$$

The practical interpretation of Eq. 2.13 is that one quantizes the contribution of the i -th player to the total game, by investigating the marginal contribution the player makes to all possible combinations that do not include the i -th player.

Even though the concept of the Shapley Value operates on marginal contributions for cooperative games, it can also be used to describe the contribution of features for a given predictive model. In this case, one interprets the model’s mapping $f(x) : I \rightarrow O$ from its input space I to its output space O as a cooperative game played by each feature in x . The outcome of the game, $f(x)$ is then treated as the “outcome” of the given game, and the i -th Shapley Value $\varphi_i(v)$ is then the marginal contribution of the i -th feature in x wrt. the predictive output $f(x)$.

While Eq. 2.13 allows for an adequate description of the contribution of each feature in a coalition, it has a number of practical challenges that limit its applicability in a machine learning context. Firstly, it is infeasible to enumerate over all coalitions of N for large N . This can be interpreted as requiring to include all possible combinations of N in the measure of the “contributational value” of a single feature. To mitigate this, one can impose various sampling strategies as to estimate the information gained by looking at all such combinations. One of the most straightforward of such strategies is using a Monte Carlo sampling strategy, such as presented in Mann and Shapley (1960), and later used in Štrumbelj and

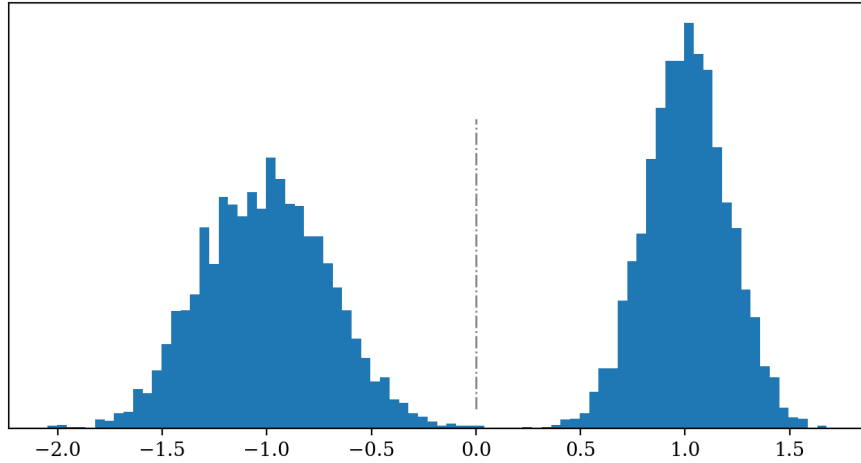


Figure 2.4: A histogram of a bimodally distributed variable. Observe that the mean falls between the two sharp peaks of the distribution.

Kononenko (2014), where one samples a set number of the coalitions inferred from Eq. 2.13, instead of using an exhaustive enumeration.

Additionally, it is not immediately obvious how to define a coalition of an input space for models with fixed input sizes. As an example, for a neural network of input-size n , it is not possible to exclude some of the input features in order to evaluate $v(\cdot)$, in this case the neural network model, on the given subcoalition. Thus, one needs to define a substitute operation that adequately represents the model’s output in the absence of a given feature. A naive approach might be to replace the value of absent features with their sampled mean, with the assumption that this value serves as a default for the given feature. While this might work for some features, this can for example be problematic for features that are bimodally distributed, as illustrated in Fig. 2.4. Neural network models have proven vulnerable to out-of-distribution feature values, as discussed in Goodfellow et al. (2015); Hendrycks and Gimpel (2016a), meaning that such an approach can be sub-optimal. A more apt approach is sampling values from the feature’s observed distribution, as this could provide more likely instances of the given feature. However, this sampling strategy also needs to consider the correlation between each sampled feature feature. This is because one might sample an instance of a feature that only occurs within a range of a certain subset of other features, which creates the same problems wrt. out-of-distribution samples. There exist many other extensions and related methods for predicting Shapley-like values, often adapted to be more applicable for specific model classes. Among these methods are KernelShap (Lundberg and Lee, 2017), DeepLift (Shrikumar et al., 2017), and Quantitative

Input Influence metrics (Datta et al., 2016). Several of these methods are coupled and packaged into the widely used SHAP-package (Lundberg and Lee, 2017).

2.6.2 Saliency maps

Saliency maps are a category of explanations that are well suited for models operating on highly structured data. This category encompasses a wide range of methods, but such methods commonly provide a visual metric of relevance as to how each section of a given sample is relevant for producing the given prediction. For visual data, such as images, this depiction is often presented as an overlying mask on the original image, which in turn constitutes an visual explanation as to what the model is “looking at” when making its inferences. For a three-channel (RGB) (n, n) -sized image I , consisting of $(n, n, 3)$ values between 0 and 255, and a heatmap H with (n, n) values between 0 and 1, each element of the visualisation mask can be defined as

$$S_{i,j} = 255 - H_{i,j} \cdot (255 - I_{i,j}). \quad (2.14)$$

Practically, this equation describes calculating the resulting colour of $I_{i,j}$ by applying the corresponding opacity-value defined by $H_{i,j}$.

Feature importance attributions, presented in Sec. 2.6.1, can also be thought of as providing saliency maps for sufficiently structured data, and for images can be thought of as operating on the basis of “superpixels”. That is, given an image, it regards collections of pixels as a single feature, and uses this to discern the importance of each such region for the given prediction.

For neural networks, there also exist many methods that can be used to provide saliency-based explanations. Methods specific to neural network models often leverage the gradients of each layer of the network wrt. the predicted output O , for a given input-output pair (I, O) . Such methods often leverage this information in order to avoid having to mask features when calculating the corresponding output for a given permutation, which is often beneficial when considering that neural network models are not reliably robust to inputs not present during training (Hendrycks and Gimpel, 2016a).

There exist many gradient-based saliency map methods for neural networks. Among these is GradCAM (Selvaraju et al., 2016), which provides a class activation map as a heatmap over the given input state. It operates on CNN-classifier models, and is originally intended for image-like data. Its procedure is described in Algorithm 3.

Algorithm 3 GradCAM for a CNN-based classifier M taking s as input, with last convolutional layer L with filter dimensions (m, n) , and l filters

$c \leftarrow \operatorname{argmax}(M(s))$ ▷ Get the predicted class
 $G \leftarrow \frac{\delta y_c}{\delta L}$ ▷ The gradients of the output corresponding to c, y_c
 $M \leftarrow \frac{1}{m \cdot n} \cdot \sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^l G_{i,j}^k$ ▷ Mean of gradients over each filter in L
 $H \leftarrow \sum_{l=1}^k G^l \cdot M^l$ ▷ Weighted sum of the gradients of each filter
 $H \leftarrow \operatorname{upscale}(H)$ ▷ Transform the dimensions of H to match the dimensions of the input space
 $H \leftarrow \frac{\operatorname{clip}(H, 0)}{\max(H)}$ ▷ Normalize between 0 and 1

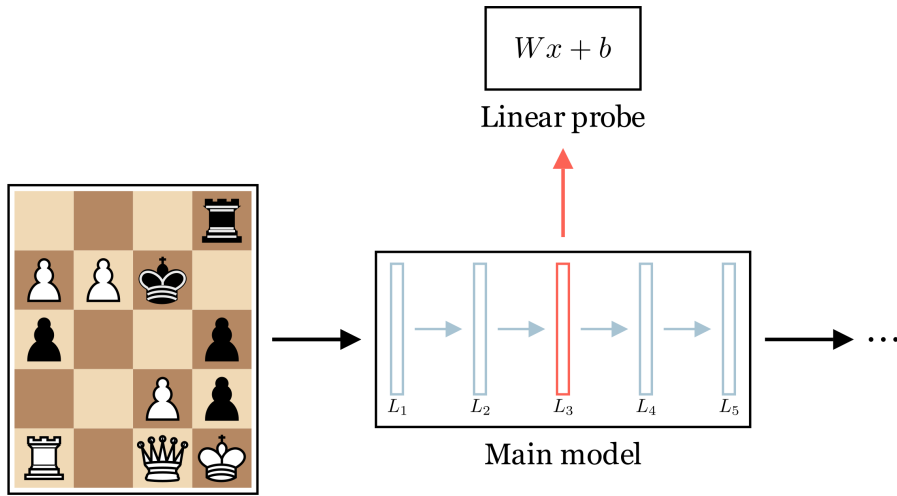


Figure 2.5: The main architecture of concept detection, operating on an arbitrary model. Here, the linear probe is inserted at layer L_3 .

2.6.3 Concept detection

It can be desirable to know which representations of established domain knowledge a given neural network model M internalises. This is particularly relevant in the context of chess, since it would make it possible to probe the model using the vast amount of accrued domain knowledge, described in Sec. 2.1. This goal is achieved by utilising a method for detecting so-called *concepts*, first described in Kim et al. (2018), and later utilised by McGrath et al. (2021) for searching for internal representations of domain knowledge in AlphaZero. For a concept function $C(s)$, where s is a given state, and the output of $C(\cdot)$ is a Boolean value stating the presence of a given concept in s , the method trains logistic probes to predict $C(\cdot)$ given the activation outputs from an intermediary layer of the model M . This

structure is illustrated in Fig. 2.5.

For a given model M , and a partial model function $M_n(S) : I \rightarrow O_n$ where I is the input space of the model, and O_n is the activation space of the n -th intermediary layer, a concept dataset is constructed. This dataset consists of a set of pairs $(M_n(s), C(s))$, where s is sampled from the state space of the model. These can often be gathered through self-play, or through known datasets, as described in McGrath et al. (2021). Then, one wishes to train a logistic probe to approximate the mapping $P(M_n(s_i)) : O_n \rightarrow C(s_i)$ for each s_i in the dataset. That is, one wishes to find some weight matrix \mathbf{w} and bias \mathbf{b} so that

$$\|\sigma(\mathbf{w} \cdot M_n(s_i) + \mathbf{b}) - C(s_i)\|_2^2 \quad (2.15)$$

is minimised for all (s_i) , where σ is the sigmoid function. Additionally, to confirm that the concept is, in fact, sparsely represented, an $L1$ -penalty is applied to \mathbf{w} and \mathbf{b} . This yields the final minimisation objective

$$\|\sigma(\mathbf{w} \cdot M_n(S_i) + \mathbf{b}) - C(S_i)\|_2^2 + \lambda \|\mathbf{w}\|_1 + \lambda \|\mathbf{b}\|, \quad (2.16)$$

where λ controls the strength of the applied $L1$ -penalty.

The presence of a concept given the logistic probe $L(\cdot)$ with weight matrix \mathbf{w} and bias \mathbf{b} is then defined to be the binary accuracy of $L(\cdot)$ when applied to a validation portion of the aforementioned data set, while correcting for random guessing. This is then defined to be

$$\frac{2}{N} \left(\sum_i^N H(L(O_i) - T) - P_i \right) - 1, \quad (2.17)$$

where H is the Heaviside step function, and T is the binary predictive threshold for the probe. Usually, T is set to 0.5.

The information described in this section has been adapted for use, and presented in Hammersborg and Strümke (2022).

2.6.4 Counterfactuals

It is also interesting to consider the applicability producing counterfactual explanations for a given predictive model M . Given an input sample I_n , a model that produces an output $M(\cdot)$, a corresponding prediction $O_n = M(I_n)$, and a desired prediction O_n^* , a valid counterfactual explanation would be a perturbation I_p , so that $M(I_n + I_p) = O_n^*$. It is desirable that the perturbation I_p is small, so that $I_n + I_p$ is semantically similar to I_n . In practical terms, one wishes to find a small

change to the given input sample so that the prediction of the model changes from O_n to O_n^* .

A standard method for generating counterfactuals, presented in Wachter et al. (2017), expresses these requirements by aiming to minimise the given loss function

$$L(I_n, I_p, O_n^*, \lambda) = \lambda \cdot (M((I_n + I_p) - O_n^*))^2 + d(I_n, I_n + I_p) \quad (2.18)$$

Here, the term $\lambda \cdot (M((I_n + I_p) - O_n^*))^2$ expresses the λ -weighted mean squared error between the prediction found by the counterfactual sample $I_n + I_p$ and the desired prediction O_n^* . $d(\cdot, \cdot)$ is a distance function expressing the distance between two input samples, here being used to measure the distance between I_n and $I_n + I_p$. The distance function can be adapted to the given problem at hand. The mean squared error aims to minimise the error of the desired prediction, while the distance metric aims to minimise the perturbation.

Another method for generating counterfactuals is presented in Dandl et al. (2020), where a minimisation-objective for guiding a search procedure for counterfactual explanations is presented. Semantically, the method is similar to the one expressed through Eq. 2.18. However, the method presented in Dandl et al. (2020) includes an additional term that aims to capture the plausibility of the found perturbation. That is, given a perturbation $I_n + I_p$, the objective also represents how the likelihood of the sample $I_n + I_p$ appearing as a sample from the given input space. This can be a very useful addition. For chess, for example, such a restriction might indirectly impose that the found position is legal within the rules of the game, which is imperative in most use cases.

Overall, most methods for identifying counterfactual instances aim to provide some mathematical heuristic guiding a search for a viable perturbation to the given input sample. Although the minimisation objectives used might vary, most intend to minimise the distance from the initial sample and the perturbed sample, in addition to making sure that the corresponding model output for the given perturbation closely matches the desired output.

Counterfactual explanations have been shown to be easily interpretable, while not requiring technical knowledge of the given model (Miller, 2017). Additionally, it is very practical to create counterfactual explanations from tree-like structures, which might lend itself well to a MCTS-like model setup, described in Sec. 2.3.

Chapter 3

Methods

3.1 High performance chess environment

For a training process as described in Sec. 2.5, it is imperative to have an environment with adequate performance. For chess, this mainly hinges on being able to rapidly generate the legal moves for a given position, since this sub-routine is used repeatedly while enumerating the children of any given leaf node in the MCTS-process. This task is difficult to perform quickly, due to the complexity of calculating these moves. The difficulty presented by this task is discussed in Sec. 3.1.1, and complemented in Appendix A.

3.1.1 Bitboards

Although most of the game state in chess can be adequately represented by standard array-like structures, one can achieve a significant gain of performance by switching to bitboard representations. This approach was described in Atkin and Slate (1988), and has since become a standard game-state representation utilised by most competitive chess playing programs. The main idea is that each combination of unique piece and colour type is given its own binary input plane, where each element represents the presence or absence of each combination of piece-type and color on the given square. This is illustrated in Fig. 3.1. For all chess-variants with board sizes up to 8x8, this then means that each input plane fits within a single 64-bit integer, which in turn means every possible piece permutation in chess can be represented by twelve 64-bit integers.¹ Each such plane is called a bitboard.

¹For the game in its entirety, some additional information is also required. This includes the current player to move, the castling rights of each player, the total move count thus far in the game, and the current en passant status.

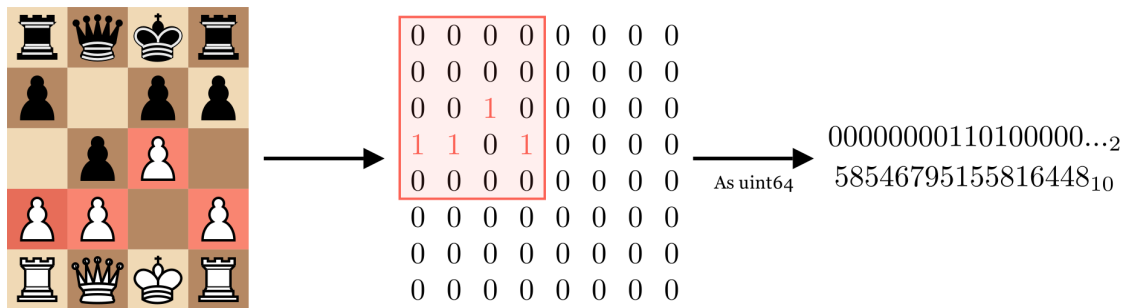


Figure 3.1: An illustration of representing parts of a chess state using bitboards. This also shows how board sizes smaller than 8x8 (equaling 64 bits in total) fit into an unsigned 64-bit integer.

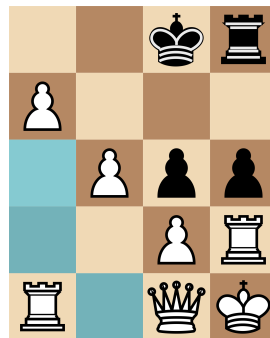


Figure 3.2: The possible moves for a rook in the lower-right corner, which can only move in a straight line.

Altering any given piece state is done by defining relevant bitwise-operations between the given bitboards and Boolean masks. Examples of this include removing or adding a certain piece from the bitboard, but also more complicated endeavours, such as calculating pinned pieces (see Appendix A), or calculating the legal attacks for a given piece.

3.1.2 Precomputing possible attacks

An integral part of the procedure for calculating the legal moves of a state, is being able to find the possible movement vectors for any given piece on the board. If this is performed in a naive way, the computation can be quite slow. While most pieces have a somewhat simple movement pattern, their movement might be restricted by other pieces on the board. An example of this is shown in Fig. 3.2.

In order to avoid having to consider blocked squares for each piece when calculating the legal moves of a position, a strategy for precalculating all such possible

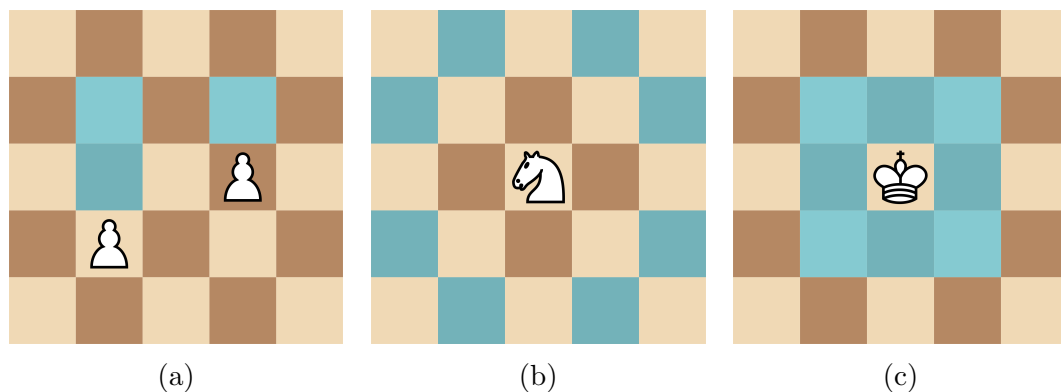


Figure 3.3: The possible moves, highlighted in blue, for (a) the pawn, (b) the knight, and (c) the king.

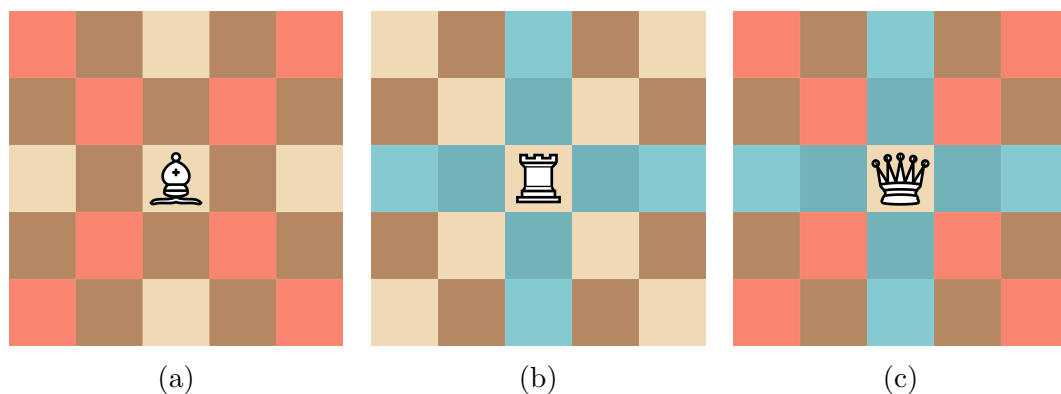


Figure 3.4: The possible moves for (a) the bishop, (b) the rook, and (c) the queen. Diagonal moves are highlighted in red, and straight-line moves in blue. The main observation is that the queen has the movement of a combined rook and bishop.

movement vectors is devised. In this way, one can use a sub-optimal routine for finding all possible moves before runtime, and substitute that operation with a simple lookup when the chess-environment is being utilised.

First, it is observed that the movement patterns for the pawns, king, and knight are easy to define. These are shown in Fig. 3.3. For these pieces, it suffices to provide a static lookup for each square, designating the possible moves for any piece of the given type on that square. The movement patterns for the queen, rook, and bishop can be decomposed into diagonal and straight-line moves, as shown in Fig. 3.4. These two types of movement can then be considered separately. In these cases, the movement map also needs to consider all other pieces that potentially hinder movement.

Let B be a bitboard that indicates which squares of the board that are occupied. Additionally, let $F_m(S, B)$ be some function that produces a bitboard of legal moves of movement type m from a square S , given the blockers B . The results of F_m are then cached as a array-like lookup table depending on S and the integer interpretation of B . However, since B is stored as a 64-bit integer, it is impractical to have a lookup table depend on a 64-bit value. In practice, since it is desired to cache F_m using a regular array-like lookup, it is also necessary to create some reduction of B for use with F_m . The first step is to observe that all pieces that do not fall within the movement path of the movement type m are irrelevant for blocking movement of type m . In practice, this means that one only needs to consider pieces that can hinder its movement. This is done by reducing B by instead looking at $B_m = B * \text{path}(S, m)$, where $\text{path}(S, m)$ is a function providing a bitboard containing all squares which are on the movement path with movement of type m starting from the square S , and the $*$ -operator is regular binary multiplication. This is then used in conjunction with a hash-mapping of B_m from 64-bit, to n -bits:

$$F_m(S, H(B_m, n)) \quad (3.1)$$

where n is the bit-length of the output of $H(\cdot, n)$. The choice of H is set to be something that is very quick to compute. In this case, H is defined as:

$$H(B_m, S, n) = (B_m * M_S) \gg (64 - n) \quad (3.2)$$

Here the $*$ -operator is regular binary multiplication, and the \gg -operator is the binary-right-shift operator. The main task here is finding an M for each S so that H is unique for all possible values of B_m . This is usually done by brute force. n is chosen heuristically, as it dictates how much storage space is needed for the produced lookup table, but also how long the brute-force search will take. The values for M_S are then stored along with the main lookup table for F_m .

3.2 Training models

In order to develop game-playing models that reach superhuman performance without the use of pre-instilled knowledge, the DRL-based neural network models are trained as described in Silver et al. (2018). The main training procedure is based on deep reinforcement learning through MCTS and self play, as described in Sec. 2.5, and directly follows Algorithm 2. The procedure is discussed further in throughout following section.

In this thesis, two models are trained on smaller variants of chess. One model is trained on Silverman 4x5 chess, shown in Fig. 3.5a, and the other is trained on Los Alamos 6x6 chess, shown in Fig. 3.5b.

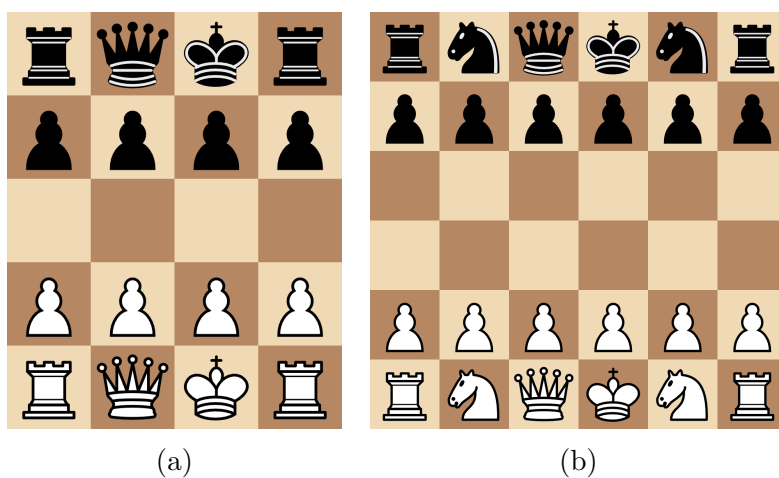


Figure 3.5: The two chess-variants used: (a) Silverman 4x5 and (b) Los Alamos 6x6.

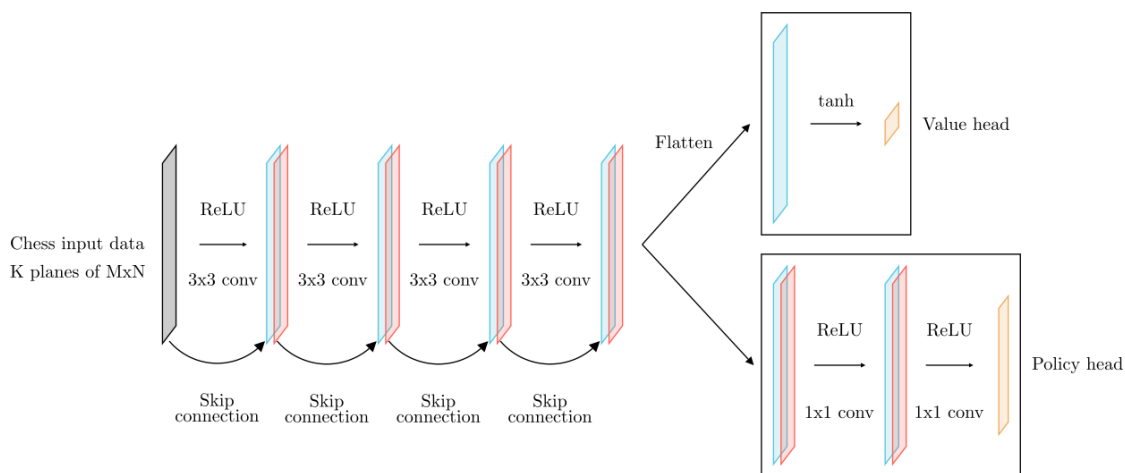


Figure 3.6: The architecture of the 6x6-ResNet model. The structure of the 4x5 model is similar, but without skip-connections and only three initial convolution layers.

Table 3.1: Description of each input channel of a chess-representation adapted for neural networks.

Channel number	Description
0 – 5	One plane for each piece-type for the player to move. (in the order of pawn, knight, bishop, rook, queen, king)
6 – 11	One plane for each piece-type for the opposing player. (in the order of pawn, knight, bishop, rook, queen, king)
12 – 15	Kingside, queenside castling rights for both players (not used in the presented variants)
16	En passant move availability (designating the square that is able to be captured)
17	Counter of the amount of turns taken (not including the current position)
18	If the player to move has the white pieces.

3.2.1 Model architecture

Both models developed and used in this thesis are feed forward convolutional neural networks (CNNs), discussed in Sec. 2.4.2. The 6x6 model has added residual connections, similar to the architecture of AlphaZero Silver et al. (2018). This is based on a standard ResNet-architecture, described in Sec. 2.4.3. For a given state, both models are tasked with predicting a policy vector over all possible moves, in addition to an evaluation of the state. An overview of the model architecture is shown in Fig. 3.6.

3.2.2 Input/output structure

A chess position with board size (m, n) is represented as a single tensor with dimensions $(m, n, 19)$. The content of each channel is described in Table 3.1. Additionally, the input is always kept player-invariant. This means that if the player to move is playing as White, the entire board is rotated 180 degrees before being passed to the network.

The size of the unflattened policy output varies with board size. For a board of size (m, n) , the policy output is of size

$$(m, n, 4 \cdot (m - 1) + 4 \cdot (n - 1) + 8 + 3 \cdot 3) \quad (3.3)$$

For each square (i, j) , this encodes $m - 1$ moves vertically upward and $m - 1$ moves vertically downward, $n - 1$ moves horizontally leftward and $n - 1$ moves

horizontally rightward, 8 knight moves, and $3 \cdot 3$ moves for each underpromotion.² This means that each policy vector might include moves that are not possible to make in the given position, but such moves are masked away before the policy output is utilised elsewhere.

3.2.3 Training architecture

Both models are trained according to the setup described in Sec. 2.5. In lieu of large GPU-resources, as required by many state-of-the-art MCTS-strategies, a fast and lightweight CPU-only procedure for generating model predictions was used as a part of the custom MCTS training loop used for this work. The training loop utilises TensorFlow-lite models (Abadi et al., 2015), which leverage fast, single-thread model inference on the CPU. This is highly relevant in this use-case, as having fast single-thread inference means that the whole training loop can be parallelized with negligible overhead. For larger models, however, such as those described in Silver et al. (2018), a robust implementation allowing more efficient use of GPU resources would most likely be necessary for productive training. The implementation of the described pipeline was optimised for efficiency on medium-level consumer-grade hardware. This means that it was intended for systems with relatively few, but fast CPU cores, and a single, lower-range GPU.

3.2.4 Availability

The described environment in Sec. 3.1, the framework for the entire training loop described in Sec. 3.2, and models trained on 4x5 and 6x6-variants of chess are publicly available³ as a viable alternative for training smaller chess models in the same manner as described in Silver et al. (2018). In order to make it as broadly applicable as possible, the described chess environment is completely detachable from all other training infrastructure, in addition to being entirely customisable. This includes custom starting-positions, board-size, and castling rules.⁴

²When a pawn reaches the final rank (meaning that it has traversed the entire board), it is promoted. The standard rules of chess state that his promotion can be to either a bishop, rook, knight, or (most commonly) a queen.

³This is available at <https://github.com/patrik-ha/explainable-minichess>.

⁴Castling is a special move-type in chess, which involves moving the king and one of the player's rooks at the same time. This can only be done once per player, and has special requirements as to what squares have to be vacant and non-attacked.

Table 3.2: Concept functions used in Sec. 2.6.3.

Name	Description
<code>has_mate_threat</code>	Checkmate is available
<code>in_check</code>	Is in check
<code>material_advantage</code>	Has more pieces than opponent
<code>threat_opp_queen</code>	Opponent’s queen can be captured
<code>has_own_double_pawn</code>	Has two pawns on the same file
<code>has_opp_double_pawn</code>	Opponent has two pawns on the same file
<code>has_contested_open_file</code>	Both players have rooks in an open file
<code>threat_my_queen</code>	Own queen can be captured
<code>random</code>	Data set with random labels

3.3 Concept detection

The concept detection procedure described in Sec. 2.6.3 is performed on both models described in Sec. 3.2.1. The concept functions used are described in Table 3.2. The concept probing data sets for a given board size are created through large amounts of self-play using trained model checkpoints. A small batch of positions is then randomly sampled from these games, and assigned labels corresponding to the concepts present in each position. This process is repeated until a complete concept dataset has been formed. Additionally, noise is added to the move selection process, in order to mitigate any potential bias of the produced concept dataset in favour of the models being tested. This also makes sure that the concept datasets contain a large variety of games.

3.4 Backpropagating concepts

While the concept detection method presented in Sec. 2.6.3 can be used to identify what kinds of domain knowledge a given chess-playing model has internalised, the method does not give any indication as to how this knowledge is used and represented internally. Essentially, one cannot know if the internalised version of the given concept matches how the concept is canonically represented in relevant domain knowledge. However, since a successfully trained probe indicates what elements in activation space that correspond with the model’s internal representation of the concept, the probe can provide information as to what kinds of activation patterns correlate with the given concept. One could then consider searching through the input space of the main chess-playing model, trying to find perturbations of a given state that successfully match these found activation pat-

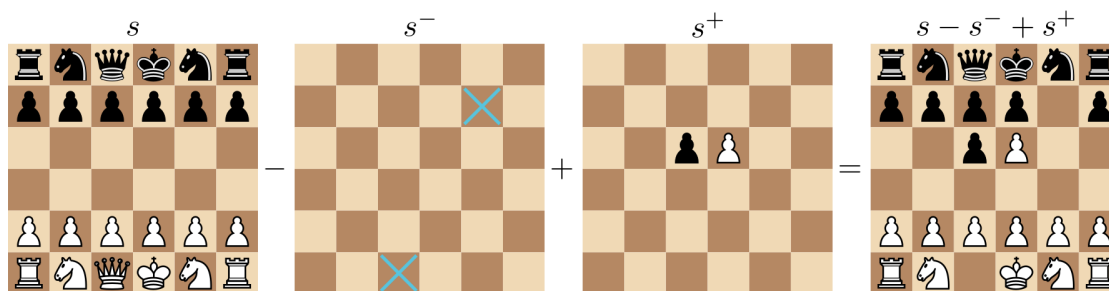


Figure 3.7: An illustration of the binary masks described in Sec. 3.4. In the shown figure, s^- is represented as a $6 \times 6 \times 1$ matrix, meaning that it masks over all pieces on the board, while both s and s^+ are represented as $6 \times 6 \times 12$ -matrices. This means that each combination of piece color and type is given its own plane in s^- .

terns. This can be thought of as trying to maximise the output of the linear probe, by only being allowed to perturb the input space of the main chess-playing model. Since the probe itself is a linear neural network, it can be used to provide gradients for this search. This creates a minimisation-problem – what is the smallest perturbation for a given state s that successfully causes the concept C to be detected by the probe P ?

The aim of the method presented in Sec. 2.6.3 is to find a sparse weight matrix \mathbf{w} that indicates which of the elements in a given output activation layer O_i linearly correlate with a predefined concept. Subsequently, it is possible to maximise the perceived detection of the given concept by altering the elements with corresponding non-zero weights in \mathbf{w} . In practice, given a composite of a pre-trained model and a trained probe as a map from an input space I , through an activation space $M(s) : I \rightarrow O_i \rightarrow C$, where s is a given state, and C is the corresponding probe-output, one wishes to find a perturbation $M(s - s^- + s^+)$ that maximises C . It is desirable that $|s^-| + |s^+|$ is as small as possible, and it is required that both s^- and s^+ are binary.

The requirement of s^- and s^+ being binary is satisfied by representing s^- and s^+ as binary masks. These are then attached as preprocessing-layers to the main model M , while “simulating” that each element in s^- and s^+ are trainable variables. These masks are made trainable by implementing them as binarized layers, as described in Sec. 2.4.4. A visual representation $s - s^- + s^+$ is shown in Fig. 3.7.

At a higher level, it is also desirable that $s - s^- + s^+$ is semantically similar to other legal positions from I . This is implemented by attaching a pre-trained a submodule that predicts the “legality” of $s - s^- + s^+$, which provides an additional objective to maximise along with C . This submodule is trained on a dataset of 300 000 positions sampled through self-play of the given model M , where 50% of

the positions are altered as to become illegal.⁵ In practice, this means that this submodule is tasked with performing a binary classification task, where the class in question is the “legality” of a given position.

Algorithm 4 Concept backpropagation for a single input sample s , partial model $M_I(\cdot)$ as a map from the input space of s to the activation space of an intermediate layer I in M , pre-trained concept probe P on the activation space I , pre-trained legality-checker $C(\cdot)$ that evaluates the legality of any given position, and a threshold T as to define adequate maximisation of the given concept.

```

 $s^+ \leftarrow \mathbf{0}$ 
 $s^- \leftarrow \mathbf{0}$  ▷ Binary masks over input, i.e. same dimensions as input
 $I^* \leftarrow M_I(s + s^+ - s^-)$ 
while  $P(I^*) < T$  do ▷ Until concept is sufficiently maximized
   $l_p = \text{bce}(P(I^*), 1)$  ▷ bce is the standard binary crossentropy loss function
   $l_c = \text{bce}(C(I^*), 1)$ 
   $l_{L1} = c_1 \cdot \sum_{i,j} |s_{i,j}^+| + c_2 \cdot \sum_{i,j} |s_{i,j}^-|$  ▷  $c_1, c_2$  tunable hyperparameters
   $s^- \leftarrow s^- + \frac{\delta(l_p + l_c + l_{L1})}{\delta s^-}$ 
   $s^+ \leftarrow s^+ + \frac{\delta(l_p + l_c + l_{L1})}{\delta s^+}$  ▷ Take gradient step for masks to minimise loss
   $I^* \leftarrow M_I(s + s^+ - s^-)$ 
end while

```

A full description of the procedure is provided in Algorithm 4.

3.5 Binarized intermediate layers

When considering explanatory methods for chess models, it is also interesting to consider the consequences of having intermediary layers with activation spaces that resemble the input space itself. As described in Sec. 3.2, the dimensionality of the input is in this case approximately equal to the dimensionality of most of the intermediate layers in the models. However, this is not true for the data type output by these layers. The part of the input space that describes the board state is strictly binary, while normal activation spaces usually span real-valued numbers, with value ranges determined by the activation functions used. However, it could be interesting to force some of these activation spaces to also be binary, to examine whether it is possible to relate activation patterns to features that appear in the

⁵For most standard chess variants, there exist some categories of positions that cannot be reached by normal play. This is what is meant when referring to “illegal positions”. Such positions can be subtle, as they are often unreachable due to restrictions posed on certain pieces, or certain moves at some stages in a given game.

input positions. Examples of such features could for example be attack masks, such as those shown in Fig. 3.2.

Algorithm 5 Search strategy for finding specified binary masks C_i in the binary activation channels B_i with n activation channels in total, for a given input sample I_i , and for a given model M that produces B_i given I_i . This finds the indices of all channels that always match the given binary mask.

```

 $L \leftarrow \{0, 1, \dots, n\}$ 
for each pair  $(I_i, C_i)$  do
   $B_i \leftarrow M(I_i)$ 
  for  $j$  in  $\{0, 1, \dots, n\}$  do
    if  $B_{i,j} \neq C_i$  then
       $L \leftarrow L \setminus \{j\}$   $\triangleright$   $B_i$  is no longer a viable candidate for representing  $C_i$ 
    end if
  end for
  if  $L = \emptyset$  then
    exit
  end if
end for

```

The middle layer of the 4x5 model described in Sec. 3.2 is replaced by a binary layer. The layer is implemented using the method described in Sec. 2.4.4. In order to account for the reduced information throughput of binary layers, the number of convolutional filters is increased by a multiple of 16. The “observation strategy” is performed by pre-defining a set of binary masks for each position that indicates the activation pattern that could potentially be found. This strategy then checks each channel of the convolutional activation space for a match to the given binary mask. The algorithm is described in Algorithm 5. Here, the main intention is to investigate if there are any channels that inhibit some sort of “specialisation”, for example if a single channel always represents the available moves for one player’s king. The results of this approach are presented and discussed in in Sec. 4.3.

3.6 Output correlations

It is also possible to observe the direct correlation between detected concepts and their detected presence for any position s . Given a vector $C(s)$ where the i -th element in $C(s)$ is a binary value indicating the presence or absence of the i -th concept in the position s , one can consider the correlation of each of the elements in $C(s)$ with the predicted value of the state $Q(s)$, given by the trained model, described in Sec. 3.2. Although the various concepts have no guarantee of affecting

$Q(s)$, overarching themes can be identified by looking at these correlations in the aggregate.

This was done by gathering 300 000 self-play positions by the trained model, and then having the same model produce $C(s)$ and $Q(s)$ for a select set of concepts. Different correlation measures were then applied to a random subset of 10% of these samples. This process was repeated 30 times for each correlation measure. The correlation measures used in these methods are the R^2 coefficient of determination, the Hilbert-Schmidt independence criterion (`hsic`) (Gretton et al., 2005), and the measure of distance correlation (`dcor`) (Székely et al., 2007).

3.7 Counterfactual explanations

Much of the playing strength of the chess models described in Sec. 3.2 comes from being utilised as a guide in an effective tree search. With this as a starting point, it is also possible to utilise the tree-structure to generate counterfactual explanations as introduced in Sec. 3.7. Given a state s and a produced policy vector $p(s)$ representing a probability distribution over all moves from the state s , one can define a strategy for producing counterfactual explanations by looking at the highest valued actions in $p(s)$. If a_1, a_2 are the highest and second-highest valued elements in $p(s)$ respectively, s_1, s_2 are the states reached from applying these actions to s , and a^* is the action with the highest valued element in $p(s_2)$, one can aim at using this to explain why a_1 is preferred over a_2 . This is done in the form of a “if-this-then-that”-explanation. The explanation follows the following template: “The player p performs a_1 in a given state s to arrive at s_1 , since if p performs a_2 , the opponent has the rebuttal a^* in the state s_2 ”.

While somewhat dissimilar to a standard counterfactual explanation, this formulation can be described as a relaxed version of the method described in Sec. 3.7. Here, the desired prediction is set to be $\operatorname{argmax} p(s_1) \neq \operatorname{argmax} p(s_2)$, and the distance function for minimisation is set to be in terms of edges between nodes in the game tree. An additional similarity constraint is added, which encompasses that s_1 and s_2 are respectively the highest and second-highest valued child-states of s . The proposed search is thus replaced by a more guided tree traversal. For the given state s , and its highest-scoring child-state s_1 , one can assign a counterfactual state s_2 as the second-highest child-state s_2 , such that $\operatorname{argmax} p(s_1) \neq \operatorname{argmax} p(s_2)$.

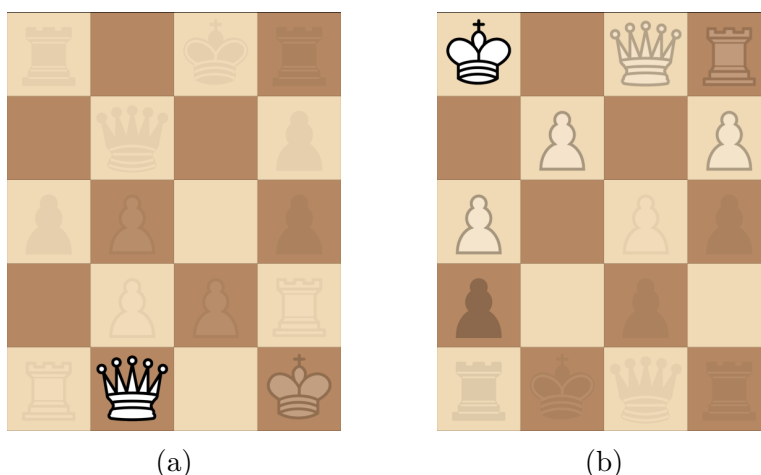
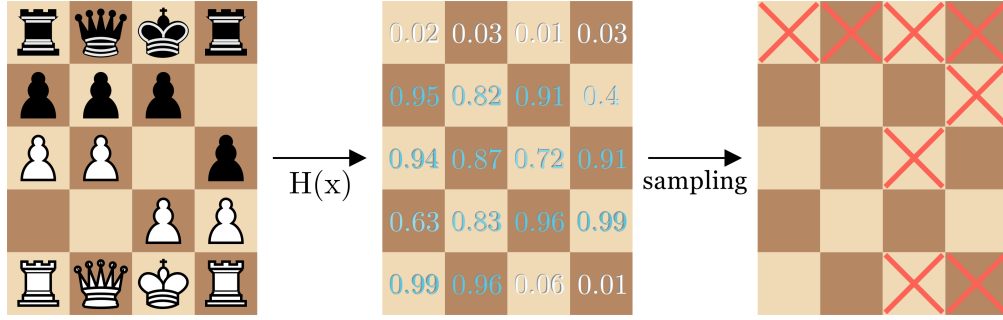


Figure 3.8: GradCAM applied to two positions of 4x5 chess. Observe that the explanation method only highlights a single piece, which is the piece to be moved for the preferred move.

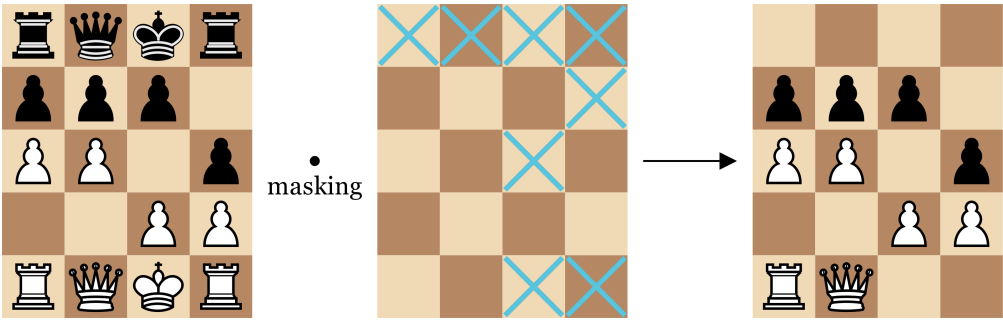
3.8 Heatmap-based explanations

Heatmap-based explanations, such as GradCAM for image-like data, have proven to be very popular. From an explanatory point of view, they provide a high-level, easy-to-understand explanation that visualises which part(s) of the input is relevant for the model when making a given prediction. Such qualities would also be beneficial when producing explanations for chess-playing models. GradCAM can be trivially adapted to chess models, following the procedure described in Algorithm 3. However, such an implementation has empirically been shown to give poor results for chess models. This can be seen in Figs. 3.8. One could think that this is mainly because the method does not account for the presence of a piece being represented in a binary fashion. Practically, if this is thought of as a means of “allocating gradient” to each square for a given move, one can think that the method would sink most of its available gradient into the square that makes the move possible in the first place, without any regard to the binary nature of the input space. Because of this, it is necessary to consider different heatmap generating methods in the case of chess playing models.

The proposed method adds an additional module to each of the models described in Sec. 3.2. The intended effect is to make the model learn an importance map for any given input board. That is, given an input state s , a model M with corresponding loss function L , and predicted outputs $O = M(s)$ with loss $L(O)$, the goal is to add a trainable masking step $R(s)$ while training the model M , so



(a) An illustration of the module stochastically producing a mask $H(x)$ for a given position, and the process of binarizing it.



(b) An illustration of how the binary mask produced by the masking module is applied to a given position.

Figure 3.9: A set of illustrations showing the masking module produces a reductive mask during training (a), and how that reductive mask is applied to the given training sample (b).

that the training procedure operates on $M(R(s))$ instead of directly on $M(s)$. In practice, this means that $R(s)$ should reduce the amount of information in s , while still retaining enough relevant information in order to produce the right prediction. $R(\cdot)$ should be trainable, and trained as a part of $M(\cdot)$, so that it can learn which parts of s is relevant for the model.

This method is realised by implementing $R(s)$ as a module that learns to predict a mask of probabilities P over each square on the board during training. That is, for a board with dimensions (m, n) , each element assigns a probability to its corresponding square on the board. Mathematically, this can be expressed as

$$P_{i,j} = R(s)_{i,j}, P_{i,j} \in [0, 1], \quad (3.4)$$

for any given square given by the matrix indices (i, j) .

This probability matrix is then used as the input for a process that stochastically masks away the information in the corresponding squares. Given an input state s , $R(s)$ is evaluated against a random sample $X \sim \mathcal{U}(0, 1)$. This evaluation is then thresholded by the Heaviside function, which produces a binary mask. This operation can be expressed as

$$M_{bin}(s) = H(R(s) - X), \quad (3.5)$$

where $M_{bin}(s)$ is the binarized mask for the state s , and H is the Heaviside-function. This process can be thought of as sampling each element in $M_{bin}(s)$ as a binary variable with the probability of success equal to the corresponding element in P . This process is illustrated in Fig. 3.9a.

The binarized masks are implemented with the strategy proposed in Sec. 2.4.4. Similar to the masks described in Sec. 3.4, the binary masks are applied to each plane of the input that contains information about the pieces. This procedure is shown in Fig. 3.9b. Additionally, the produced binary mask is concatenated to the masked input state itself, and passed to the main model. This is done in order to provide the model with information about which parts of the input might have been removed during masking. The model can then be trained as a standard neural network, with $R(s)$ being added as a preprocessing step with trainable weights.

After training, the produced probability matrix P can then be used to provide heatmap-based explanations for any state s . Here, these matrices have the interpretable property that the magnitude of any non-zero valued element in P directly corresponds to the perceived importance by the model for the corresponding square in the input.

Chapter 4

Results

4.1 Concept detection

The concept detection strategy described in Sec. 3.3 is applied to the trained 4x5 and 6x6 models described in Sec. 3.2. The majority of the concepts for detection were chosen to align with the main concepts highlighted in McGrath et al. (2021). All probed concepts are listed and described in Table 3.2.

For the 4x5 model, Fig. 4.1 shows that most concept detection curves follow a similar trend, namely quickly flattening out during training. It is also observed that some concepts are somewhat regressable from the input layer, namely that it is possible to linearly deduce the concept using the relevant information in the input state itself. Examples of this can be seen in Figs. 4.1e and 4.1f. The concept `material_advantage`, shown in Fig. 4.1a, stands out for being highly detectable during large parts of the training process.

In the case of the concepts detected in the 6x6 model, shown in Fig. 4.2, one observes a starker difference in how concepts manifest themselves in different parts of the model. This can be seen in Figs. 4.2d, 4.2e and 4.2f. It is also observed that some concepts continue to develop throughout the entire training process, in contrast to the case of the 4x5 model.

The results for the concepts shown in Figs. 4.1a to 4.1d and 4.2a, to 4.2d form the basis for Hammersborg and Strümke (2022), which has been accepted as a publication for the 22nd World Congress of the International Federation for Automation and Control. Therefore, Hammersborg and Strümke (2022) also contains an analysis and discussion of these results.

4.2 Concept backpropagation

The method described in Sec. 3.4 was applied to the last checkpoint of the 6x6 model, while considering the concepts `opponent_double_pawn`, `threat_my_queen`, `threat_opponent_queen`, `in_check`. The probe is inserted at the layer with the best detection result shown in Fig. 4.2. The results are presented in Fig. 4.3.

In most of the cases shown in Fig. 4.3, the proposed method has found a state that achieves maximisation of the given concept. Additionally, all found states except one (shown in Fig. 4.3f) are legal within the rules of chess. Some positions also have some degree of introduced noise in the found perturbations, as seen in Figs. 4.3g, and 4.3d. This phenomenon is discussed further in Sec. 5.2.

4.3 Binary intermediate layers

The strategy described in Sec. 3.5 was applied to a copy of the 4x5 model described in Sec. 3.2. The model was then trained from scratch. The purpose of the described search was to find a binary mask of the legal moves of either player’s king. A selection of the binary activation patterns for an arbitrary position are shown in Fig. 4.4, and it is apparent that the search yielded no positive findings. This was also the case for all probed positions.

4.4 Correlations

The method described in Sec. 3.6 was applied to 300 000 randomly sampled positions for the 4x5 and 6x6 models. The results for the described distance measures are presented in Fig. 4.5.

For the 4x5 model, all three distance measures show high correlations with the concept `has_mate_threat`, as shown in Figs. 4.5a, 4.5c, and 4.5e. This also applies to the 6x6 model, as seen in Figs. 4.5b, 4.5d, and 4.5f.

4.5 Heatmap

The method presented in Sec. 3.8 is evaluated empirically on a selection of positions, and produces sensible results for most of these. The model architecture for producing $R(\cdot)$ is a duplicate of the architecture for the main model, but without the corresponding value and policy heads (described in Sec. 3.2). Results for the 4x5 model are shown in Fig. 4.6. Saliency maps generated by GradCAM for the same positions are shown in Appendix D.

4.6 Counterfactual explanations

Three counterfactual explanations are generated by use of the method described in Sec. 3.7. The positions shown are sampled sequentially from the same game, and are shown in Fig. 4.7. Fig. 4.7a shows a position where the second-best move results in the game being a definite loss, while Figs. 4.7b and 4.7c show positions where the result of the game is not immediately obvious.

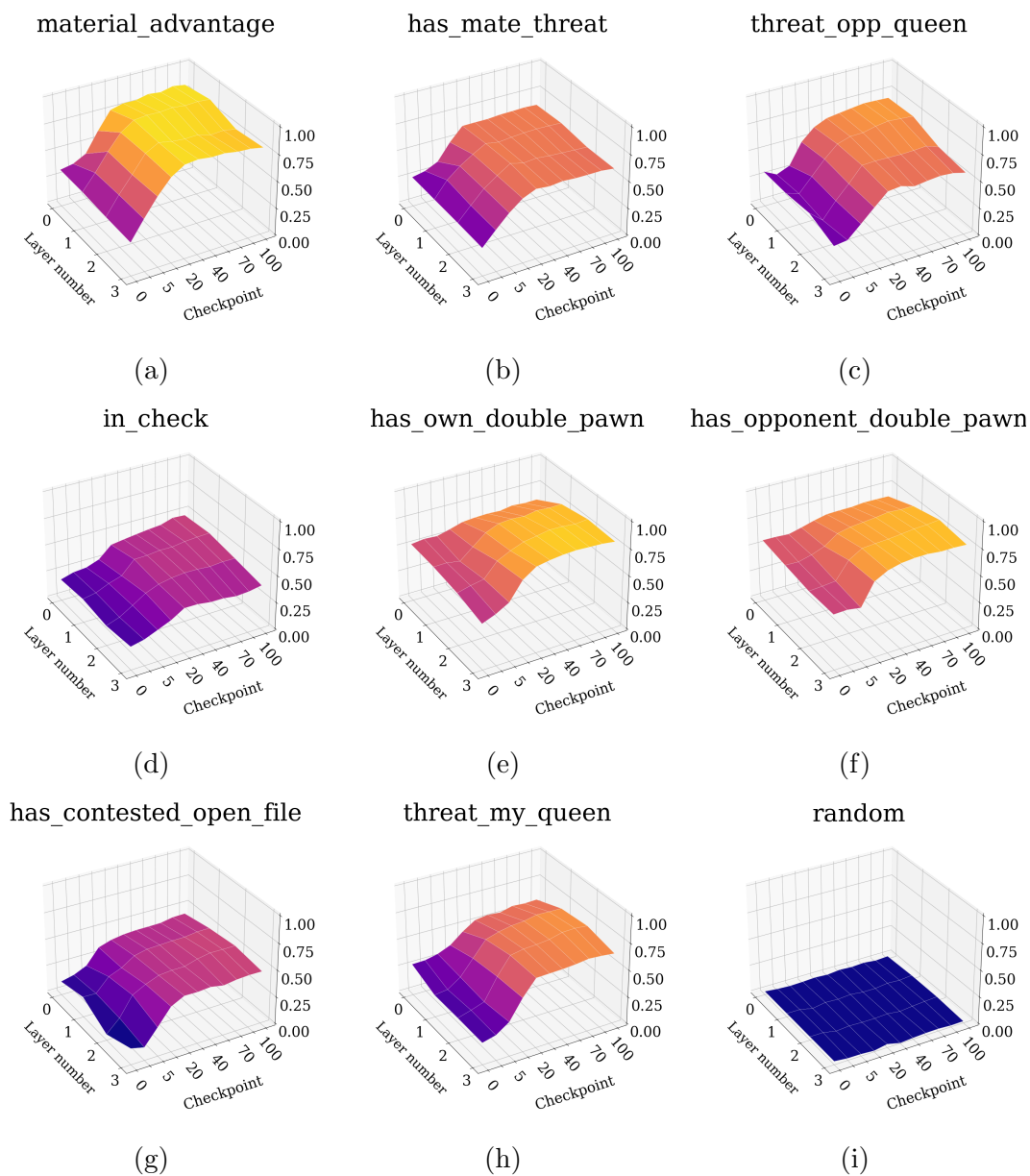


Figure 4.1: Concepts modelled by the 4x5 chess agent, showing the model’s ability to detect (a) whether the player to move has a material advantage, (b) whether the opponent is currently presenting a mate-threat, (c) whether the opponent’s queen is under threat, (d) whether the player to move is in check, (e) whether it has a double-pawn, (f) whether the opponent has a double-pawn, (g) whether both players contest an open file on the board, and (h) whether the player to move’s queen is threatened, and (i) being a sanity check performed on a data set of random labels.

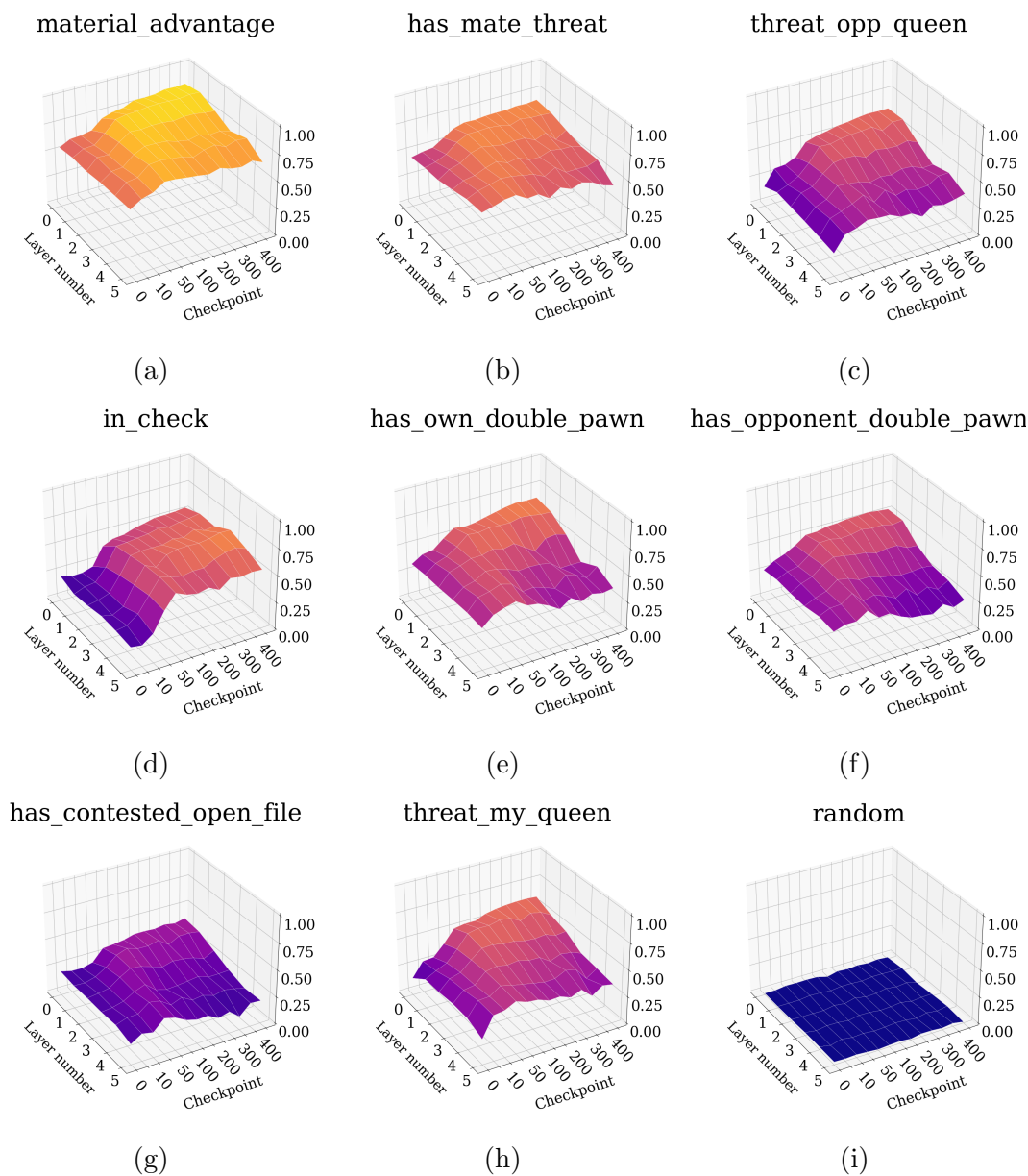


Figure 4.2: Concepts modelled by the 6x6 chess agent, showing the model’s ability to detect (a) whether the player to move has a material advantage, (b) whether the opponent is currently presenting a mate-threat, (c) whether the opponent’s queen is under threat, (d) whether the player to move is in check, (e) whether it has a double-pawn, (f) whether the opponent has a double-pawn, (g) whether both players contest an open file on the board, and (h) whether the player to move’s queen is threatened, and (i) being a sanity check performed on a data set of random labels.

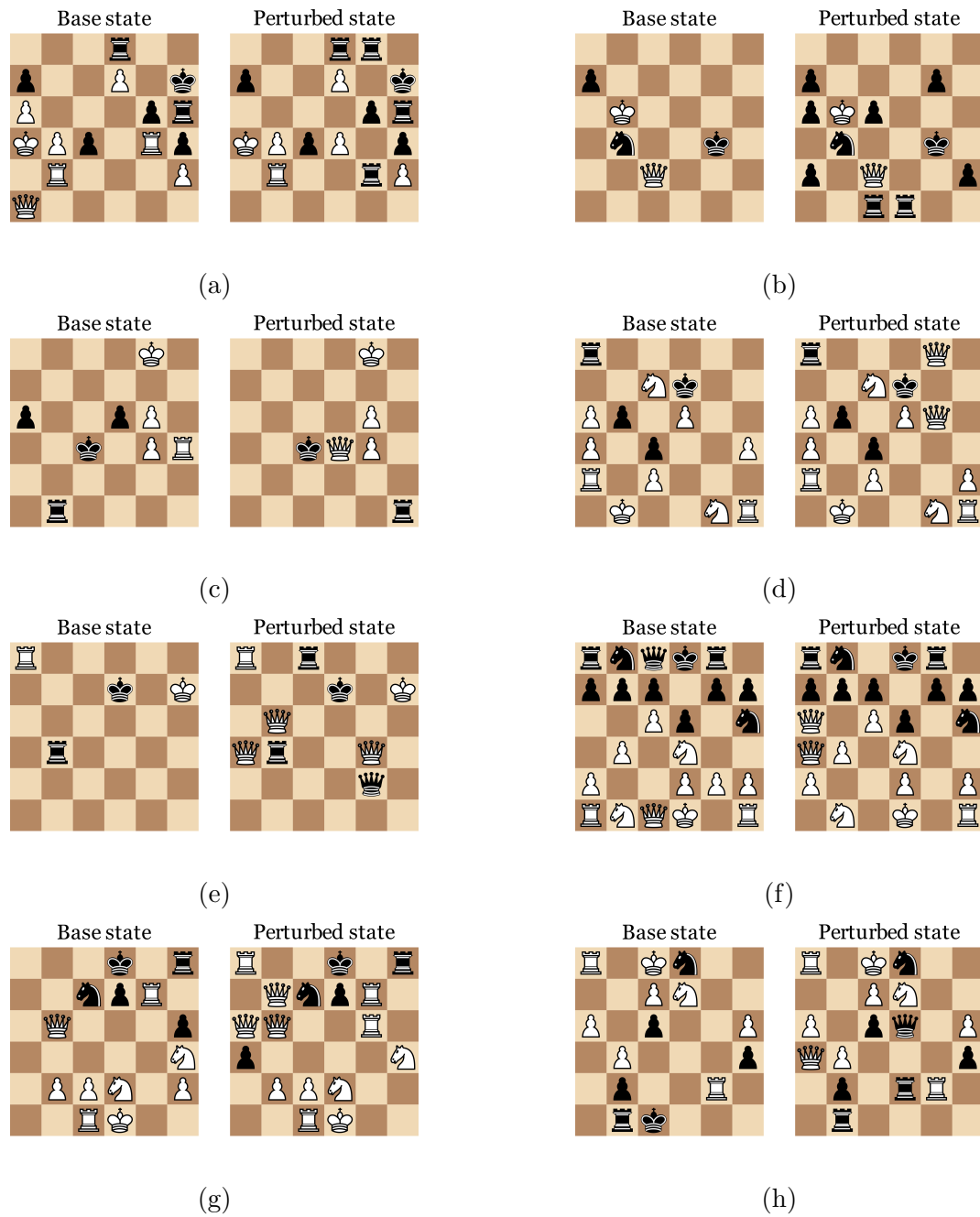


Figure 4.3: Backpropagation of concepts performed on the concepts (a), (b) `has_own_double_pawn`, (c), (d) `in_check`, (e), (f) `threat_my_queen`, and (g), (h) `threat_opponent_queen`.

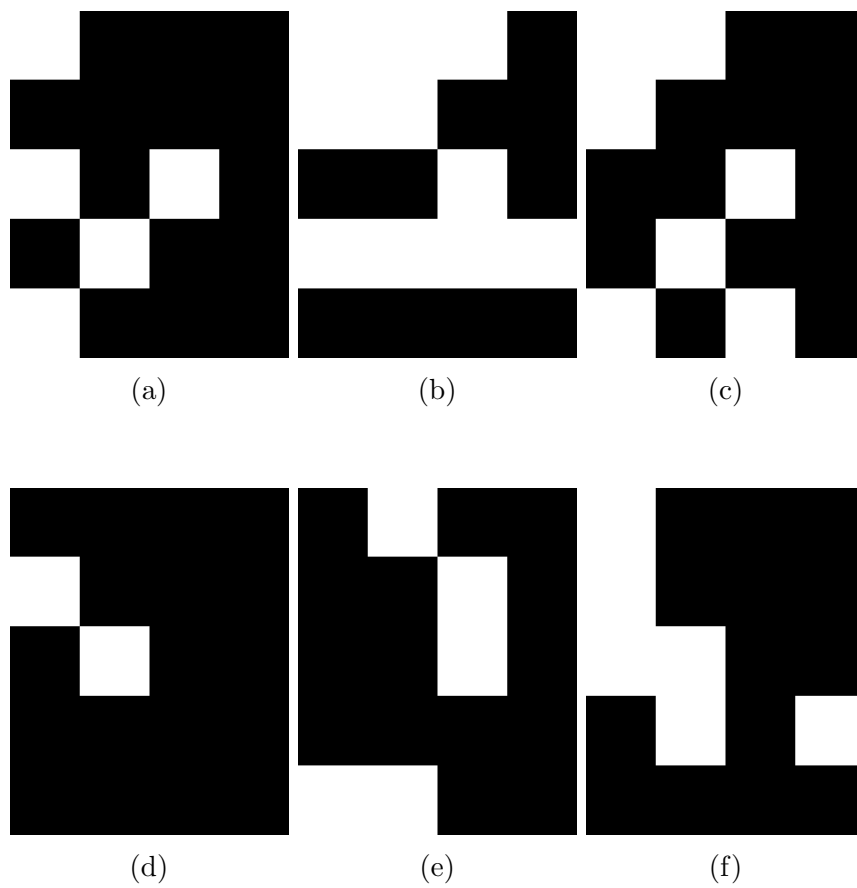


Figure 4.4: The first six channels of binary activation patterns for a given position.

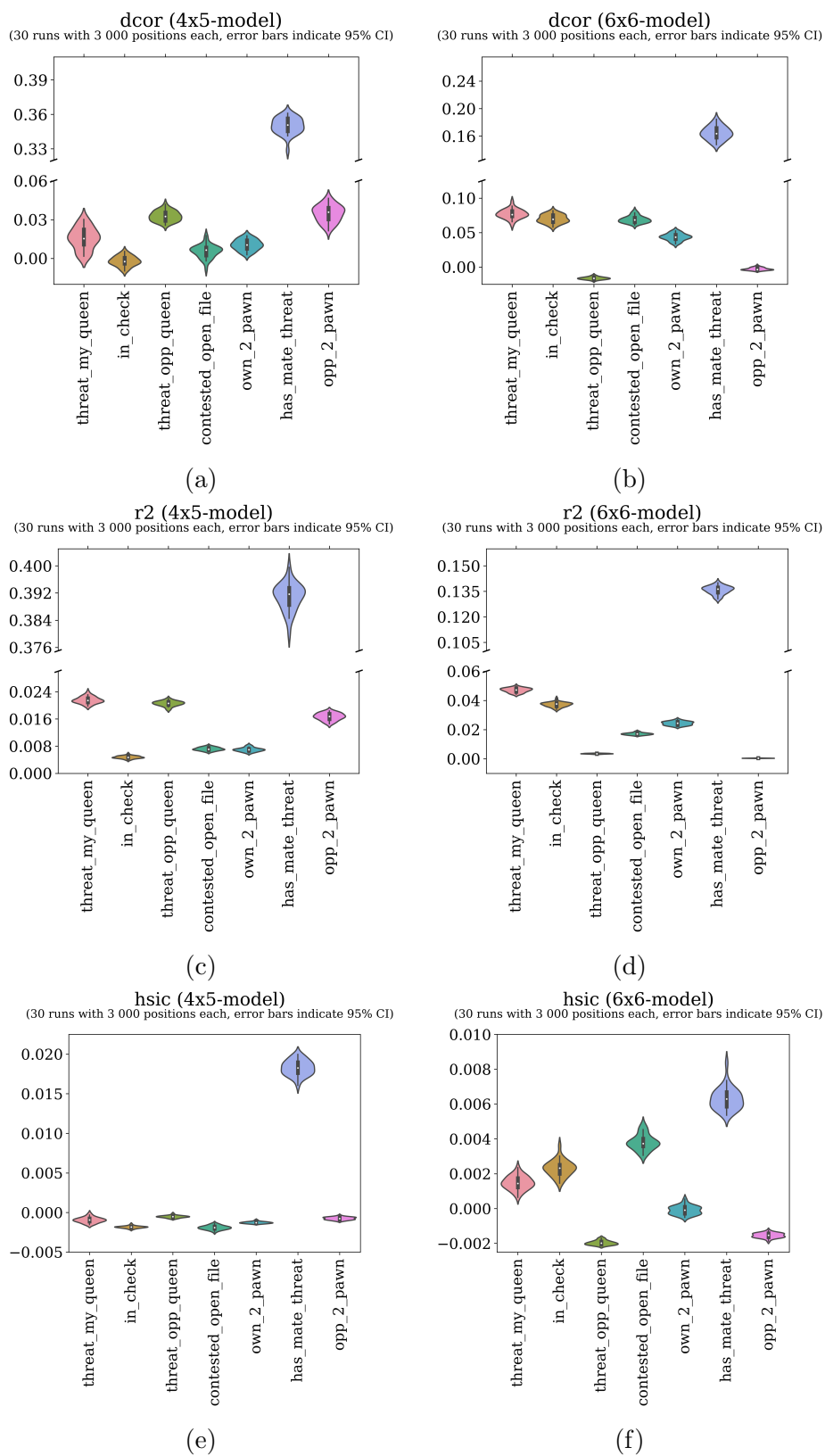


Figure 4.5: Correlation results between concept probes and Q-output using the (a), (b) dcor, (c), (d) r2, and (e), (f) hsic-correlation metric for the 4x5 and the 6x6 model.

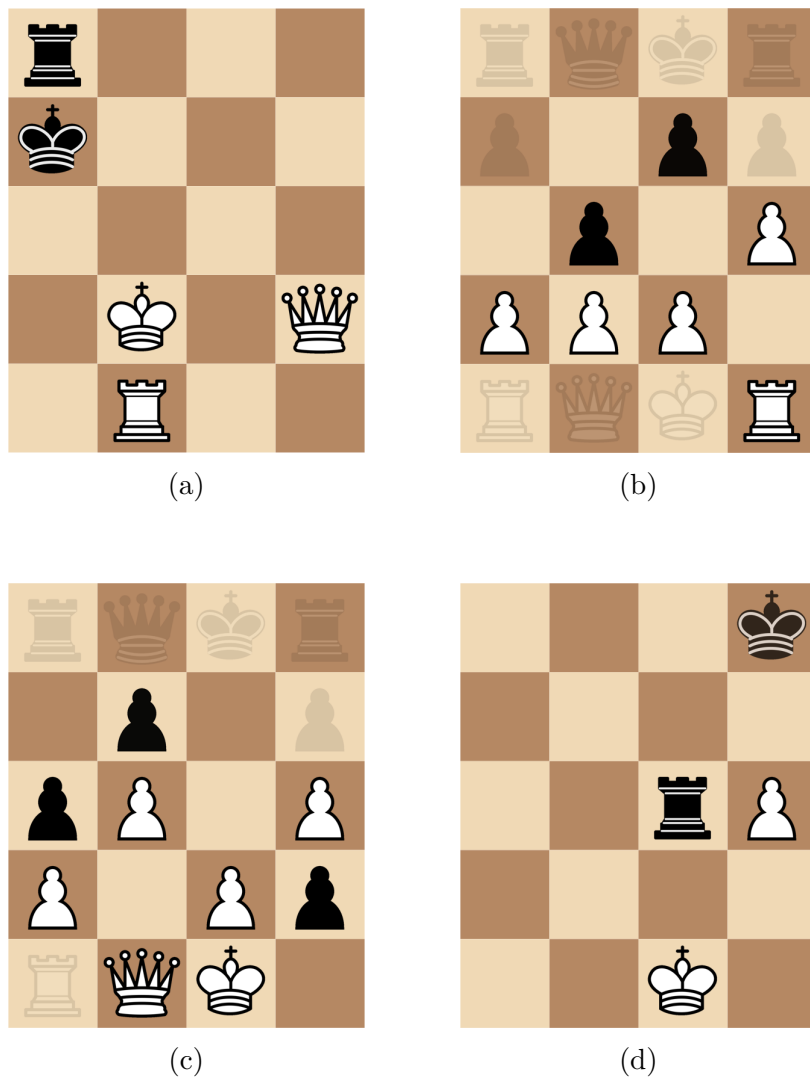
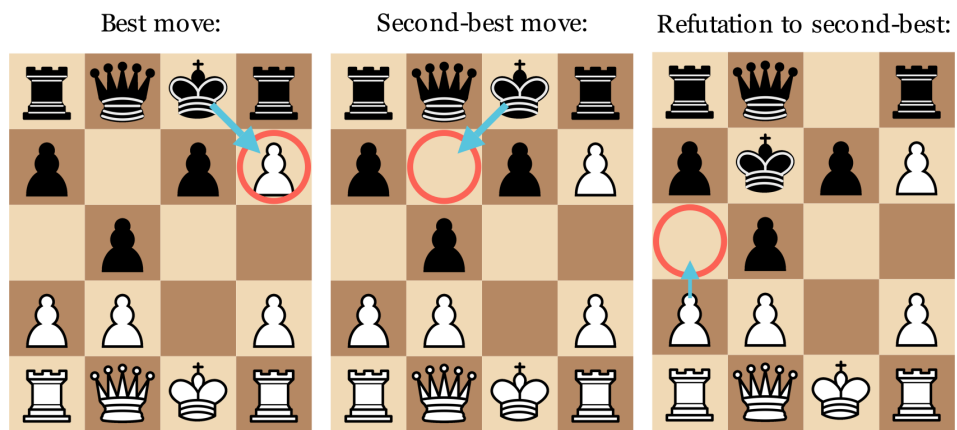
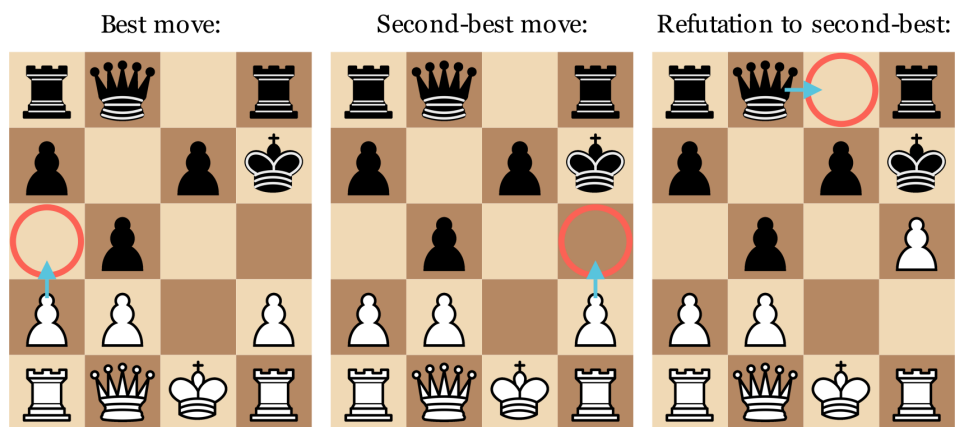


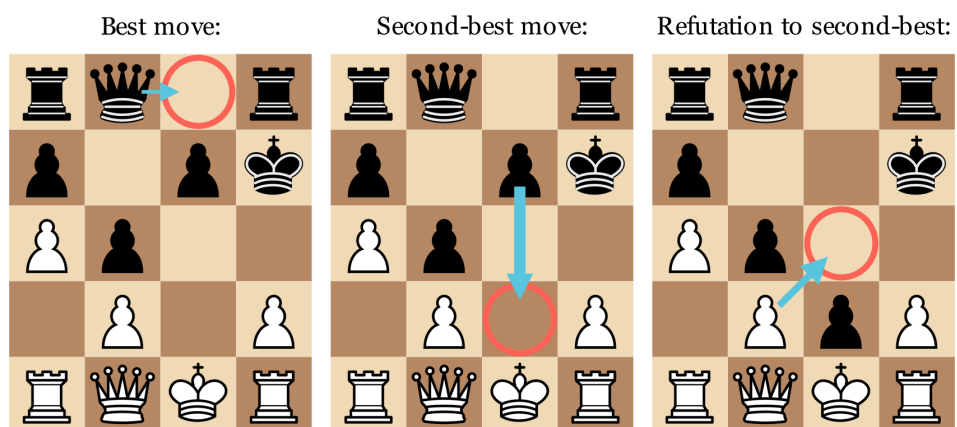
Figure 4.6: Positions with an applied heatmap. The opacity of each piece shows the learned importance per piece for the given position.



(a)



(b)



(c)

Figure 4.7: Generated counterfactuals for a game of 4x5 chess. All positions are sampled sequentially from the same game, with a random move selection process up to the first position shown in (a).

Chapter 5

Discussion

5.1 Concept detection

When considering the results shown in Figs. 4.1 and Figs. 4.2, the first concepts to emerge are highly connected with “accidentally” winning the game. This happens because untrained agents in a MCTS-training loop usually only provide highly random actions. For both discussed chess variants, this means that concepts relating to material advantage (Figs. 4.1a and 4.2a), or concepts relating to threats on own and the opposing player’s queen (Figs. 4.1c, 4.1h, 4.2c, and 4.2h) are among the first to appear. In any case, it makes sense that these appear early, as highly imbalanced positions have a higher probability of leading to decisive outcomes with random play.

The results shown in Figs. 4.1 and Figs. 4.2 also point to several interesting differences between the chess variants used in this work: For the 4x5 model, one surprising observation is that the model seems to have found that this variant has an optimal opportunity leading to a draw for White given optimal play.¹ This is likely the reason for the concept-graphs flattening out during training for the 4x5 model, as seen in Fig. 4.1. This also illustrates how the concept detection method can be used to supervise the training process of the model.

The pawn-centric concepts, seen in Figs. 4.1e, 4.1f, 4.2e and 4.2f, also show a distinct example regarding the role pawn structures play for the presented chess variants. Pawn structures are an important part of standard 8x8 chess, and are often a key decider as to what aspects of the position become crucial. It is observed that these concepts are detected strongly in the 4x5 model, while not being as

¹This was empirically verified using FairyStockfish 14, up to a game-tree depth of 90.

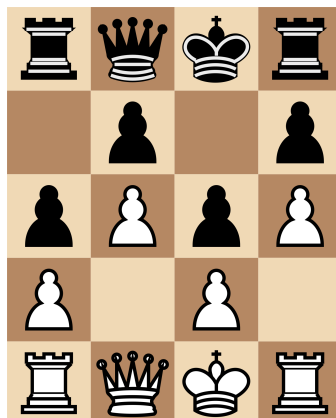


Figure 5.1: A typical position occurring under optimal play in the presented 4x5 chess variant. Observe that none of the pawns for either player have any legal moves.

clearly detectable in the 6x6 model. This might be due to the closedness of 4x5 chess: For the 4x5 chess variant, the observed optimal solution occurs by locking most of the central squares with one’s own and the opposing pawns (shown in Fig. 5.1). This means that any model wishing to capitalise on positions that are not locked off, could use the notion of “doubled pawns” to identify such positions.

It is also worth noting that this method implicitly assumes that there is a direct correlation between concepts being “detectable”, and concepts being represented linearly. This has been empirically observed as likely, as described in Alain and Bengio (2017), but is not something that can be guaranteed. If one were to think of “detected concepts” as being metrics of information content, a detected concept can only serve to confirm the presence of the information required to deduce the concept in the given layer. However, if a concept is not detectable, that could mean that the required information is present, but not represented linearly. However, as additional empirical evidence, this linear-concept phenomenon can be observed to happen with the `incheck`-concept for the 6x6 model, as seen in Fig. 4.2d. Here, the detectability increases in the later layers of the model, meaning that these layers can be thought of as “distilling” the information required for detecting the concept.

In this vein, one can also consider the effect of sparsity on the weight matrix \mathbf{w} for the given trained probe. Even though the probe is restricted to only find linearly-represented concepts, the L1-penalty discussed in Sec. 3.3 also ensures that the probe picks up on concepts that are sparsely represented. The weight of the applied L1-penalty then indirectly decides how sparsely the concepts need to be represented in order to be detectable. At first glance, it is not obvious if this

feature of the method is desirable. In McGrath et al. (2021), this requirement for sparsity is justified by claiming that this is required from keeping the probes from learning complex representation of their own. This claim is also used to warrant not using more capable probes. In this case, not allowing for complex representations means that the probes are more likely to only detect concepts when they are in fact distilled by the model itself, and not just when enough information is present to create the mapping in the probe. This means that a weak probe is more likely to only detect the concepts when the model itself has “learned” to extract them, which is the intention of the method. However, this again hinges on the main assumption that the models need to represent the given concepts linearly for them to be detectable.

5.2 Concept maximisation

The method described in Sec. 3.4 aims to provide a method for visualising how select concepts in Table 3.2 are represented by the given model. This is done by searching for which parts of the input need to be changed in order to maximise the given concept. In general, this seems to work well. However, the method seems to struggle in some cases where the learned probe is trained to detect concepts that appear symmetrically. An example of this can be seen in Figs. 4.3g and 4.3h. Even though the method in both cases produce states where a queen is threatened, in Fig. 4.3e the new threat is on the wrong player’s queen(s). Intuitively, this makes some degree of sense. In the presented position, White has a large material advantage, and is by much more likely to win given adequate play. Black has lost its queen, meaning that maximising the threat on Black’s queen would require inserting a new queen on their side. This would drastically change the nature of the position. One could therefore think that inserting more pieces for White is likely less of an internal shift for the model, since the position is already drastically in White’s favour. It would therefore be logical that an incomplete representation of the concept `threat_opp_queen` would result in the position shown in Fig. 4.3e, while not strictly maximising the given concept.

The utilisation of this method also requires suitable weights to be found for the $L1$ -penalties applied to s^+ and s^- , as described in Algorithm 4. These weights define how important it is to remove or add information to s , respectively, and as such also affect how difficult it is to find a perturbed state that maximises the given concept. In some cases, this might make it challenging to find the correct balance between the difficulty in finding a relevant perturbation, and the relevancy of the maximised states themselves. This is exemplified through the case shown in Fig. 4.3b: Here, the perturbed state seems to display the correct representation for

the concept `has_own_double_pawn`, but it also introduces a non-negligible amount of noise, meaning that many of the added pieces should not be relevant for the given concept. This makes it somewhat harder to determine which parts of the perturbed states represent what the model has learned about the given concept.

It is also worth highlighting that generating semantically similar positions is in many cases orthogonal to maximising a given concept. This aligns with the main intention of the method, namely being able to visualise the internal representation of a given concept. Since it is sometimes difficult to find sparse s^+ and s^- , this might result in cases where valid representations of a concept are generated at the cost of the “sensicalness” of the position. An example of this is shown in Fig. 4.3h, where the generated perturbed state has no Black king. This is in reality something that should be prevented by the additional “legality loss” introduced as an additional maximisation-objective. In technical terms, this is due to the method being unable to minimise both l_p and l_c at the same time, as described in Algorithm 4.

The method described in Sec. 3.4 is also envisioned to be easily applied on other problem cases where concept detection is relevant. This is especially true for cases with continuous input spaces, and especially when there are no requirements wrt. semantic similarity or “sensibleness” to the generated positions.

5.3 Shapley values in concept space

Given a model M with inputs I_m and outputs O_m , the Shapley value φ_i for input component i assigns the share of the total contribution the input feature has on the outputs O_m . In the original formulation of the Shapley value, this is calculated through the definition shown in Eq. 2.13. Through this equation, the Shapley value represents the contribution of each feature by considering all possible combinations of the involved features.

Applying such a method to the domain of chess involves numerous challenges. Firstly, the input space of chess is ill-suited: It is discrete, with all input-components being highly interdependent, meaning that the value of a piece on a given square might depend on almost all other pieces present the board. These dependencies are almost always also decided on a position-by-position basis, meaning that it is near-impossible to define a static characteristic function for all such subcoalitions of pieces.

For an explanation based on the Shapley value to be valuable, one needs to redefine the input space with regard to which these values are calculated. From Sec. 2.6.3, one could expect that an interesting prospect would be to create a mapping from

the inputs I_m to an n -dimensional concept vector C , where each element in C designates the presence of a specified concept given by pre-trained linear probes for some intermediary layer L in the model. Being able to operate on this as an assumed representation of the input of the actual model, would mean that the Shapley values could now be calculated on the concept vector, instead of the chess positions. Since the probes which construct C are logistic models, this would mean that $C \in \mathbb{R}^n$, which is necessary for generating valid perturbations of the now defacto input space. This would also create useful explanations, being able to attribute changes in the evaluation of the position to higher-level concepts.

Such an approach, however, has a major issue. While it is easy to find the mapping $L \rightarrow C$, it is not as feasible to find an inverse mapping $C \rightarrow L$. Using the concept backpropagation method from Sec. 3.4 is a theoretical possibility, but it is too slow in practice. A simplified version of this method, only incorporating the movement towards the steepest gradient, also often fails to converge to the desired concept output, especially when each concept vector in C has many elements. However, with a given state s_n , for any pair of output and concept vectors L_n and C_n , it is possible to create a set of equations to be solved in order to generate a state L_n^* that gives the desired concept vector C_i when passed through the trained probes.

Let P_i be the trained probe that has learned to identify the i -th concept. Additionally, let C be the desired output vector so that the i -th element is the output of i -th probe when provided with the desired state L_n^* . Since P_i is a logistic probe, this can be written as

$$\sigma(\mathbf{w}_i \cdot (L_n^*) + \mathbf{b}_i) = C_i, \quad (5.1)$$

where \mathbf{w}_i and \mathbf{b}_i are the weight and bias matrices, respectively, for P_i . The derivation of this equation can be found in Appendix B. Eq. 5.1 can then be rewritten to provide a solution for finding L_n^* that satisfies the equation for the combined weight matrices, biases, and concept vector \mathbf{w} , \mathbf{b} , and C

$$\mathbf{w}^{-1}(\sigma^{-1}(C) - \mathbf{b}) = L_n^*. \quad (5.2)$$

Here, σ^{-1} is the inverse sigmoid function, and \mathbf{w}^{-1} is the left-inverse of \mathbf{w} . The left inverse (Moore, 1920) is computed as

$$\mathbf{w}^{-1} = (\mathbf{w}^T \mathbf{w})^{-1} \mathbf{w}^T. \quad (5.3)$$

The operation described in Eq. 5.2 can then be performed in order to generate a valid L_n^* for any given concept vector C . The derivation of Eq. 5.2 can be found in Appendix B.

Although Eq. 5.2 can provide a quick, exact solution, there is no guarantee that L_n^* will resemble other states from the activation space L . In many ways, this

renders the perturbations meaningless, since they are not samples that any use of the model M would sensibly generate. A possible solution to this could be to solve Eq. 5.2 while minimising the distance to a sample of other vectors in L . However, any such solver is too slow when combined with the rest of the routine.

Additionally, for this method to be viable one would need C to be an exhaustive description of the input position. If not, this means that the outputs of the model for a given position might be adversely affected by information not available in C , thus rendering the generated explanations useless. Ensuring that C is exhaustive is almost impossible, since this would mean a priori knowing what information the model represents in its intermediary layer L . While Shapley values are a popular part of the XAI literature, and the described method seems promising at first glance, it is therefore concluded that it is not likely to produce promising results.

5.4 Correlations

The results presented in Sec. 4.4 are interesting for the selected chess variants used in this work. Most notably, it is observed that `has_mate_threat` has a significantly high correlation with the predicted outcome for the 4x5 model. This means that a high amount of threats made in all observed games lead to situations where the threat can be carried out. This is hypothesized to be due to the closedness of 4x5 chess, as previously discussed in Sec. 5.2. Practically, this can be thought of as the king very often being blocked by pieces of the same colour, which is induced by the size of the board. However, for all variants of chess, it is undoubtably true that making threats is beneficial to winning. This is because the notion of threats is interlinked with the game being over – one loses when one is not able to defend against all presented threats on the board.

For the presented method, the correlations were calculated based on the predicted outcome for a given set of models. However, using these correlations to explain the chess variant itself means that one implicitly assumes that the models are capable of providing unbiased evaluations of any given position. For smaller problem spaces, such as the chess variants presented in this work, this can reasonably assumed to be true. While the methodology continues to be viable for larger problem spaces, it does not necessarily lend itself to the same interpretation. In this case, the arising problem can be thought of as an alignment problem. The true goal of a chess model is to win chess games, and the evaluations for an unbiased model for any such position should reflect this. But for imperfect models, the calculated correlations between detected concepts and game outcome might say more about how the models themselves have learned to approach any given position, instead of directly representing the true nature of the given chess variant.

5.5 The viability of counterfactual explanations for chess

While the notion of counterfactual explanations for chess seems enticing, these are difficult to generate adequately. Finding some small perturbation s^* to produce a desired policy vector $p(s + s^*)$ is difficult for discrete input spaces, including chess. Any perturbation strategy aiming at producing a viable alternative state also needs to impose some restrictions on how these perturbations can be chosen. Although one is in theory free to find any perturbation that generates the desired $p(s + s^*)$, the usefulness of counterfactual explanations hinges on the perturbed states being likely to represent regular game-play. Additionally, one could also argue that it is necessary for the perturbed states to be related to the original state s . This is because the generated states might need to have some relevance to the original state in order for them to serve as adequate counterfactual explanations. This also raises the question as to how such a similarity measure should be defined, as it is hard to quantify what constitutes “similarity” between positions in chess.

The methodology described in Sec. 3.7 tries to circumvent some of the proposed problems wrt. counterfactual explanations for chess. The produced game tree is implicitly used to address the question of similarity, where states that share common parent states are regarded as similar. The specificity the desired policy vector $p(s + s^*)$ is also simplified, in this case to consider only the first and second highest policy elements of the original policy vector $p(s)$. The counterfactual explanation is then presented in the form of a rebuttal a^* to the state s_2 reached by the second highest policy element in $p(s)$. However, this method has some shortcomings: The most obvious problem is that it is not guaranteed that a^* presents enough information as to effectively rule out the viability of traversing to s_2 . It is difficult to know whether the expected value loss of allowing a^* is evident directly after a^* , or instead only becomes evident after later moves. This in turn would require being able to enumerate and explain more states after performing a^* , which quickly becomes unmanageable.

Despite the discussed shortcomings, the method produces some successful results. In Fig. 4.7a, it is observed that the highlighted “second best move” and “refutation” show a pattern that results in immediate loss for the opponent, and the presented “if-this-then-that”-statement therefore constitutes a valid counterfactual explanation, given the requirements described in Sec. 3.7. However, both. Figs. 4.7b and 4.7c show situations where it is not immediately obvious how the game state is affected by the suggested moves, which in turn means that the explanation loses most of its usefulness.

The proposed method additionally fails to be relevant in situations where the given

position does not warrant a “refutation” given optimal play. Common examples of this in the chess terminology are known as “quiet moves” (Akl and Newborn, 1977). This failure case is also relevant where two moves are both in some senses optimal, and it would be incorrect to present one of them as the best move.

5.6 Do methods need to be model agnostic?

When considering any proposed explanatory method for chess playing models, it is also worth discussing whether such methods need to be model agnostic. Many XAI methods have wide applicability due to their model agnostic nature, and is therefore something to be investigated for the methods presented in this work as well. The methods proposed in Secs. 3.3 to 3.8, excluding Sec. 3.6, all benefit from information inherent to the structures provided by neural networks. However, some methods present a greater reliance on this than others: The concept detection method presented in Sec. 2.6.3 largely relies on the model having some internal structure, which does not explicitly exclude other types of models. Both the concept backpropagation (Sec. 3.4), and the heatmap methods (Sec. 3.8), on the other hand, rely on being able to access gradients that inform training of added structures, and are as such highly specialised for neural networks. Lastly, neither the correlation measures based method nor the counterfactual method have requirements regarding model type.

Even though the most powerful methods discussed are made specifically for neural network models, this might not be as limiting in the case of explaining chess models as for applications of XAI methods in general: One can expect that neural-network-based models will continue to be relevant for MCTS-based chess engines, implying that methods tailored for these will continue to be useful. Additionally, both the input and output spaces of chess are difficult to work with, requiring methods to be specifically adapted, which makes it difficult to develop generalisable methods.

5.7 Information content

While the methods presented in Secs. 3.8 and 2.6.3 are both adept at answering partial questions regarding what information the model utilises, it is still relevant to assess the information content of what is actually being presented. The two methods are similar in this regard, as they both indicate what information the model is using, but not necessarily *how* the information is used. For concept detection, the method communicates what aspects (i.e. concepts) are important when evaluating a chess position. It does not, however, guarantee that the degree to which a concept is detected directly correlates with the importance of the con-

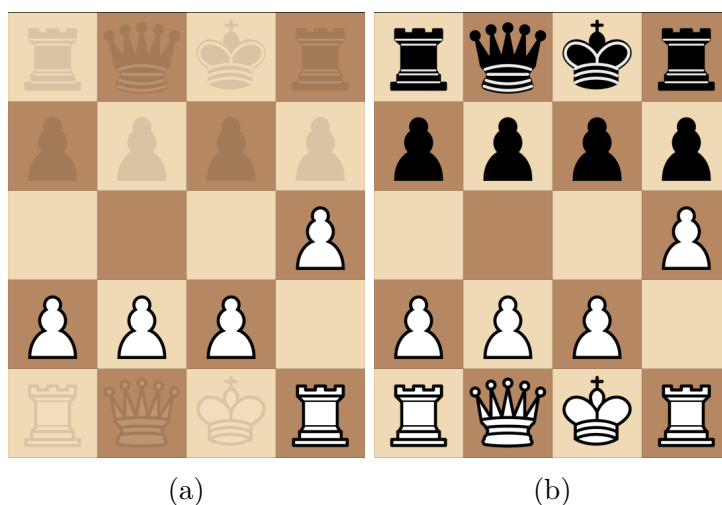


Figure 5.2: An example position simplified with a heatmap (a), and the same position (b), with no simplifying heatmap.

cept. Similarly, for the heatmap method, it can in fact be guaranteed that the model is trained to not have access to the parts of the position not highlighted by the heatmap. This can, in many cases, greatly simplify the process of deducing what the model deems important. However, in the edge case where the heatmap highlights all the available pieces on the board, it does not act as a reductive model, and says nothing about what parts of the position are actually important. However, this might mean that the position as a whole is non-reductive - i.e. that removing even a single piece from the position would leave the model unable to accurately assess it.

There is also something to be said when considering the information content in the binary activation patterns, such as those presented in Sec. 4.3. Although the amount of information per channel is significantly reduced, the information content in aggregate is most likely unaffected. In many ways, this highlights an interesting idea when trying to assess what information is present in a given layer. In the present case, the idea was to sparsely distribute the information of the given layer, while simultaneously forcing it to operate with the same dimensionality and data type as the input. Ideally, this would create a map between the given binary intermediate layer and the input space. Nevertheless, this did not happen in practice. It might however still be promising to sparsify the informational content in this manner, where such a strategy might for example open up for novel strategies using binary activation patterns to produce explanations (such as facilitating concept detection).

5.8 What makes for a good explanation?

The assessed value of an explanatory method depends on the purpose of the produced explanation, i.e. what kind of knowledge one wants to extract from the model. In an imagined teacher-student scenario, where a chess player wants to query the model for its evaluation of a given position, one could think that the heatmap method would be very relevant. It does not necessarily state how the evaluation is calculated from the given position, but might help in knowing where to look in the given position as to make a correct judgement. An example of this in practice can be seen in Fig. 5.2. Here, the proposed method has been used to greatly simplify the given position in order to aptly communicate which parts of the position are relevant.

In cases where more global explanations are required, one might consider the concept related methods presented in Sec. 2.6.3 and 3.4, and the correlation-based method presented in Sec. 3.6. These methods might aid in making judgements about both the models and the chess variants they play, as a whole. Under the assumption that the models reach an adequate playing strength, their behaviour can be used as a proxy for what is deemed to be close-to-optimal play. This means that they can provide useful heuristics for what makes positions and moves viable. An example of this can be seen by considering the correlation between the game outcome and the concept `has_mate_threat` for the 4x5 model, shown in Figs. 4.5a, 4.5c and 4.5e, and discussed in Sec. 5.4. Knowing that threats are disproportionately advantageous in 4x5 chess, might for example lead to a more aggressive playing style than what is fruitful for other chess variants.

Another point that should be considered when evaluating the presented explanatory methods is the various guarantees that each method makes wrt. verification. The presented heatmap method is interesting in this context, as it makes strong guarantees regarding what information actually reaches the model. Conversely, the proposed concept detection method can only make weaker guarantees in this area, as discussed in Sec. 5.1. However, one could imagine combining this method with the proposed concept maximisation method, in order to verify the internal representation of each concept in the model.

Chapter 6

Conclusion

6.1 Contributions

This thesis makes several contributions. Addressing the first research question posed in Sec. 1.2, the work presents a chess environment which serves as a viable alternative for training AlphaZero-like chess models on smaller chess variants. Additionally, the produced chess environment provides adequate functionality allowing it to be easily extendable, meaning that it is possible to use it as a testing ground for new explanatory techniques for chess. This is shown by being able to produce similar results and metrics for detecting concepts from DRL-based chess models, as described in McGrath et al. (2021).

With regards to the second research question posed in Sec. 1.2, the thesis presents several novel explanatory methods which can be applied to DRL-based chess models. The concept backpropagation method, described in Sec. 3.4, builds on the existing concept detection method, and allows for effective visualisation of which aspects of a given concept are represented in the model. The notion of effective visualisation is also relevant for the presented heatmap method (Sec. 3.8), which illustrates the possibility of providing a novel importance-based heatmap for any given chess position. The work also presents model agnostic methods, exemplified through the correlation-measure-based method (Sec. 3.6). This shows how it is possible to use outputs from a given model, in aggregate, to see how the concepts it learns to represent are correlated with the predicted outcome of the game. This then provides a global explanation of the chess variant itself, indicating what concepts are likely to be important for optimal play.

6.2 Future work

6.2.1 Architectures for linear probing

Even though the methodology presented in Sec. 2.6.3 can be applied to any neural network architecture, it is also relevant to examine how the architecture of the model might facilitate the detection of concepts. As discussed in Sec. 5.1, the proposed method implicitly assumes that any concept that is sufficiently utilised in the model should also be linearly detectable. That is, if a model sufficiently distills a concept, the assumption is that it is represented linearly. However, it is reasonable to assume that some architectures might be better suited to meet this assumption. In this vein, one possible avenue to explore is the relation between the sparsity of the activation space, and the linear representation of concepts. Additionally, it would be interesting to consider crafting custom layer structures facilitating linear concept representation.

6.2.2 Methodologies for concept backpropagation

While the results presented in Sec. 4.2 show that the concept backpropagation method is adequate for visualising the model’s understanding of a given concept, it is not obvious how to best incorporate information from all layers of the model. As described in Sec. 3.4, the probe due to be maximised is inserted at the layer of the model where the given concept was most likely to be detected, but one might achieve better results by simultaneously maximising the given concept in *all* layers of the model instead. While this seems promising, it turns the problem into a multi-objective optimisation problem. It is hypothesised that the viability of the method in part is due to the simplicity of maximising a single linear probe (and the accompanying legality module), and maximising multiple such probes at once might make the method unfeasible. This might of course vary based on the given model architecture, but it would nevertheless be interesting to consider more complex probing methodologies that could utilise information across multiple layers of the model.

6.2.3 Extending the alternative heatmap explanation

The method for generating heatmap-based explanations, described in Sec. 3.8, was created to specifically accommodate many of the difficulties involved in generating explanations for chess. However, the method is also applicable to other domains, as an alternative to saliency-based explanation methods. In this case, it could be used to provide saliency maps for image models, or create relevancy maps for regression in cases where the Shapley values on tabular data are traditionally used.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

Selim G. Akl and Monty Newborn. The principal continuation and the killer heuristic. In *ACM '77*, 1977.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *ArXiv*, abs/1610.01644, 2017.

L. Atkin and D. Slate. *Chess 4.5-The Northwestern University Chess Program*, pages 80–103. Springer-Verlag, Berlin, Heidelberg, 1988. ISBN 0387913319.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL <http://arxiv.org/abs/1308.3432>.

Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artif. Intell.*, 134(1-2):57–83, jan 2002. ISSN 0004-3702. doi: 10.1016/S0004-3702(01)00129-1. URL [https://doi.org/10.1016/S0004-3702\(01\)00129-1](https://doi.org/10.1016/S0004-3702(01)00129-1).

Guillaume M. J. B. Chaslot, Mark H. M. Winands, and H. Jaap van den Herik. Parallel monte-carlo tree search. In H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, and Mark H. M. Winands, editors, *Computers and Games*, pages 60–71, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers, editors, *Computers and Games*, pages 72–83, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-75538-8.

Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016. URL <http://arxiv.org/abs/1602.02830>.

- Susanne Dandl, Christoph Molnar, Martin Binder, and Bernd Bischl. Multi-objective counterfactual explanations. In *Parallel Problem Solving from Nature – PPSN XVI*, pages 448–469. Springer International Publishing, 2020. doi: 10.1007/978-3-030-58112-1_31. URL https://doi.org/10.1007/978-3-030-58112-1_31.
- Anupam Datta, Shayak Sen, and Yair Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 598–617, 2016. doi: 10.1109/SP.2016.42.
- Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.
- Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL <http://arxiv.org/abs/1412.6572>.
- Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In Sanjay Jain, Hans Ulrich Simon, and Etsuji Tomita, editors, *Algorithmic Learning Theory*, pages 63–77, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31696-1.
- Patrik Hammersborg and Inga Strümke. Reinforcement learning in an adaptable chess environment for detecting human-understandable concepts, 2022. URL <https://arxiv.org/abs/2211.05500>.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL <http://arxiv.org/abs/1512.03385>.
- Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *CoRR*, abs/1610.02136, 2016a. URL <http://arxiv.org/abs/1610.02136>.
- Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016b. URL <http://arxiv.org/abs/1606.08415>.
- Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: [https://doi.org/10.1016/0893-6080\(91\)90026-8](https://doi.org/10.1016/0893-6080(91)90026-8).

org/10.1016/0893-6080(91)90009-T. URL <https://www.sciencedirect.com/science/article/pii/089360809190009T>.

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2668–2677. PMLR, 10–15 Jul 2018.

Donald Ervin Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artif. Intell.*, 6:293–326, 1975.

Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In *NIPS*, 1989.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf>.

Dhruv Mahajan, Ross B. Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. *CoRR*, abs/1805.00932, 2018. URL <http://arxiv.org/abs/1805.00932>.

Irwin Mann and Lloyd S. Shapley. *Values of Large Games, IV: Evaluating the Electoral College by Montecarlo Techniques*. RAND Corporation, Santa Monica, CA, 1960.

Thomas McGrath, Andrei Kapishnikov, Nenad Tomasev, Adam Pearce, Demis Hassabis, Been Kim, Ulrich Paquet, and Vladimir Kramnik. Acquisition of chess knowledge in alphazero. *CoRR*, abs/2111.09259, 2021. URL <https://arxiv.org/abs/2111.09259>.

Nicholas Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949. ISSN 01621459. URL <http://www.jstor.org/stable/2280232>.

Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *CoRR*, abs/1706.07269, 2017. URL <http://arxiv.org/abs/1706.07269>.

- E.H. Moore. The fourteenth western meeting of the American Mathematical Society. *Bulletin of the American Mathematical Society*, 26(9):394–395, 1920.
- Tord Romstad, Marco Costalba, and Joona Kiiski. Stockfish. <https://stockfishchess.org/>, 2022. [Online; accessed 30-November-2022].
- Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. *American Journal of Psychology*, 76:705, 1963.
- Christopher D. Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61:203–230, 2011.
- Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016. URL <http://arxiv.org/abs/1610.02391>.
- Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *CVPR*, 2023.
- Lloyd S. Shapley. *Notes on the N-Person Game I: Characteristic-Point Solutions of the Four-Person Game*. Santa Monica, CA, 1951. doi: 10.7249/RM0656.
- Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. *CoRR*, abs/1704.02685, 2017. URL <http://arxiv.org/abs/1704.02685>.
- David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404. URL <https://www.science.org/doi/abs/10.1126/science.aar6404>.
- Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016. URL <http://arxiv.org/abs/1602.07261>.
- Gábor J. Székely, Maria L. Rizzo, and Nail K. Bakirov. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769 – 2794, 2007. doi: 10.1214/009053607000000505. URL <https://doi.org/10.1214/009053607000000505>.

John von Neumann, Oskar Morgenstern, and Ariel Rubinstein. *Theory of Games and Economic Behavior*. Princeton University Press, 1944. ISBN 9780691130613.

Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowl. Inf. Syst.*, 41(3):647–665, dec 2014. ISSN 0219-1377. doi: 10.1007/s10115-013-0679-x. URL <https://doi.org/10.1007/s10115-013-0679-x>.

Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *CoRR*, abs/1711.00399, 2017. URL <http://arxiv.org/abs/1711.00399>.

Appendices

A A selection of edge-cases for calculating legal moves for a given chess position

While all pieces in chess have discrete movement rules, as discussed in Secs. 2.1 and 3.1, the notion of calculating the legal moves in a given position can still be quite complex. Rules regarding piece movement cannot be applied blindly, as the legality of any given chess position takes precedent in such cases. An example of this can be shown in Fig. A.1. Here, blindly allowing the highlighted move would lead to an illegal position, as this would uncover a threat on the player to move's own king. In this case, this example can be thought of as an extension of the blocker-type movement rules discussed in Sec. 3.1.2, but it is nevertheless an example of the considerations that need to be made when creating a procedure for calculating legal moves quickly.

B Equation for maximising single probe-output

$$\begin{aligned} P(L_n^*) &= C_i \\ \sigma(\mathbf{w}_i \cdot L_n^* + \mathbf{b}_i) &= C_i && | \text{ Inserting } P_i(L_n^*) = \sigma(\mathbf{w}_i \cdot L_n^* + \mathbf{b}_i) \\ \sigma^{-1}(\sigma(\mathbf{w}_i \cdot L_n^* + \mathbf{b}_i)) &= \sigma^{-1}(C_i) && | \text{ Solving for } L_n^* \\ \mathbf{w}_i \cdot L_n^* + \mathbf{b}_i &= \sigma^{-1}(C_i) \\ \mathbf{w}_i \cdot L_n^* &= \sigma^{-1}(C_i) - \mathbf{b}_i \\ \mathbf{w}_i^{-1} \cdot \mathbf{w}_i \cdot L_n^* &= \mathbf{w}_i^{-1} \cdot (\sigma^{-1}(C_i) - \mathbf{b}_i) && | \mathbf{w}^{-1} = (\mathbf{w}^T \mathbf{w})^{-1} \mathbf{w}^T \text{ (Moore, 1920)} \\ L_n^* &= \mathbf{w}_i^{-1} \cdot (\sigma^{-1}(C_i) - \mathbf{b}_i) \end{aligned}$$

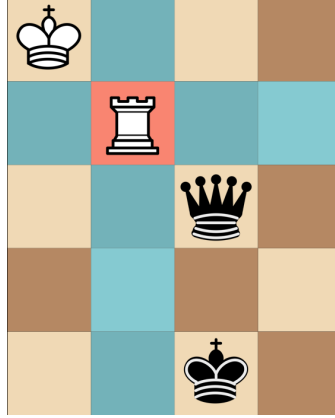


Figure A.1: A position where White’s rook (highlighted in red) is pinned to its own king. The moves highlighted in blue are not legal, even though they correspond to the rook’s standard pattern of movement.

C Collapsing a n -layered feed-forward, fully connected neural network with linear layers into a single layer

Let the i -th layer with weights w_i and bias b_i of a n -layered neural network be denoted by the mapping

$$F_i(x) = L(\mathbf{w}_i \cdot x + \mathbf{b}_i)$$

where $L(\cdot)$ is a linear function on the form $L(x) = a \cdot x + c$ and let the entire model be written as a chain of such layers, so that

$$F(x) = F_0 \circ F_1 \circ \dots \circ F_n(x)$$

where \circ is the standard function-composition operator.

First, it is to be shown that a linear activation function for a layer L_i can be written as a single operation, incorporating a and c into \mathbf{w}_i and \mathbf{b}_i . In other words, if $\mathbf{w}_i \cdot x + \mathbf{b}_i$ is linear, and L is linear, then $L(\mathbf{w}_i \cdot x + \mathbf{b}_i)$ must be linear.

$$\begin{aligned} F_i(x) &= L(\mathbf{w}_i \cdot x + \mathbf{b}_i) \\ F_i(x) &= a \cdot (\mathbf{w}_i \cdot x + \mathbf{b}_i) + c \\ F_i(x) &= a\mathbf{w}_i \cdot x + a\mathbf{b}_i + c \\ F_i(x) &= \mathbf{w}_i^* \cdot x + a\mathbf{b}_i + c & | \quad a\mathbf{w}_i = \mathbf{w}_i^* \\ F_i(x) &= \mathbf{w}_i^* \cdot x + \mathbf{b}_i^* & | \quad a\mathbf{b}_i + c = \mathbf{b}_i^* \end{aligned}$$

Then, one can continue by collapsing the first and second layer. Let $F_i^*(x)$ denote the composition

$$F_i^*(x) = F_i \circ F_{i+1} \circ \dots \circ F_n(x)$$

This means that one can write

$$F(x) = F_0(F_1(F_2^*(x)))$$

Expanding this yields:

$$\begin{aligned} F(x) &= F_0(F_1(F_2^*(x))) \\ F(x) &= \mathbf{w}_0^* \cdot F_1(F_2^*(x)) + \mathbf{b}_0^* \\ F(x) &= \mathbf{w}_0^* \cdot (\mathbf{w}_1^* \cdot F_2^*(x) + \mathbf{b}_1^*) + \mathbf{b}_0^* \\ F(x) &= \mathbf{w}_0^* \mathbf{w}_1^* \cdot F_2^*(x) + \mathbf{w}_0^* \mathbf{b}_1^* + \mathbf{b}_0^* \\ F(x) &= (\mathbf{w}_0^* \mathbf{w}_1^*) \cdot F_2^*(x) + (\mathbf{w}_0^* \mathbf{b}_1^* + \mathbf{b}_0^*) \end{aligned}$$

Since all \mathbf{w}_i and \mathbf{b}_i are trainable constants wrt. x , this means that this process can be continued until the entire model is represented as a single linear function. This is also indirectly shown from all F_i being linear.

D GradCAM applied to positions used for validating the proposed heatmap method

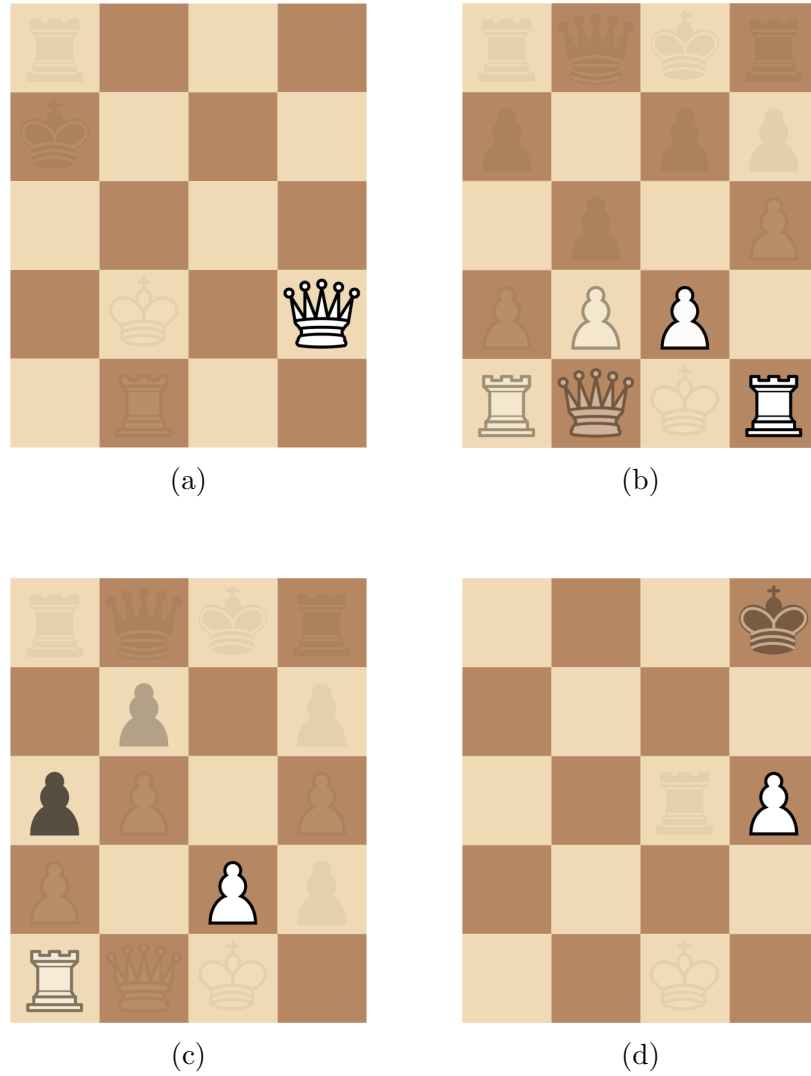
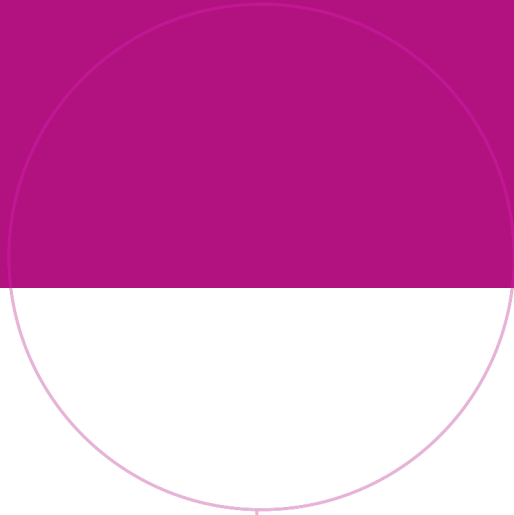


Figure D.1: Positions with an applied saliency map generated by applying the GradCAM method described in Sec. 2.6.2. The opacity of each piece shows the learned importance for each piece for the given position.



Norwegian University of
Science and Technology