Time Management in Chess with Neural Networks and Human Data

Tillmann Rheude

August 27, 2021

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department
Artificial Intelligence and Machine Learning Lab

Abstract

Time management in two-player board games such as chess describes the efficient use of the available playing time to generate the best possible moves without losing the game due to lack of time.
The present work addresses the limited application of machine learning and deep learning methods for this research topic. Previous time managers are based on heuristics and rarely use more complex methods. Thus, it is investigated how computer chess engines can benefit from the human gaming behaviour in terms of the allocation of time for each move. For this purpose, three neural networks are presented that learn the human behaviour of time allocation and reproduce it in chess games. One of these neural networks emerges as the most suitable for this problem. A generic implementation of this neural network into the game process of two chess engines is developed which can be applied to any UCI-ready chess engine and to any time control. The neural network thus acts as a time manager without applying well-known heuristics and without being built into the specific search process of a chess engine.
The developed neural networks which are designed to handle the art of time management are called Chess Time Management Network (*CTM Net*) 1.0, 2.0 and 2.1. Although no significant and positive ELO difference for *CTM Net* was found, the present work is the first of its kind and represents an important groundwork in the development of modern time managers. *CTM Net* is tested with state of the art chess engines such as *Stockfish* to generate comparability and classification of the results in today's process of chess AI.

Keywords: Deep Learning, Chess, Stockfish, Neural Networks, Cute Chess

Contents

1. Introduction

The efficient use of available time in board games such as chess is an important resource beside the resource of material used in the game like the game pieces that are available for any player (Šolak et al. 2009). There is no universally valid definition of the term time management as „several factors [...] [like] remaining thinking time, expected number of moves until the time expires or until the end of the game, attributes of the analyzed position [...] and an opponent's remaining thinking time and strength“ (Šolak et al. 2009) play an important role. According to Markovitch et al. (1996), „even human expert players find it hard to formulate explicitly the reasoning behind their resource allocation decisions“. An efficient „[...] time management can have a very significant effect on playing strength“ (Huang et al. 2010). But in fact, time management „[...] did not receive enough attention“ (Markovitch et al. 1996). Considering this, the efficient management of time in chess influences the outcome of the game and is therefore an important component.

The present work investigates whether chess engines can benefit from human play behaviour or should even fundamentally adapt their own behaviour with regard to the allocation of time to the human behaviour. Therefore, multiple neural networks are developed to study the efficient use of time in the game of chess by using human data.

In the present chapter, the term „time management“ is specified (chapter 1.1), the efficient use of time is explained (chapter 1.1.1) and the context to so-called time managers is drawn (chapter 1.1.2). At the end of this chapter, related work in the area of time management as well a s two-player board games like chess are presented (chapter 1.2).
Chapter 2 describes the programming part of the present work. An MLP (chapter 2.3.1), CNN (chapter 2.3.2) and a residual CNN (chapter 2.3.3), namely *CTM Net*[1] 1.0, 2.0 and 2.1, are presented. CNNs „[...] [have] been widely applied to image data“ (Bishop 2006). Nevertheless, CNNs can also be applied to computer chess. For example Czech et al. (2019) implemented the chess board „[...] as a 8×8 multi-channel image“ such that a CNN can use the board as input.
In chapter 3, the results of the present work on the basis of various experiments and comparisons are presented. Chess engines such as *Stockfish*[2] are used to be able to draw state of the art comparisons.
In chapter 4, a self-critical look at the present work and advantages as well as disadvantages of the methods used are presented. Finally, in chapter 5, findings are summarized and possible future work is presented.

1.1. Time Management

For the term „time management“, no valid definition exists because multiple variables can play a role in order to find the amount of time needed for a ply (Šolak et al. 2009, Hyatt 1984, Markovitch et al. 1996). The allocation of time in general can be described and simplified as a threshold problem. Each player of the board game should search longer in hope to find better moves but each player should stop the search if a good solution has been found. This does not only apply to chess but to the management of time in any field of work.

In the following, several criteria from the literature are presented which make it possible to correctly classify the approach of time management. First, the concept of managing the available time in an efficient way is explained in chapter 1.1.1. Second, the use of a time manager is explained in chapter 1.1.2.

1.1.1. Efficient Use of Time

In the following, the concept of scheduling the available time in an efficient way is explained by referring to Šolak et al. (2009) and Donninger (1994).

Šolak et al. (2009) explain that the available time should be spent depending on two simplified factors:

1. „[...] the expected number of moves until the end of the game [...] “
2. „[...] the remaining thinking time“

Šolak et al. (2009) further state that it is possible to classify each player according to the board positions[3]. There are two possibilities for such positions according to Šolak et al. (2009): Quiet positions are positions in which „[...] only one possibility of recapture [...]“ is available. Stormy positions are positions in which the player is not following the goal of recapturing the game directly but does it indirectly via choosing a move „[...] that will create a greater threat or even sacrifice [...] [of another] piece“.
These four criteria already illustrate that chess is a multifaceted problem in which there is often no clear distinction regarding the

[1]„*CTM Net*“ is an acronym for Chess Time Management (Neural) Network
[2]`https://stockfishchess.org/`, accessed 2021-08-20
[3]A board position can be any arrangement of pieces on the board of a chess game. A simple example of a board position would be the starting position in chess.

management of time. For example, the remaining time can be low and yet the decision to think for a short or long time can remain open depending on a stormy or quiet position.

Donninger (1994) explains similar indicators to use time efficiently. Donninger (1994) declares the „[...] four golden rules for the use of one's time over the board“ which are „[...] generally agreed in the human chess community [...]“. The player should ...

1. ... „[...] not waste time in easy positions with only one obvious move“,
2. ... „use the opponent's thinking time efficiently“,
3. ... „spend considerable time before playing a crucial move“ and
4. ... „try to upset the opponent's thinking time“.

The first rule refers to quiet positions which are classified by Šolak et al. (2009). For the second rule, it is important to understand the concept of pondering. In human chess games, it is common to think about the next move if the opponent is on the move. This process is called pondering. In computer chess games, this option is disabled normally as chess engines would be way too strong if they use the opponent's time to think about the own next move. In the present work, pondering is therefore not used. The third rule refers to stormy positions which are classified by Šolak et al. (2009) in the paragraph before. The fourth rule refers to human gaming behaviour as a computer generally decides differently compared to humans: Normally, it is not possible to put the opponent into a hectic state by gaining a time advantage of one's own as it might be possible in a human chess game because one chess engine searches for moves regarding its own remaining time and not regarding to the (additional) time of the opponent. Nevertheless, a time advantage is also useful in computer chess and should be the goal of a time manager, as long as the decision for moves is not too spontaneous. Spontaneous decisions, i.e. a very short thinking time which led to the decision, are not necessarily the goal of a time manager: Through the depth search of a chess engine, the aim is to find the best possible moves. If the thinking time is too short, this search depth cannot be achieved. Although a computer engine cannot get into a hectic state because the opponent has more time at his disposal, it can think longer for moves due to a time advantage. This longer thinking time can result in more precise moves due to a larger depth in the search process.

The essence of the two literature extracts is that there are ways to pay attention to one's time management and to use the available time efficiently. Even though it is an abstract treatment of the topic, there are frameworks that can lead to efficient time management.

1.1.2. Time Managers

The idea of the present and related work is to make use of a time manager. In the following, the use of a time manager is explained by referring to Šolak et al. (2009) and Markovitch et al. (1996).

A time manager can be directly programmed into the chess engine or can operate as a separate instance. Since the aim of the present work is to create a time manager that is as generic as possible and can be applied to any UCI chess engine, the latter variant was chosen. This has the disadvantage of additional computational time, but also the advantage of comparability with different chess engines and a simpler implementation. Further details about the implementation are described in chapter 2.1. Regardless of the implementation, the task of the time manager is to calculate the exact time needed for a specific move of a player. This time is used by the engine to stop searching for further moves if the proposed time of the time manager is reached. (Šolak et al. 2009) In order to avoid spending too much time calculating how much time should be used for the next move, the time manager should not be too complex. This additional calculation time is deducted from the available playing time and thus serves as a disadvantage. (Fogel et al. 2006) For example, with a time control of 60+0 seconds, the time manager might need a second to calculate how long the chess engine should search for the best move.[4] However, this second has a negative effect on the time in relative terms, since a maximum of 60 moves would be possible if the chess engine needs zero seconds for each move. A minimization of the time required by the time manager should therefore be aimed for.
Šolak et al. (2009) explain that chess engines only use the recommendation of time as a target direction but do „[...] not strictly follow the recommendation [...]“. This deviates from the objective of the present work, as *CTM Net* specifies the exact time.

Markovitch et al. (1996) define a time manager similarly as follows: „A resource allocation strategy is an algorithm that decides how to distribute the resources between the tasks [...] [and which] decides how much resource the program should spend on the calculation of each move“. Markovitch et al. (1996) further classify the way of programming a time manager into three strategies: static, semi-dynamic and dynamic time managers. Static time managers decide before the start of the game how the time(s)[5] should be allocated. Semi-dynamic time managers decide how the time should be allocated „[...] before each move calculation [...]“. Dynamic

[4] This time control refers to 60 seconds for every player in total without an increment for every ply.
[5] Theoretically, a uniform distribution over the time per ply and the plies played would be possible. That's why it is possible to denote „time“ and „times“. In this work, however, the term „time“ is used.

time managers „[...] can communicate with the move calculation process, and update their resource allocation decisions, while that process is being carried out“. The time managers presented in this work, *CTM Net* 1.0 - 2.1, are classified as semi-dynamic as further explained in chapter 2.1.

In summary, a time manager can be seen as an organiser, without whom the chess engines would search infinitely for the best move. Time managers are therefore an important component of every chess engine.

1.2. Related Work

The related work in this area of research can be divided into conventional approaches and approaches that make use of machine learning algorithms. Based on this subdivision, related literature is presented below. The classification into conventional and machine learning approaches is done by the author and and has not been found in any related work after an extensive literature analysis.

Conventional approaches for time managers generally ignore the exact board position. They are mostly classified as static strategies if the classification from Markovitch et al. (1996) in chapter 1.1.2 is regarded. They try to solve the challenge of time allocation with heuristics depending on variables such as the number of the moves (Lai 2015). In the following, conventional approaches are demonstrated.

The best-known and earliest example of a time manager available is proposed by Hyatt (1984) who investigates human behavior. Several grandmaster[6] games were analyzed and yielded to an allocation of time per move depending on the moves already played (figure 1). Hyatt (1984) state this kind of a time manager as a success as it is „[...] like humans [...]“: „[...] [It is] using more time when the game is complicated, and then tapering off as the game simplified“. Even if Hyatt (1984) only uses the variables of moves played and the respective grandmaster thinking time as indicators for complicated situations, the idea of using human data to solve the task of time management is similar to the idea of this work. Nevertheless, the algorithm of Hyatt (1984) differs from the present work due to its use of heuristics instead of machine learning algorithms which are more flexible in terms of decision making.

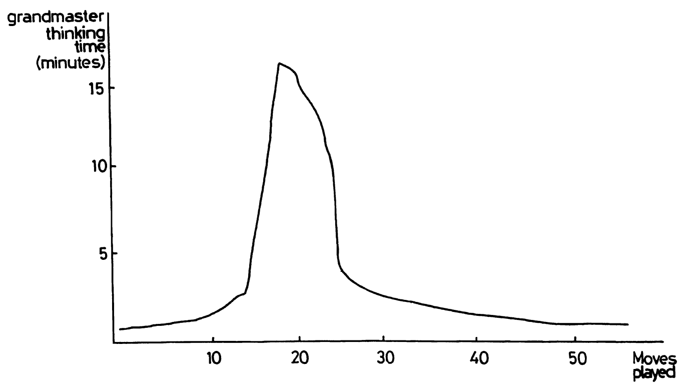


Figure 1: Time management proposed by Hyatt (1984) with the help of human data

More detailed solutions are presented by Šolak et al. (2009) by introducing six different models M1 - M6. The model's complexity is increasing with the number indicating the model. The easiest model, M1, and the most complex model, M6, are presented in the following to give an overview of the range of possibilities to calculate time. In model M1, the time τ per ply[7] is calculated by dividing the available time T through the average number of moves in the game N which has been empirically determined through the evaluation of games in the past (equation 1).

$$\tau = \frac{T}{N} \tag{1}$$

Šolak et al. (2009) explain that this way of using time has a significant disadvantage. If the game is shorter than the calculated average number of games N, a certain amount of time will be left over and thus not used. If the game is longer than the number of games N, there will be no time left for the additional moves and the game will be lost due to lack of time. In model M6, more variables such as „[...] the total material of both players [...] [and] the total coefficient of safety for both kings [...]“ (Šolak et al. 2009) are taken into account. These variables are chosen because of the higher amount of thinking time „[...] if there is more material [...]“ (Šolak et al. 2009). The use of multiple variables is similar to the present work but the work differs as Šolak et al.

[6]A grandmaster is a professional chess player.
[7]A ply is a synonym for a turn and a turn is a move of one player. A move in the game therefore consists of two plies: One of each player.

(2009) are not using complex machine learning algorithms such as neural networks.

Similar heuristics are presented by Huang et al. (2010). The most simple model calculates the time τ needed for a move by dividing the remaining time T through a constant variable C (equation 2).

$$\tau = \frac{T}{C} \tag{2}$$

The value of the constant variable is derived by several experiments. This model is very similar to model M1 proposed by Šolak et al. (2009) (equation 1). The most complex function of Huang et al. (2010) also takes plies into account. Nevertheless, the authors refer to the game of Go and not to chess. Although time allocation is also a relevant topic in Go, it already differs from the present work in terms of the variables used. Further, the authors make use of pondering for their results and are not using machine learning algorithms for improvements. The work of Huang et al. (2010) therefore differs from the present work.

In summary, conventional approaches rely on the use of heuristics and static functions instead of dynamic and flexible algorithms.

In contrast, Machine learning approaches can work in multiple ways. Generally, these approaches learn by using data from the past. For example, time managers can use timestamps relative to board positions from games in the past to predict a time for every ply depending on the exact board position. This is the approach which the present work is working with. But time managers can also be implemented in the engine directly and predict time depending on various other variables such as the depth of the searching tree. These approaches differ from the present work. Machine learning approaches would be classified as semi-dynamic or dynamic strategies regarding to Markovitch et al. (1996)'s classification as their prediction depends on specific game-related variables and cannot be determined in advance. In the following, the approach of Kocsis et al. (2001) and the Monte Carlo Tree Search (MCTS) approach proposed by Baier et al. (2016) are referred in order to be able to draw comparisons with the present work.

Kocsis et al. (2001) handle the problem of time allocation „[...] from the set of search decisions“ by examining „[...] how temporal difference learning and genetic algorithms can be used to improve [...] decisions made during game-tree search“. Even if Kocsis et al. (2001) use neural networks, their approach differs from the present work due to its training data and the way of implementing the neural network into the engine's search process. The present work uses human data from online chess games. Further, the time manager of the present work is implemented as a separate instance and not directly into the engine's search process.
Baier et al. (2016) use MCTS „[...] for fine-grained time control [...]“ in the game of Go. The present work does not use MCTS to handle the allocation of time. It therefore differs from the approach of Baier et al. (2016).

In summary, the present work therefore varies from the related work which is listed above in terms of the following criteria:

- *CTM Net* is a neural network and therefore does not use any pre-defined static functions or heuristics.
- *CTM Net* uses human data in the game of chess for training and predicting the thinking time needed for a ply.
- *CTM Net* is not implemented into the search process of the engine directly. It only limits the time which the engine is able to search for the best move. Therefore, *CTM Net* does not use MCTS or similar search algorithms.
- Most importantly, *CTM Net* is theoretically able to boost any chess engine as the time manager is implemented as a separate instance. This is a conclusion from the previous bullet point.

Further, the present work has an another impact on two-player board games in general: The underlying idea of *CTM Net* could be applied to any two-player board game if a database (chapter 2.2) is available.

Lai (2015) stated that „[...] all other chess engines today use simple heuristics to decide how much time to spend per move. It may be beneficial to use a neural network to make these decisions instead.“. Nevertheless, it was written as future work for the engine *Giraffe* (Lai 2015) and did not gain any attention since then. Even though Lai (2015) already published this idea in 2015, no publications were found even after an extensive literature search.

2. Chess Time Management Network

In the following subsections, the implementation details and network architectures are elaborated. *CTM Net* is written in Python[8] and PyTorch[9]. The code for *CTM Net* is available at Github[10]. For the exact versions of the frameworks used, it is also referred to the Github repository.

Chapter 2.1 describes the overall layout of *CTM Net* regarding to its implementation in the game process. Chapter 2.2 explains the choice of training variables for *CTM Net*. Chapter 2.3 explains the different architectures of *CTM Net*, namely *CTM Net* 1.0, *CTM Net* 2.0 and *CTM Net* 2.1 in the subsections 2.3.1 - 2.3.3.

2.1. Overview and Integration

In the following, the overall integration of *CTM Net* into the game process is explained to understand the advantages and possibilities the network can reveal.

Figure 2 shows the overall layout of two chess engines playing against each other by using *Cute Chess*[11]. *Cute Chess* sends information to the engines by using the UCI protocol. Every engine therefore gets information about all the last moves of every engine and the remaining time of every engine as described below figure 2. Further, it is possible to set opening suites, specific time controls, the amount of engines and various other hyperparameters that are of interest for chess games with *Cute Chess*. *Cute Chess* can be seen as the organizer of the chess game while the engines are searching for moves and send them back to *Cute Chess* by using the UCI protocol too.

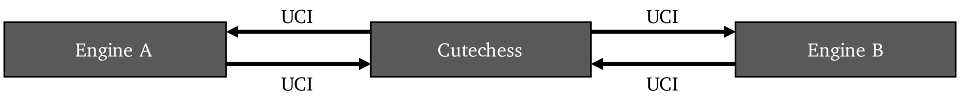


Figure 2: Process of two (chess) engines playing against each other. The communication with *Cute Chess* is done by using the UCI protocol. (Source: own representation)

The present work fools *Cute Chess* as it lets *Cute Chess* think it communicates to two engines (figure 3). In reality, however, *Cute Chess* is only directly communicating to engine B while engine A from figure 2 gets the information from *Cute Chess* by a devious route. To be able to draw direct comparisons for the use of *CTM Net*, engine A equals engine B. Therefore, the same two engines are playing against each other.

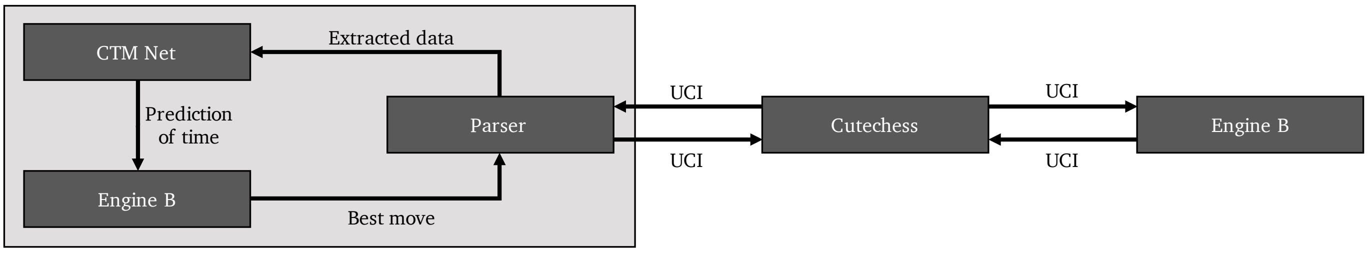


Figure 3: Process of two (chess) engines playing against each other. Engine A from figure 2 is now also engine B. The same engines are playing against each other therefore. One of the engines (left part of the representation) is boosted with *CTM Net* which gets the extracted data from the parser. The network predicts the amount of time the engine should search for and engine B calculates the best move which is sent back to the parser afterwards again. The communication with *Cute Chess* is done by using the UCI protocol. (Source: own representation)

The crucial part is that one of these engines is boosted by the prediction of time due to *CTM Net*. The information which *Cute Chess* sends along the devious route, is processed in a parser. The parser extracts all the data which is used by *CTM Net* to predict a specific amount of time. The specific implementation of the parser of *CTM Net* is inspired by *Sunfish*[12] which is a chess engine written in Python. In general, this extracted data can be adjusted for multiple use cases but it should be noted that *Cute Chess* only sends information about the time left and the last moves. Further details about the amount of information and the information which the

[8] `https://www.python.org/`, accessed 2021-08-20
[9] `https://pytorch.org/`, accessed 2021-08-20
[10] `https://github.com/TillmannRheude/Time-Management-in-Chess-with-Neural-Networks-and-Human-Data`, accessed 2021-08-25
[11] `https://CuteChess.com/`, accessed 2021-08-20
[12] `https://github.com/thomasahle/sunfish`, accessed 2021-08-22

parser extracts is described in chapter 2.2. The extracted data from the parser are compressed and are used as input for *CTM Net*. *CTM Net* processes the input and predicts an amount of time which engine B will need to think about the specific board position. Engine B therefore searches for the best move depending on the prediction of time of *CTM Net*. The best move is forwarded to the parser again which sends it back to *Cute Chess* by using the UCI protocol . Due to this integration of *CTM Net*, *CTM Net* could be applied to any engine as engine B could be swapped with another engine. It should be noted that engine B no longer uses its built-in time manager when using *CTM Net*, but instead it is replaced by *CTM Net*.
For communicating with the engines, the following important UCI messages are sent by *Cute Chess* or the engine respectively:

```
(1) Cute Chess: position startpos moves e2e4 c7c6
(2) Cute Chess: isready
(3) Engine:     readyok
(4) Cute Chess: go wtime 59000 btime 59000
```

The first message (1) is sent by *Cute Chess* and contains the last moves of both engines. The second message (2) is also sent by *Cute Chess* and asks the engine if it is ready to search for the amount of time specified in the final message written above (4). The engine has therefore as much time as it needs to process the last moves until the time on the clock is decreasing as the engine starts its search process. This property is important for the implementation of *CTM Net*. Almost all calculations are done before the engine answers with the third message written above (3). By doing this, the calculation time for the time manager can be reduced. Only one calculation has to be done after the answer (3) of the engine („*readyok*“): Since the remaining time is submitted (4) to the engine after its response (3), the time that the engine should use for the search can only be calculated at this point. This is explained in more detail in chapter 2.2 with equation 3. The whole reduction of calculation time is important as described in chapter 1.1.2 by referring to Fogel et al. (2006).

2.2. Training Data

The training data for *CTM Net* are human chess games. Even if humans automatically activate pondering, the way humans think about board positions may be more efficient. As a board position quickly reveals to be quiet or stormy (chapter 1.1.1), a human player decides fast if it is useful to think longer about the best move. During the development of *CTM Net*, it became clear that it should be avoided to use computer generated chess games as training data. The possibility exists that *CTM Net* only learns the way to allocate time which is programmed into the engines which are used for the computer generated chess games. By using computer generated chess games from multiple engines, this could end in a chaotic way of predicting time. By using computer generated chess games from only one kind of engine, this could end in a reproduction of the engine's time management. Both ways are misleading to derive a solution which solves the problem of allocating the „correct“ amount of time needed for a specific ply.

The idea is to use the human decision-making process in order to create better computer decisions in terms of allocating time. According to Bishop (2006), this idea can be classified as supervised learning as the human decisions are assumed to be the ground truth. In fact, it is not possible to declare human decisions in tasks related to time management as a ground truth. There is no way to determine that the time needed for thinking about one specific board position is definitely the correct one. The art of time allocation has to be seen in context and is different for every human player. As one player may declare a board position as an easy board position, another one could declare it as a board position which is not obvious in regarding to the best move. Nevertheless, the assumption is that human players of a professional ELO[13] level (e.g. grandmasters) are thinking similarly about their time management in different board positions. This problem is also stated by multiple authors in the different context of reinforcement learning[14] and is called credit assignment problem (Fürnkranz 1996; Sutton et al. 2018). According to Fürnkranz (1996), „[...] it is hard to identify the moves that have contributed the most to this outcome“. In context of time management, it would be therefore hard to identify the timestamps that have contributed the most to the win of the game.

The data used in the present work is provided by Lichess[15] as it is one of the few databases which also implements timestamps about every move of every game. The games are further filtered by using pgn-extract[16]. This filtering is important to ensure the quality of the training data as the training data comes from online chess. The games had to fulfill the following filter criteria to be chosen as training data:

- The game has to end with checkmate, i.e. one of the two players has won the game by checkmating the opponent and not, for example, by one of the players leaving the game. Since the goal is to win the game, games that ended in a draw are not considered for training.

[13]ELO is used as a measure of the playing strength of players. In human chess, a player above an ELO of 2500 is classified as grandmaster.
[14]Reinforcement learning is one type of machine learning beside unsupervised and supervised learning. In reinforcement learning, „[...] the learning algorithm is not given examples of optimal outputs, in contrast to supervised learning, but must discover them by a process of trial and error.“ (Bishop 2006)
[15]`https://lichess.org/`, accessed 2021-08-20
[16]`https://www.cs.kent.ac.uk/people/staff/djb/pgn-extract/`, accessed 2021-08-20

- The game has to be played by players with an ELO of at least 2000 in order to extract the most valuable timestamps.
- The game has to be played in the specific time control (for example 60+0 seconds) in order to avoid to be out of distribution.

In addition to this filtering, another criteria beside using pgn-extract is applied: As the data from Lichess is generated by humans playing chess online, the validity of the data has to be confirmed even further. Therefore games are removed from the dataset if a player needed more than 15% of the total time for a ply (e.g. ten seconds for a ply in a time control of 60+0 seconds). The percentage value of 15% was determined through various tests and evaluations of the allocations of time in existing chess engines such as *Stockfish* 13 which are described in chapter 3.3.

In the following, the used training data for the neural networks is presented. *CTM Net* is trained by the following features:

1. The current chess board
2. The total number of pieces of the player who won the game depending on the current chess board
3. The time a player used to think about the best move depending on the current chess board in relation to the remaining time of the player

The third point in the enumeration above refers to the output (prediction) feature of the neural networks. The underlying idea behind this feature is to learn a percentage of the remaining time rather than a specific time. The engine should not lose the game due to lack of time towards the end of the game. For example, the neural network may have learned a time for a board position that came about when there was still a lot of time left. In another game, however, this board position could come about when there is time pressure. Thus, it is only important for the engine to know whether it has to react quickly or slowly in relation to the remaining time. An example from the dataset would be the time of 1.3 seconds to think about the next ply for a specific board position. In the game of the dataset, these 1.3 seconds were possible as the player still has ten seconds left on the clock. Nevertheless, these 1.3 seconds can lead to the loss of the game if there is only one second left on the clock. That's why the engine should calculate the time τ needed for this move by using the prediction ρ of the neural network and the remaining time t_r as following:

$$\tau = \rho \cdot t_r = \frac{1.3s}{10s} \cdot 1s = 0.13s \tag{3}$$

In this case, the engine therefore thinks for 0.13 seconds to calculate the best move. Nevertheless, the time for thinking can be higher or lower, depending on how much time the player has left.

The first and second point in the enumeration above are the input features. These features were chosen because of specific reasons which are explained in the following and can be categorised in three categories. In total, these input features result in a vector with 769 entries which is explained in more detail in chapter 2.3.1.

As *CTM Net* is based on the idea of a Markov model, the first category of features are features which fulfill the Markov property. The Markov Property states that a state in the future depends only on the current state, but not on states before the current state. Mathematically, for a state s_t with time indicator t the state s_2 for example only depends on s_1 but not on s_0. (Sutton et al. 2018) This is the reason why features like NAGs[17], number of moves in the past, chess boards in the past, etc. are not implemented for the training process. The variable „result" is an exception. It is passed on for training indirectly. It would also be reasonable to select only won games of one player (e.g. the white player) when filtering the pgn-files via pgn-extract. However, the underlying idea is to use the data efficiently. That's why the game board is mirrored in relation to the player who won the game. The information is extracted by the winning player and not due to its color. The application of the Markov assumption in chess can be criticised in certain game situations. For example, it is possible that player A has lost his queen in an early move. Player A could therefore play strategically in such a way that he captures player B's queen as quickly as possible. For the time decisions that arise from the loss of player A's queen, past states are therefore of great importance.
The second category of features are features which fulfill the Markov property but are not implemented in the training process of *CTM Net*. For example, it is possible to include the remaining time of the opponent player. However, it should be noted that one's own chess engine should be designed as independently as possible from the opponent's chess engine as the desired improvement of the time manager for the own engine should work against as many other engines as possible and not only against one specific engine. For example, by implementing the opponent's remaining time, match situations could occur that are only associated with this remaining time in the time manager's training because they have not occurred more often with other remaining times. This would affect the generalization ability of the time manager.
The third and last category of features are features which fulfill the Markov property and which are included in the training process but could be improved. For example, one of these improved features would be the exact specification of the individual pieces for the current chess board instead of the total number of all pieces: *CTM Net* can be adjusted to take the amount of kings (1), queens (2),

[17]Acronym which stands for numeric annotation glyphs. In computer chess games, NAGs indicate additional information for specific moves of any player. For example, a NAG can indicate that a chosen move was a bad choice, a so-called blunder.

pawns (8), etc. as features in a board positions instead of a more simple, single value (the sum of all pieces on the board).

Although *Cute Chess* only sends a limited amount of information (section 2.1), a lot of additional information can already be derived from this information. In summary, the choice of features with which *CTM Net* is trained is not limited, but should also not be made without further considerations.

Feature scaling is applied to the training data to guarantee a better training process. Specifically, z-score normalization is used to normalize the input and output data to zero mean and unit variance. The normalization is also applied to the validation dataset and is used later to de-normalize the prediction in the game process. In case of *CTM Net* 2.0 and 2.1, no normalization is applied to features which are already in a range between zero and one as they are already standardized.
The networks are trained by using Adam[18] (Kingma et al. 2015) as optimizer and the minimum squared error (MSE) as loss criterion. The choice of Adam was not arbitrary: Different optimizers such as Stochastic Gradient Descent (SGD) in combination with different learning rate schedulers were tested. However, Adam turned out to be consistently the best option. The same applies to the choice of the loss function: in addition to the MSE loss function, other loss functions (L1, SmoothL1 (Huber), Log Cosh and XSigmoid) were also tested. MSE turned out to be the best choice.
The criterion for the evaluation of the training process is the MSE value and the R2-score. As there is no comparable literature on time management, no metrics have been established so far. Accordingly, the R2-score was introduced as an additional metric to better interpret the training results. Furthermore, after each training epoch, the network is validated on a validation dataset. The validation dataset remains the same for every training process (also between different versions of *CTM Net*) to ensure comparability. Even though it is unusual in the field of machine learning, there is more than enough data available in the field of time management to train a neural network. The data from Lichess is divided into years and months. The games from the year 2020 and the months January to April were selected as training data. Despite the filtering criteria described previously, 13.791.081 plies remained to train the architectures (10.375.348 plies for training and 3.415.733 plies for validating).

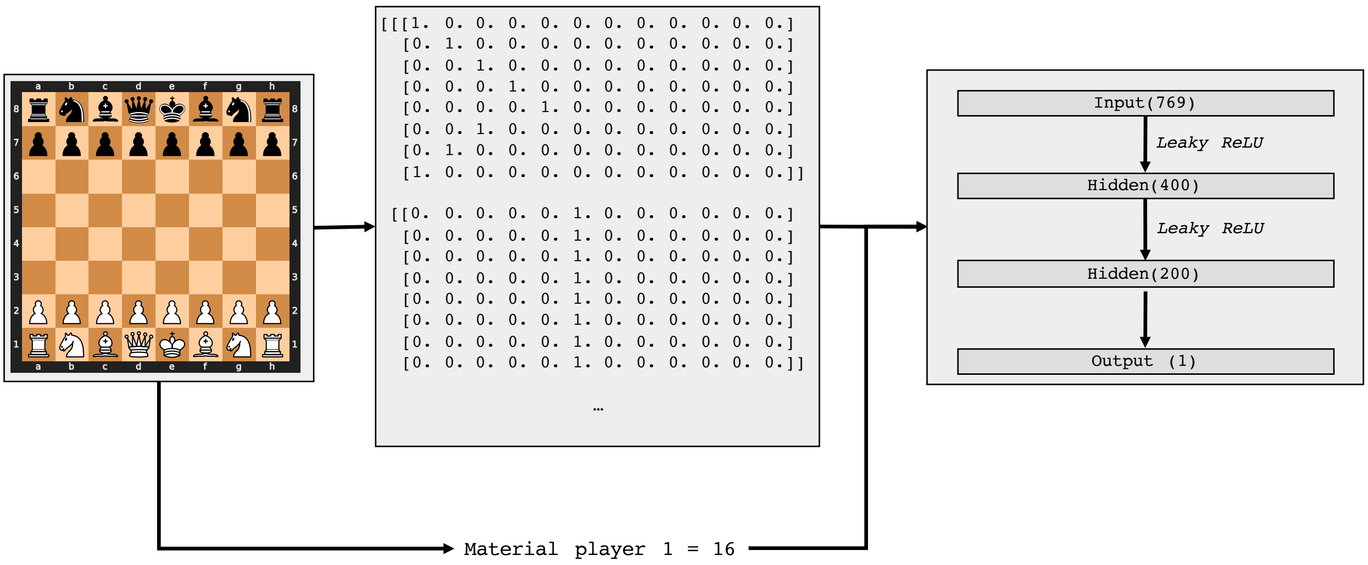
[18]Acronym which stands for Adaptive Moment Estimation.

2.3. Architectures

In the following, *CTM Net* and its variants are presented. As every architecture has certain characteristics, every architecture is explained in detail in the chapters 2.3.1 - 2.3.3. After their explanations, an evaluation is attached in order to be able to draw conclusions, understand the development process and understand the development decisions..

2.3.1. CTM Net 1.0

CTM Net 1.0 can be described as a feedforward neural network, specifically as a Multi-Layer Perceptron (MLP). The underlying idea of an MLP, the perceptron, was originally proposed by Rosenblatt (1958). A schematic visualization of *CTM Net* 1.0 is shown in figure 4. The input layer of *CTM Net* 1.0 consists of 769 neurons which equals the amount of input data. The chess board is converted to a vector of 768 values (six different pieces, 8 horizontal and 8 vertical positions and two players result in a vector of 768 values). The amount of material for the player who won the game is added additionally (1 value). By summing up the values, a total vector of 769 values for every chess board in a game is used as input for the training of *CTM Net* 1.0. This is further described in chapter 2.2. The output layer consists of one neuron to predict a single value - the time in relation to the remaining time (1 value). The two hidden layers between the input and the output layers are chosen to contain 400 and 200 neurons respectively. The number of neurons was chosen based on several performance tests regarding the loss values. The choice of 400 and 200 neurons turned out to be the most consistent one. For example, less neurons led to higher loss values and more neurons led to overfitting[19]. The same applies to the number of layers.



```
[[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

 [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]]

                    …
```

Figure 4: Extraction process of the input features and the structure of *CTM Net* 1.0. First, the specific chess board is given as input. Starting from this, the pieces of player 1 are counted and the chessboard is formatted into an array notation. These two features are transformed as a 1D vector and thus, after a subsequent normalisation, serve as input for the following neural network. The first layer (769 neurons) represents the input vector which is described in chapter 2.2. The last layer (one neuron) represents the output vector with one entry (the predicted time). The layers in between, the so-called hidden layers, connect the input and output layers. The input features presented in the middle of the representation are not normalized yet to demonstrate that player one still owns all the figures. (Source: own representation)

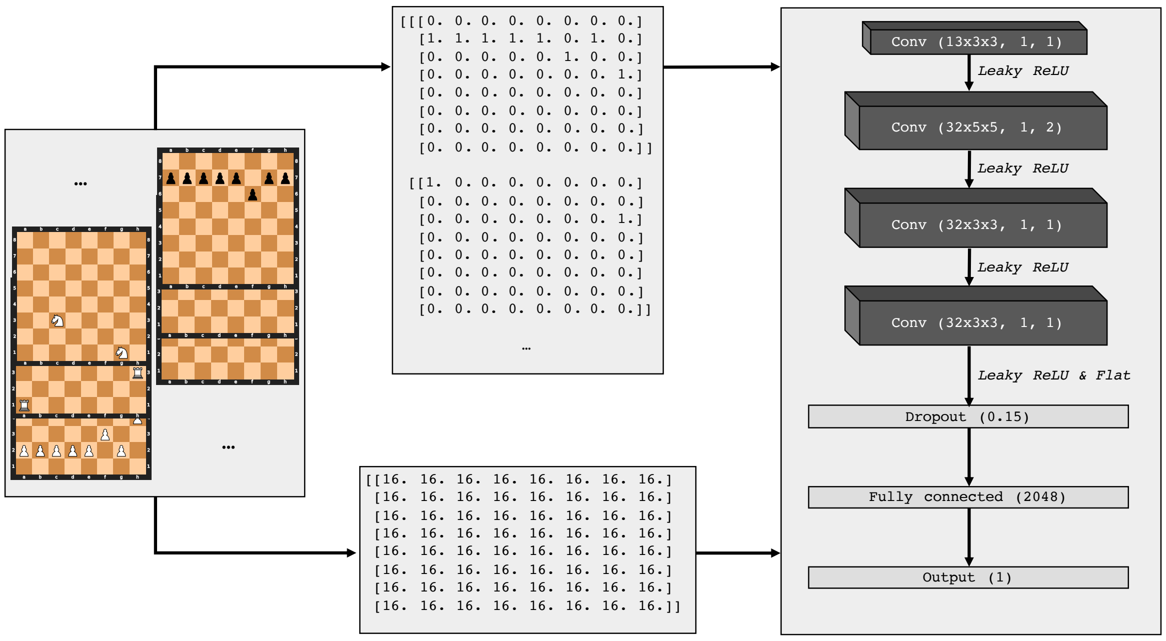
[19]Overfitting describes the problem that a machine learning algorithm learns the training data very well, but performs very poorly on new data (the validation dataset). In the case of overfitting, the architecture has learned by rote instead of learning rules. (Murphy 2021)

2.3.2. CTM Net 2.0

To improve *CTM Net*, a second version of it, *CTM Net* 2.0, is built which can be classified as convolutional neural network (CNN) (LeCun et al. 1989). The idea of CNNs is used for *CTM Net* 2.0 for reaching the same goal as *CTM Net* 1.0: Predict the amount of time needed for a ply. Figure 5 shows the architecture of *CTM Net* 2.0.

For *CTM Net* 2.0, another choice for designing the input of variables had to be made compared to *CTM Net* 1.0. As a CNN uses bitmaps as inputs, the training data changed in the following way: For every figure on the board a separate bitmap is designed. Further, the material of the player who won the game is converted to an additional bitmap. This bitmap has the same dimensions as the other bitmaps and is filled with one scalar value: the material of the player. These two design decisions are illustrated in figure 5. Finally, the normalized bitmaps are the input for *CTM Net* 2.0.

Since even a small change of a piece on a chessboard can have a big effect, the convolution layers are chosen to be 32x5x5 (output size, filter size) with stride 1 and padding 2 or 32x3x3 with stride and padding 1. The number of convolutional layers proved to be the best as fewer convolution layers led to worse results and more convolutional layers led to overfitting. The same applies for the filter size of 5x5 and 3x3 respectively and the output sizes of the convolutional layers. Pooling layers[20] (e.g. max pooling) are not used in order not to lose the exact information about the pieces. If CNNs are applied to images, this design decision is unusual.
Leaky ReLU is used as the activation function, as it showed slightly better results than ReLU. Other activation functions (Tanh, Sigmoid) could not outperform Leaky ReLU.
The number of fully connected layers after the convolution layer is not arbitrary. First, multiple fully connected layers with 2048, 1024 and 512 neurons were used to make the network more flexible. However, this led to overfitting, which is why the number of fully connected layers was reduced to one. After the last fully connected layer, no activation function (i.e. a linear one) is used as the network should predict a continuous value, the time in seconds.
To reduce overfitting even more, a dropout[21] layer with 15% was inserted before the fully connected layer. The value of 15% represented the best trade-off between generalisation and loss reduction. The learning rate is set to 3e-4 and the batchsize to 2048. Larger learning rates led to oscillating loss values and lower learning rates led to a longer training duration until convergence.



```
[[[0. 0. 0. 0. 0. 0. 0. 0.]
  [1. 1. 1. 1. 1. 0. 1. 0.]
  [0. 0. 0. 0. 0. 1. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 1.]
  [0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0.]]

 [[1. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 1.]
  [0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0.]]

            ...
```

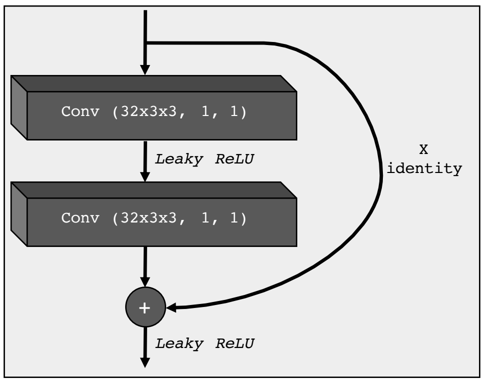
```
[[16. 16. 16. 16. 16. 16. 16. 16.]
 [16. 16. 16. 16. 16. 16. 16. 16.]
 [16. 16. 16. 16. 16. 16. 16. 16.]
 [16. 16. 16. 16. 16. 16. 16. 16.]
 [16. 16. 16. 16. 16. 16. 16. 16.]
 [16. 16. 16. 16. 16. 16. 16. 16.]
 [16. 16. 16. 16. 16. 16. 16. 16.]
 [16. 16. 16. 16. 16. 16. 16. 16.]]
```

Figure 5: Representation of *CTM Net* 2.0 and its inputs features. The array filled with the value of 16 is not normalized yet. The arrays filled with zeros and ones are not normalized as they are already standardized. The extraction process of the variables is similar to figure 4. The block on the right demonstrates the different layers of *CTM Net* 2.1. (Source: own representation)

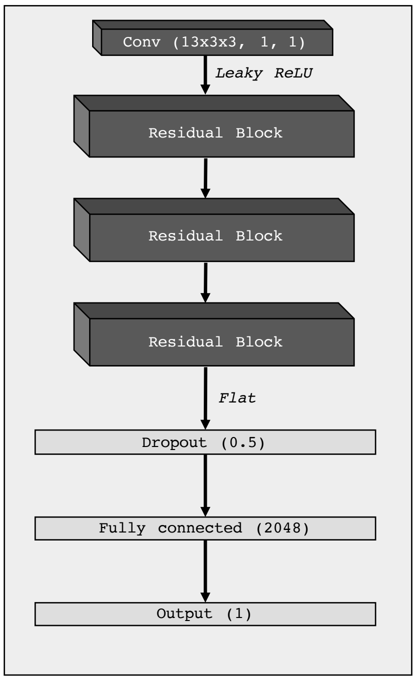
[20]Pooling layers are able to make a neural network „invariant to the location“ (Murphy 2021) which can be useful to find out if something is somewhere in the image but it is not that important where exactly the object in the image is located.
[21]Dropout layers „turn off all the outgoing connections from each neuron with probability p“ (Murphy 2021) which in this case are 15%.

2.3.3. CTM Net 2.1

To further improve *CTM Net* 2.0, the use of residual blocks (He et al. 2015) is possible. Figure 6 (a) shows the architecture of one residual block adapted to *CTM Net*. As „[...] gradients can flow directly from the output to earlier layers [...]“ (Murphy 2021), neural networks with residual layers are generally „[...] easier to train [...]“ (Murphy 2021). Figure 6 (b) shows the resulting *CTM Net* 2.1 with three residual block layers. The number of three blocks proved to be the most stable and best choice in experiments. More blocks increased the risk of overfitting while fewer blocks led to lower training success. The number of the input features remain the same as in *CTM Net* 2.0. The same applies to the first convolution layer, the dropout layer, the final fully connected layer and the activation functions. For dropout, a probability of 0.5 proved to be better in this case.



(a)



(b)

Figure 6: Figure (a): One residual block in *CTM Net* 2.1 which was originally proposed by He et al. (2015). The use of batch normalization was intentionally not implemented into the residual blocks of *CTM Net* 2.1 as this led unexpectedly to worse results with regard to the generalization of the network. Figure (b): Schematic representation of *CTM Net* 2.1 based on the idea of residual blocks (He et al. 2015). The input is the same as the input for *CTM Net* 2.0 (chapter 2.3.2, figure 5). (Source: own representation)

3. Results

In the following, various results are shown on the basis of different characteristics. First, the training results of the different architectures are evaluated in section 3.1 and its subsections 3.1.1 - 3.1.3. Then, the Eigenmann Rapid Engine Test (Eigenmann 2017) is applied to check the predictions of *CTM Net* 2.1 in section 3.2. Finally, *CTM Net* 2.1 is applied in different game processes to evaluate its success in section 3.3.

3.1. Training Process

The following subsections show the training progress of the individual architectures which are explained in sections 2.3.1 - 2.3.3. Therefore, section 3.1.1 - 3.1.3 evaluate the training of *CTM Net* 1.0, 2.0 and 2.1.

3.1.1. CTM Net 1.0

CTM Net 1.0 shows poor training performance. The training process is illustrated in figure 7 (a) and (b). It is clearly visible that the training is not stable and overfitting occurs. By evaluating figure 7 (a), a good training process until epoch eleven is obtained. Nevertheless, figure 7 (b) shows that the R2-score of the validation set is decreasing and that the R2-score is almost always below zero. Different numbers of neurons and hidden layers were tested, such as 20 and 10 neurons in the hidden layers or only one hidden layer with 10-100 neurons. However, this did not improve the results for *CTM Net* 1.0. A dropout layer after the last linear layer did not improve the overfitting. Especially the R2-scores are disappointing as the best R2-score which can be scored in general is one. This poor performance in the training process led to the development of *CTM Net* 2.0.
It should be noted that the validation loss for *CTM Net* 1.0 (and the following architectures) is below the training loss. This is uncommon but can be explained by the different sizes of the data sets. As the validation set contains approx. only 30% of the chess games compared to the training set, the validation set contains simpler predictions.

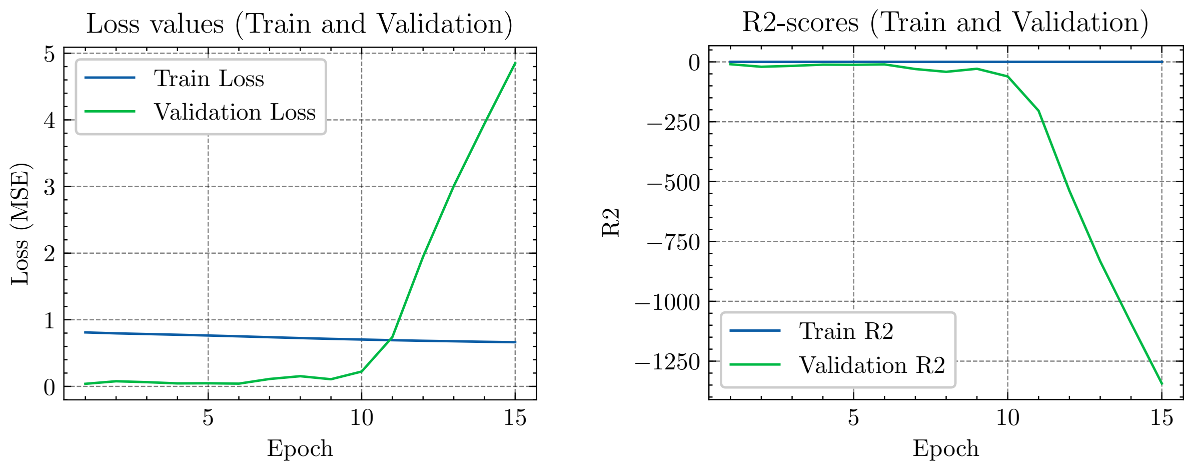


Figure 7: Training process of *CTM Net* 1.0. Figure (a): Loss values for the train and validation dataset with a learning rate of 3e-4 and a batchsize of 2048. Figure (b): R2-scores for the train and validation dataset with the same learning rate and batchsize.

3.1.2. CTM Net 2.0

The resulting *CTM Net* 2.0 shows better generalization and faster training in comparison to *CTM Net* 1.0. The training process is illustrated in figure 8 (a) and (b). It is clearly visible that the training is more stable and there is no extreme overfitting in the first ten training epochs. However, small oscillations in the validation loss and validation R2-score are visible. The drop in the loss around epoch nine comes from a step learning rate scheduler with $\gamma = 0.1$ and a step size of ten. For *CTM Net* 1.0 (chapter 2.3.1), no learning rate scheduler was used because the loss values were not promising enough. As the training loss is above the validation loss until epoch eleven for *CTM Net* 2.0, the perfect training would end at this epoch. Nevertheless, in figure 8 (b), the R2-score for the validation set is already below the R2-score for the training set. This is used as indicator for overfitting in this case. That's why *CTM Net* 2.0 should be trained until epoch ten to ensure to not suffer from overfitting.

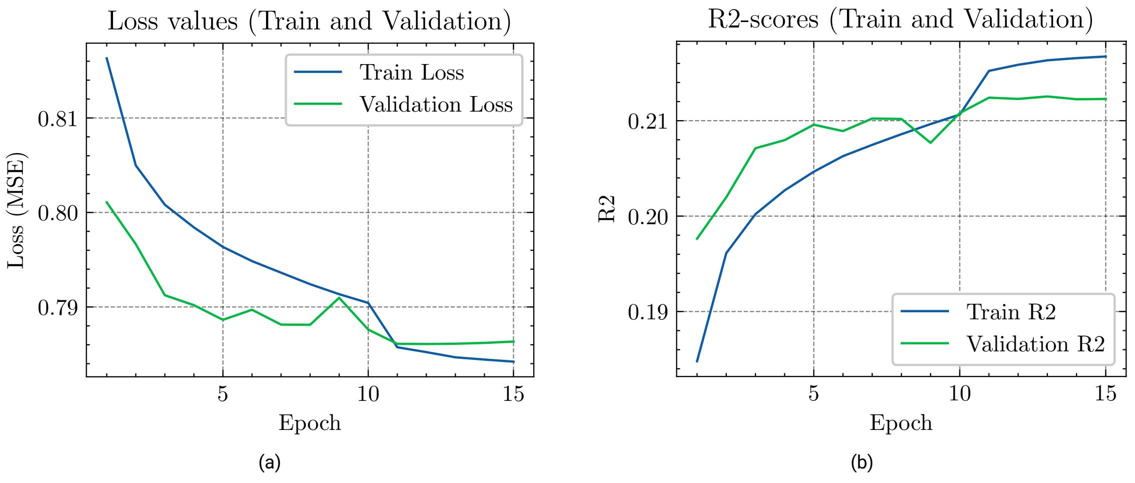


Figure 8: Figure (a): Loss values of *CTM Net* 2.0. Figure (b): R2-scores of *CTM Net* 2.0. The learning rate is set to 3e-4 and the batchsize to 2048.

3.1.3. CTM Net 2.1

The resulting *CTM Net* 2.1 shows slightly better generalization and faster training in comparison to *CTM Net* 2.0. The training process is illustrated in figure 9 (a) and (b). *CTM Net* 2.1 reaches its best state at epoch ten without overfitting regarding the loss values or the R2-scores. The loss value is even slightly lower than in *CTM Net* 2.0 and the R2-score slightly higher than in *CTM Net* 2.0.
To complete the model, the number of training epochs was increased. For this, a step learning rate scheduler was implemented. Multiple experiments with different step sizes and γ parameter values were done: γ value of 0.01 and step size of ten, γ value of 0.01 and a step size of five and a γ value of 0.1 and a step size of five. All these experiments did not improve the learning process. *CTM Net* 2.1 overfits after a number of ten epochs which leads to the conclusion that a learning rate scheduler can not improve *CTM Net* 2.1. The training in this case should therefore be stopped when the loss on the validation data set becomes higher than on the training data set (after nine epochs). This concept is also called „early stopping“ (Murphy 2021).

The optimal model is therefore *CTM Net* 2.1 with an early stopping after ten epochs. As this architecture proved to be the best, it was tested to increase the amount training data even more but this did not lead to a better training process. Further it was tested to use the material difference between the two players as input feature instead of the amount of material of player one. This also did not lead to a better training process.

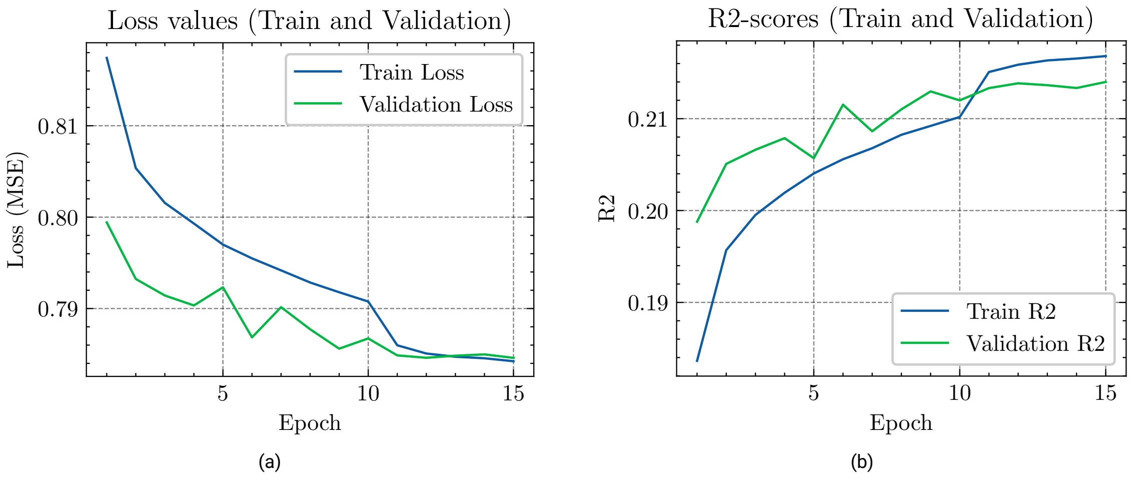


Figure 9: Evaluation of *CTM Net* 2.1. Figure (a): Loss values for the train and validation dataset with a learning rate of 3e-4 and a batchsize of 2048. Figure (b): R2-scores for the train and validation dataset with the same learning rate and batchsize.

3.2. Eigenmann Rapid Engine Test

To evaluate the performance and the learning results of *CTM Net* 2.1, specific board positions are evaluated by using the Eigenmann Rapid Engine Test (ERET) (Eigenmann 2017). The board positions from this test are classified as difficult. The test delivers different board positions and the best move which is possible in this position. The following table 1 shows the different predictions of *CTM Net* 2.1 for the proposed board positions of the ERET. The predictions are rounded to four decimal points. To ensure a better understanding of the game situation, the first FEN[22] position of the following table is shown on a chessboard in figure 10.

FEN	**Prediction**
r2r2k1/1p1n1pp1/4pnp1/8/PpBRqP2/1Q2B1P1/1P5P/R5K1 b	0.0484
2rq1rk1/pb1n1ppN/4p3/1pb5/3P1Pn1/P1N5/1PQ1B1PP/R1B2RK1 b	0.0423
5rk1/pp1b4/4pqp1/2Ppb2p/1P2p3/4Q2P/P3BPP1/1R3R1K b	0.0493
r1bqk1r1/1p1p1n2/p1n2pN1/2p1b2Q/2P1Pp2/1PN5/PB4PP/R4RK1 w q	0.0446
r1n2N1k/2n2K1p/3pp3/5Pp1/b5R1/8/1PPP4/8 w	0.0811
r1b1r1k1/1pqn1pbp/p2pp1p1/P7/1n1NPP1Q/2NBBR2/1PP3PP/R6K w	0.0537
5b2/p2k1p2/P3pP1p/n2pP1p1/1p1P2P1/1P1KBN2/7P/8 w	0.0568
r3kbnr/1b3ppp/pqn5/1pp1P3/3p4/1BN2N2/PP2QPPP/R1BR2K1 w kq	0.0421
r2qk2r/ppp1bppp/2n5/3p1b2/3P1Bn1/1QN1P3/PP3P1P/R3KBNR w KQkq	0.0312
rnb1kb1r/p4p2/1qp1pn2/1p2N2p/2p1P1p1/2N3B1/PPQ1BPPP/3RK2R w Kkq	0.0475
r1b2r1k/ppp2ppp/8/4p3/2BPQ3/P3P1K1/1B3PPP/n3q1NR w	0.0541

Table 1: Extract of board positions from ERET and the predictions of *CTM Net* 2.1.

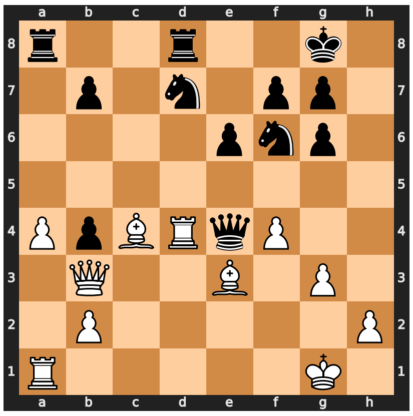


Figure 10: Chess board of the first FEN entry of table 1. It is the black player's turn. This board comes from the ERET and is therefore classified as difficult.

In order to be able to compare these predictions with the mean predictions of *CTM Net* 2.1, the amount of pieces on the board has to be counted. The black player owns twelve figures in the position which is illustrated in figure 10. In 100 games against *Stockfish* 13 (section 3.3), the mean value of twelve figures on the board is reached in ply number 21. The mean remaining time in ply number 21 is approx. 37 seconds. For the first prediction listed in table 1, the time to think about this board position would therefore be calculated as following:

$$\tau = 0.0484 \cdot 37s = 1.7908s \tag{4}$$

The engine should therefore search for 1.7908 seconds in case of this specific board position (figure 10). In comparison with the mean values of *CTM Net* 2.1 (figure 11), these 1.7908 seconds are almost one of the highest prediction times regarding the mean values. In the case of the fifth prediction from table 1, the time would even increase to approx. three seconds. This would approx. correspond to one of the extreme values from figure 12 (b). *CTM Net* 2.1 therefore predicts a longer thinking time when the board positions are more difficult (ERET). However, the ERET board positions that are used should lead to extreme values instead of mean values. This is not always achieved as a result of the first example above, but as the second example above shows, it is partially achieved.

[22]Acronym which stands for Forsyth-Edwards-Notation. This notation allows to represent every board position in the game of chess.

3.3. Game process

Since *CTM Net* 2.1 turned out to be the best choice for the available training data when predicting time in the context of time management in chess, it is used in the following for testing the model with two engines in the game process.
Syzygy tablebases[23] and opening suites[24] with an advantage of 150-159 pawns for the white player were used to be able to draw faster and more significant results and to avoid many draws which have no clear significance in showing which engine is stronger. Every opening suite was played two times to ensure that every engine is one time the white and one time the black player. To refer to Fogel et al. (2006) (chapter 1.1.2), the computational time of the prediction of *CTM Net* 2.1 is calculated: approx. 0.02 seconds. The time control for the following tests is 60+0 seconds.

First, *Stockfish* 13[25] without any configurations (beside the default configurations as e.g. setting threads to one) is used to play chess games. As described in chapter 2.1, one *Stockfish* 13-engine (engine A) is boosted with *CTM Net* 2.1. After 100 games, the statistics (Win for *CTM Net* 2.1 - Win for SF13 - Draw) of *CTM Net* 2.1 vs. *Stockfish* 13 are as following: 32 - 44 - 24. This results in an ELO difference of -41.9 $\pm$ 60.2. *CTM Net* 2.1 in combination with *Stockfish* 13 was therefore not able to win against *Stockfish* 13.

In order to be able to draw conclusions about the time management of the two engines in the context of the present work, the games were evaluated afterwards. This is described in the following.

For every ply, the mean time value over all games was calculated (figure 11). The first plies clearly represent values of zero seconds, as these plies were played by the opening suites. In the first ply after these plies by the opening suite, a significant difference in the mean value of both engines is obtained. For the mean value, *Stockfish* 13 thinks approx. three seconds while *CTM Net* 2.1 only thinks approx. one second. While the times of *Stockfish* 13 after this ply starts to oscillate and have a logarithm-like shape, the times of *CTM Net* 2.1 are smoother. Further, the time management of *CTM Net* 2.1 is similar to the obtained time management of Hyatt (1984) which is described in chapter 1.2 and which is visualized in figure 1. *CTM Net* 2.1 allocates more time between the second and the approx. 30th ply while *Stockfish* 13 allocates more time than *CTM Net* 2.1 after the approx. 30th ply.

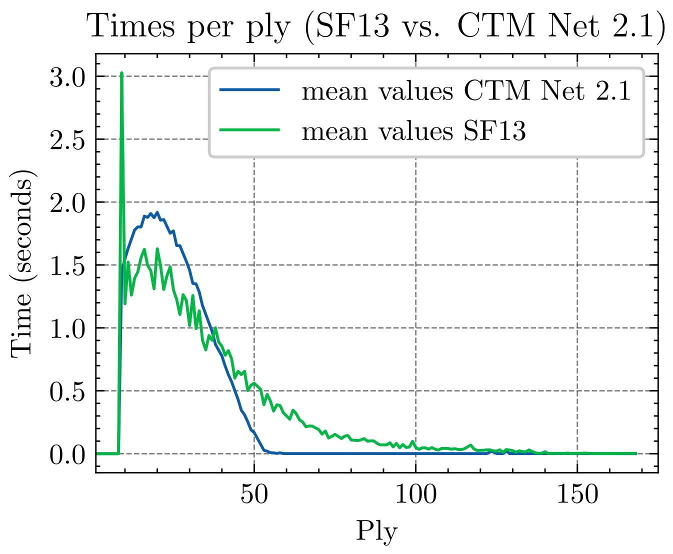
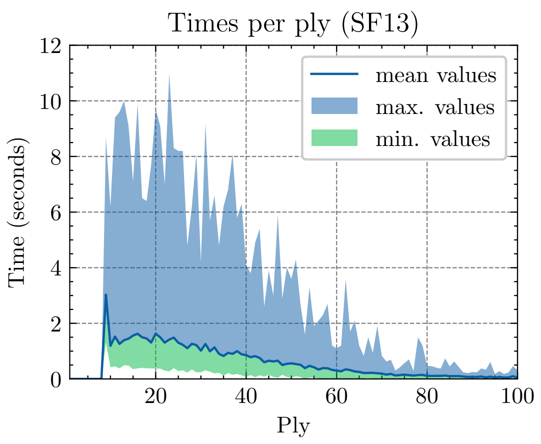


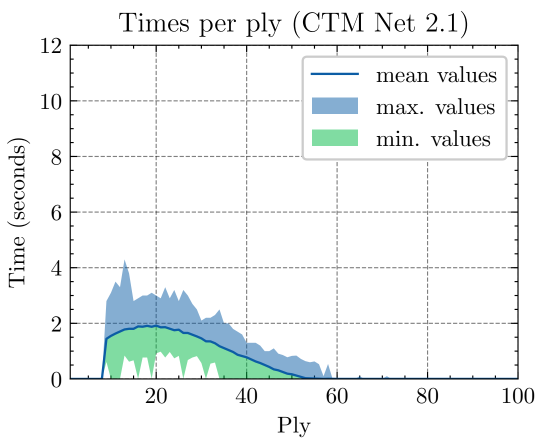
Figure 11: Mean values of the time needed for every move number in 100 games of *Stockfish* 13 vs. *CTM Net* 2.1.

Further, the minimum and maximum values between the two engines are compared as visualized in figure 12 (a) and (b). In general, *CTM Net* 2.1 does not show extreme peaks in its time management. *CTM Net* 2.1 reaches a maximum thinking time of approx. 4.1 seconds. *Stockfish* 13 reaches a maximum thinking time of approx. 10.1 seconds. *Stockfish* 13 therefore uses extreme peaks for thinking in certain plies. Nevertheless, these peaks are rare as the mean values of *Stockfish* 13 are between zero and three seconds similar to *CTM Net* 2.1.
Figure 12 (c) - (e) show the time management of the two engines relative to their playing color. It is noticeable that *Stockfish* 13 tends to think longer regarding the peaks mentioned before when playing as the black player which is visible after the 50th move.

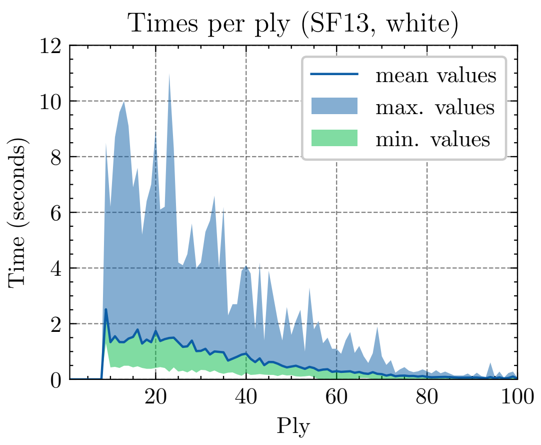
[23]`https://syzygy-tables.info/`, accessed 2021-08-20
[24]`https://www.sp-cc.de/anti-draw-openings.htm`, accessed 2021-08-20
[25]`https://stockfishchess.org/blog/2021/stockfish-13/`, accessed 2021-08-26



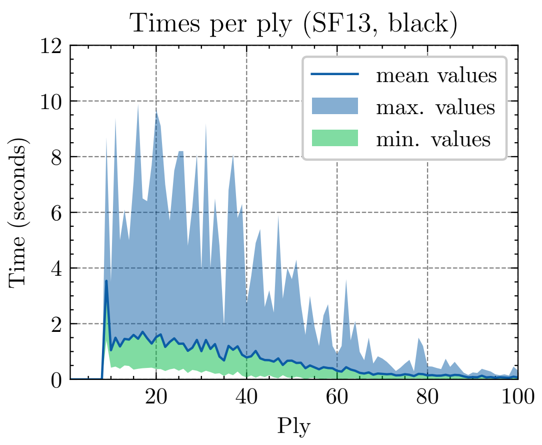
(a)



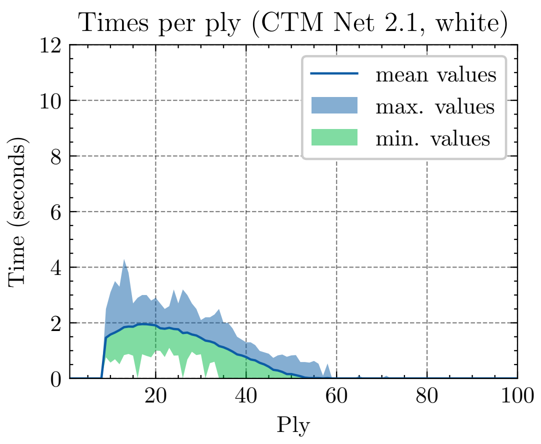
(b)



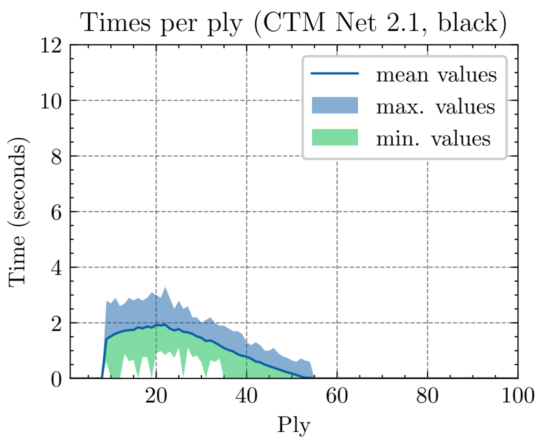
(c)



(d)



(e)



(f)

Figure 12: Further evaluations of mean, minimum and maximum values between the two engines. Figure (a) and (b) show the statistics of the two engines independent of their playing color. Figure (c) - (e) show the statistics of the two engines dependent on their playing color.

As *CTM Net* 2.1 was trained to predict the time needed for a ply relative to the remaining time, it should be able to handle all time controls. In table 2, the results of 100 games with different time controls than 60+0 seconds are presented. In summary for this evaluation, the application of *CTM Net* 2.1 also failed to significantly win for different time controls against *Stockfish* 13.

Time control	Win *CTM Net* 2.1	Win *Stockfish* 13	Draw	ELO difference
10+0 [sec]	23	52	25	-103.7 ±61.4
60+1 [sec]	28	46	26	-63.2 ±59.8
120+0 [sec]	32	45	23	-45.4 ±60.7

Table 2: Results of *CTM Net* 2.1 vs. *Stockfish* 13 with different time controls than 60+0 seconds in 100 games. Every opening was played two times to guarantee that each engine is playing the white player one time.

Beside *Stockfish* as chess engine, *CTM Net* 2.1 was also tested on other chess engines. The engines tested further are *Ethereal*[26] 12.75, *Igel*[27] 2.6.0, *Slow Chess Blitz*[28] 2.6, *Xiphos*[29] 0.6.1 and *Rubi Chess* 1.9[30]. During the development of *CTM Net* 2.1, *Stockfish* 14[31] was released. Therefore, *CTM Net* 2.1 was also tested in combination with *Stockfish* 14. *CTM Net* 2.1 was not able to gain an advantage against these engines. For this evaluation, a time control of 60+0 seconds was used again. The specific results are listed in table 4 (appendix A).

It is remarkable that *CTM Net* 2.1 thinks very briefly from the 55th move onwards (figure 12 (b)). From this number of moves onwards, the opponent's engine (e.g. *Stockfish* 13) still uses a few seconds for the individual moves (figure 12 (a)). This led to the idea to start *CTM Net* 2.1 later. The amount of time to think about a board position is thus done by *Stockfish* 13 for the first moves. After a certain number of moves, *CTM Net* 2.1 handles the time management. After the results described above, the aim is to investigate whether chess engines can benefit from human behaviour instead of changing their whole time management to the human behaviour. Table 3 shows the results of this experiment. Different start values (measured by move numbers after the opening suite) are evaluated. The first entry in table 3 comes from the idea of using only the extreme peak in the mean of *Stockfish*'s first move (figure 11) as this long thinking time could be crucial. Even if the ELO differences are not as large as before, the use of *CTM Net* 2.1 is still not beneficial for *Stockfish* 13.

Start value	Win *CTM Net* 2.1	Win *Stockfish* 13	Draw	ELO difference
1	37	47	16	-34.9 ± 63.3
15	33	46	21	-45.4 ± 61.5
25	35	39	26	-13.9 ± 59.1
35	32	45	23	-45.4 ± 60.7

Table 3: Results of *CTM Net* 2.1 vs. *Stockfish* 13 with a delayed start of *CTM Net* 2.1. Until the start value, *Stockfish* 13 handles the time management. From the starting value, *CTM Net* 2.1 handles the time management. For every entry of the table, 100 games with two rounds and a time control of 60+0 seconds were played.

In contrast, it was also investigated how the results behave when *CTM Net* 2.1 first takes over the time management and, from a certain starting value, *Stockfish* 13 handles the time management again. The specific results are listed in table 5 (appendix A). The ELO difference is decreased again but no advantage for *CTM Net* 2.1 is shown.
Due to the fact that the mid game of chess games is the most important part, another evaluation is done: *CTM Net* 2.1 handles only the time management for the mid game (measured by a starting value and an end value). The start and end values tested were 5-15, 15-25, 25-35 and 35-45. Nevertheless, this attempt also failed to be an advantage for *CTM Net* 2.1. The specific results are listed in table 6 (appendix A).

In order to compare the performance of *CTM Net* 2.1 with certain baselines, *Stockfish* 13 is used again in the following evaluations. Here, the time management of *Stockfish* 13 is first set to a fixed amount of one second until only five seconds are left. From then on, *Stockfish* 13 plays with its own time management to avoid losses due to lack of time. Furthermore, the time management of *Stockfish* 13 is set to a fixed percentage of the remaining time. This is set to five percent. The specific results are shown in table 7 (appendix A). At least for the first case, *CTM Net* 2.1 was able to reach a draw. The second case did not show an advantage for *CTM Net* 2.1.

[26] `https://github.com/AndyGrant/Ethereal`, accessed 2021-08-20
[27] `https://github.com/vshcherbyna/igel`, accessed 2021-08-20
[28] `https://www.3dkingdoms.com/chess/slow.htm`, accessed 2021-08-20
[29] `https://github.com/milostatarevic/xiphos`, accessed 2021-08-20
[30] `https://github.com/Matthies/RubiChess`, accessed 2021-08-20
[31] `https://stockfishchess.org/blog/2021/stockfish-14/`, accessed 2021-08-20

4. Discussion

It was investigated whether neural networks are able to reproduce the human behaviour in allocating time in chess games. This was successfully confirmed by a positive training process and by an evaluation using mean values compared to Hyatt (1984) and the Eigenmann Rapid Engine Test (Eigenmann 2017). The time allocation of neural networks trained on a human data basis is similar to the time management proposed by Hyatt (1984).

Creating a generic model for multiple time controls and multiple engines has been neither successful nor unsuccessful. Since losses on the part of *CTM Net* 2.1 can be observed in every time control, the generic approach has failed and did not gain a benefit for any engine. Nevertheless, the present work could show that the approach is generic and can be applied to any time control and to every UCI-ready chess engine.

Further, the present work investigated to what extent computer chess engines can profit from this human behaviour by using neural networks. A fundamental change in the used time control with the help of neural networks is not recommended on the basis of the present results. The use of *CTM Net* to specific sections of a game has also not been successful. One assumption is that the too short allocation of time after the 50th move and the increased time in the moves before have a negative influence on the result. Another assumption could be the exact usage of the time proposed by the time manager which is not suggested by the literature as described in section 1.1.2. By using this exact prediction of time, the depth in the tree search of a chess engine will be canceled without further evaluations if a small of additional time should be used for this search. Further, it is possible that the assumptions for the use of the training data described in section 2.2 were too restrictive. For example, the percentage value for removing training data whose time in a ply exceeds 15% of the total time could be set higher or lower. It would be possible to determine the ideal percentage value over many experiments. In addition, the use of input features is currently very limited as only the player material is used additionally to the board positions. Thus, extensions with regard to further input features are possible. It is also possible that the time of 0.02 seconds required by the time manager is still too high. Another reason for the results could be the architecture of CTM Net 2.1. It may be beneficial to improve the design of the neural network for gaining better results. Additionally, an intensive annotation of chess games would be advantageous, as this would eliminate some assumptions in the preprocessing step. This requires the human chess knowledge of multiple grandmasters and the annotation step would take a lot of time. Nevertheless, this would possibly further improve the training of *CTM Net*. Another larger improvement could be to replace the Markovian idea and predict time due to multiple states before. In this case, time dependent architectures like LSTMs[32] (Hochreiter et al. 1997) could be beneficial.

5. Conclusion

The present work shows that neural networks can learn the human game behaviour in chess. Furthermore, it shows that neural networks for the task of allocating time in chess should nevertheless follow a certain complexity in order to process the data successfully. This was shown by developing a simpler architecture (section 2.3.1) and more advanced ones (section 2.3.2 and 2.3.1).

Since no literature has been published on a separate time manager trained with the use of human data yet, the present work represents an important finding. However, an advantage due to human game behaviour with regard to time could not be shown.

Future work could include a neural network which follows the idea of visual transformers (Dosovitskiy et al. 2021) to improve the learning process even more.

Acknowledgements

The author would like to thank Prof. Dr. Kristian Kersting for the opportunity to conduct scientific research on Deep Learning methods on an Nvidia DGX2 server proposed by the Artificial Intelligence and Machine Learning Lab of the TU Darmstadt. The author would also like to thank the developers of the chess engines used, which are available free of charge. Big thanks are directed to Thibault Duplessis and the lichess.org team, whose comprehensive database made it possible to investigate this scientific topic in the first place. This also applies to the developers of pgn-extract (David J. Barnes) and python-chess (Niklas Fiekas), whose programming work greatly facilitates the introduction to computer chess. Finally, the author thanks Johannes Czech for the supervision of the present work and various helpful suggestions, without which the work would not have been possible.

[32]Acronym which stands for Long Short-Term Memory

References

Baier, Hendrik and Mark H. M. Winands (2016). “Time Management for Monte Carlo Tree Search”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 8.3, pp. 301–314. DOI: 10.1109/TCIAIG.2015.2443123.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag. ISBN: 0-387-31073-8.

Czech, Johannes et al. (2019). “Learning to play the Chess Variant Crazyhouse above World Champion Level with Deep Neural Networks and Human Data”. In: *CoRR* abs/1908.06660. URL: http://arxiv.org/abs/1908.06660.

Donninger, Christian (1994). “A la Recherche du Temps Perdu: “That was easy””. In: *J. Int. Comput. Games Assoc.* 17.1, pp. 31–35. DOI: 10.3233/ICG-1994-17106.

Dosovitskiy, Alexey et al. (2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv: 2010.11929 [cs.CV].

Eigenmann, Walter (2017). “Computerschach: Testaufgaben für Programme (ERET)”. In: *Glarean Magazin*. URL: https://glarean-magazin.ch/2017/03/05/computerschach-testaufgaben-engines-eigenmann-rapid-engine-test-eret/.

Fogel, David B. et al. (2006). “The Blondie25 Chess Program Competes Against Fritz 8.0 and a Human Chess Master”. In: *2006 IEEE Symposium on Computational Intelligence and Games*, pp. 230–235. DOI: 10.1109/CIG.2006.311706.

Fürnkranz, Johannes (1996). “Machine Learning in Computer Chess: The Next Generation”. In: *J. Int. Comput. Games Assoc.* 19, pp. 147–161.

He, Kaiming et al. (2015). “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385. URL: http://arxiv.org/abs/1512.03385.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long Short-Term Memory”. In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

Huang, Shih Chieh, Remi Coulom, and Shun Shii Lin (2010). “Time management for Monte-Carlo tree search applied to the game of Go”. In: *Proceedings - International Conference on Technologies and Applications of Artificial Intelligence, TAAI 2010*. Proceedings - International Conference on Technologies and Applications of Artificial Intelligence, TAAI 2010, pp. 462–466. ISBN: 978-0-7695-4253-9. DOI: 10.1109/TAAI.2010.78.

Hyatt, Robert M. (1984). “Using Time Wisely”. In: *ICCA Journal* 7.1, pp. 4–9. ISSN: 0920-234X.

Kingma, Diederik P. and Jimmy Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980.

Kocsis, Levente, Jos Uiterwijk, and Jaap van den Herik (2001). “Learning Time Allocation Using Neural Networks”. In: *Computers and Games*. Ed. by Tony Marsland and Ian Frank. Springer Berlin Heidelberg, pp. 170–185. ISBN: 978-3-540-45579-0.

Lai, Matthew (2015). “Giraffe: Using Deep Reinforcement Learning to Play Chess”. In: *CoRR* abs/1509.01549. URL: http://arxiv.org/abs/1509.01549.

LeCun, Y. et al. (1989). “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4, pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

Markovitch, Shaul and Yaron Sella (1996). “Learning of resource allocation strategies for game playing”. In: *Computational Intelligence* 12.1, pp. 88–105. DOI: https://doi.org/10.1111/j.1467-8640.1996.tb00254.x.

Murphy, Kevin P. (2021). *Probabilistic Machine Learning: An introduction*. MIT Press.

Rosenblatt, F. (1958). “The perceptron: A probabilistic model for information storage and organization in the brain.” In: *Psychological Review* 65.6, pp. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519.

Šolak, Rade and Vladan Vučković (2009). “Time Management During a Chess Game”. In: *ICGA Journal* 32.4, pp. 206–220. ISSN: 13896911, 24682438. DOI: 10.3233/ICG-2009-32403.

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Second. The MIT Press. ISBN: 978-0-2620-3924-6.

A. Additional tables

Opponent	Win *CTM Net* 2.1	Win Opponent	Draw	ELO difference
Ethereal 12.75	21	44	35	-81.4 ± 56.0
Igel 2.6.0	30	30	40	0.0 ± 53.1
Slow Chess Blitz 2.6	23	37	40	-49.0 ± 53.3
Xiphos 0.6.1	27	40	33	-45.4 ± 56.4
Rubi Chess 1.9	25	36	39	-38.4 ± 53.7
Stockfish 14	31	50	19	-66.8 ± 62.8

Table 4: Results of *CTM Net* 2.1 vs. other chess engines with a time control of 60+0 seconds in 100 games. Every opening was played two times to guarantee that each engine is playing the white player one time.

Start value	Win *CTM Net* 2.1	Win *Stockfish* 13	Draw	ELO difference
15	35	42	23	-24.4 ± 60.4
25	38	45	17	-24.4 ± 62.8
35	36	41	23	-17.4 ± 60.3

Table 5: Results of *CTM Net* 2.1 vs. *Stockfish* 13 with an early stopping of *CTM Net* 2.1. Until the start value, *CTM Net* 2.1 handles the time management. From the starting value, *Stockfish* 13 handles the time management. For every entry of the table, 100 games with two rounds and a time control of 60+0 seconds were played.

Start & end value	Win *CTM Net* 2.1	Win *Stockfish* 13	Draw	ELO difference
5-15	26	49	25	-81.4 ± 60.7
15-25	27	47	26	-70.4 ± 59.9
25-35	22	44	34	-77.7 ± 56.4
35-45	31	46	23	-52.5 ± 60.8

Table 6: Results of *CTM Net* 2.1 vs. *Stockfish* 13 with a static start and end value for *CTM Net* 2.1. Until the start value, *Stockfish* 13 handles the time management. From the starting value, *CTM Net* 2.1 handles the time management until the end value. For every entry of the table, 100 games with two rounds and a time control of 60+0 seconds were played.

Time management	Win *CTM Net* 2.1	Win *Stockfish* 13	Draw	ELO difference
1 [sec]	38	38	24	0.0 ± 59.9
5 [percent]	33	40	27	-24.4 ± 58.7

Table 7: Results of *CTM Net* 2.1 vs. *Stockfish* 13 with a defined time management for *Stockfish* 13. For every entry of the table, 100 games with two rounds and a time control of 60+0 seconds were played.