hochschule mannheim

Entwicklung eines Efficiently Updatable Neural Network (NNUE) zur Evaluation von Schach Positionen

Marvin Karhan

Bachelor-Thesis

zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)

Studiengang Informatik

Fakultät für Informatik

Hochschule Mannheim

31.08.2022

Betreuer

Prof. Jörn Fischer, Hochschule Mannheim

Prof. Thomas Ihme, Hochschule Mannheim

Karhan, Marvin:
Entwicklung eines NNUE zur Evaluation von Schach Positionen / Marvin Karhan. –
Bachelor-Thesis, Mannheim: Hochschule Mannheim, 2022. 20 Seiten.

Karhan, Marvin:
Development of an NNUE for the Evaluation of Chess Positions / Marvin Karhan. –
Bachelor Thesis, Mannheim: University of Applied Sciences Mannheim, 2022. 20 pages.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Mannheim öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Mannheim, 31.08.2022

Marvin Karhan

Abstract

Entwicklung eines NNUE zur Evaluation von Schach Positionen

Jemand musste Josef K. verleumdet haben, denn ohne dass er etwas Böses getan hätte, wurde er eines Morgens verhaftet. Wie ein Hund! sagte er, es war, als sollte die Scham ihn überleben. Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu einem ungeheueren Ungeziefer verwandelt. Und es war ihnen wie eine Bestätigung ihrer neuen Träume und guten Absichten, als am Ziele ihrer Fahrt die Tochter als erste sich erhob und ihren jungen Körper dehnte. Es ist ein eigentümlicher Apparat, sagte der Offizier zu dem Forschungsreisenden und überblickte mit einem gewissermaßen bewundernden Blick den ihm doch wohl bekannten Apparat. Sie hätten noch ins Boot springen können, aber der Reisende hob ein schweres, geknotetes Tau vom Boden, drohte ihnen damit und hielt sie dadurch von dem Sprunge ab. In den letzten Jahrzehnten ist das Interesse an Künstlern sehr zurückgegangen. Aber sie überwanden sich, umdrängten den Käfig und wollten sich gar nicht fortrühren.

Development of an NNUE for the Evaluation of Chess Positions

The European languages are members of the same family. Their separate existence is a myth. For science, music, sport, etc, Europe uses the same vocabulary. The languages only differ in their grammar, their pronunciation and their most common words. Everyone realizes why a new common language would be desirable: one could refuse to pay expensive translators. To achieve this, it would be necessary to have uniform grammar, pronunciation and more common words. If several languages coalesce, the grammar of the resulting language is more simple and regular than that of the individual languages. The new common language will be more simple and regular than the existing European languages. It will be as simple as Occidental; in fact, it will be Occidental. To an English person, it will seem like simplified English, as a skeptical Cambridge friend of mine told me what Occidental is.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Computer Schach ist ein viel betrachtetes Thema. Schon Alan Turing und Claude Shannon hab sich damit befasst [1], [2]. In seinem 1950 verfassten paper beschrieb Shannon [2] Funktion zu Evaluation einer Schachposition, ihm war jedoch auch klar, das es wahrscheinlich niemals eine exakte Evaluation für Schach geben wird. Deshalb liegt es nahe dafür ein neuronales Netz (NN) zu verwenden, denn dessen Aufgabe ist es solche Funktion zu Approximieren. Leider ist es für die Evaluation in einem Schachcomputer wichtig sowohl genau als auch schnell die Position zu bewerten. Je genauer die Stellung bewertet wird, desto stärker spielt das Programm. Je schneller die Bewertung stattfindet, desto weiter kann der Computer voraussehen, was ebenfalls zu einer höheren Spielstärke führt. Herkömmliche NNs Architekturen scheitern jedoch an einer zu lagen Berechnungszeit oder bei sehr kleinen Netzen an einer zu ungenauen Bewertung.

Eine Lösung für die Probleme herkömmlicher NNs, wurde 2018 von Nasu [3] in seinem Japanischem paper vorgestellt. Er erkannte das inkrementelle Aktualisierungen, wie sie bereits in hand-crafted evaluation (HCE) verwendet wurden, in NNs verwendet werden kann. Der Schlüssel dafür ist ein binäres und dünn besetztes Feature Set, basierend auf den Figuren und ihren Positionen. Die Eingabeschicht, auch affine Transformer genannt, muss nicht bei jeder Aktivierung alle Elemente seines Ausgabevektors neu berechnet. Steht q für die aktuelle Stellung, p für die vorherige Stellung und x als der Vektor Feature Sets wird $v^{(q)}$ konkret mit folgender Gleichung berechnet:

$$v^{(q)} = v^{(p)} - \sum_{j \in \left\{k | x_k^{(p)}=1 \wedge x_k^{(q)}=0\right\}} W_1(:,j) + \sum_{j \in \left\{k | x_k^{(p)}=0 \wedge x_k^{(q)}=1\right\}} W_1(:,j) \tag{1.1}$$

Die NNUE Architektur ist darauf ausgelegt schnell auf einer CPU zu laufen. Sie nutzt CPU basierte Optimierungs-Möglichkeiten wie Single Instruction, Multiple Data (SIMD) und die im letzten Absatz genannte inkrementellen Aktualisierungen, um die Geschwindigkeit zu erlangen, die für ihre Nutzung als Evaluationsfunktion zu rechtfertigen.

Nasus [3] hat die NNUE Architektur für die Verwendung in japanischen Schach Variante Shogi entwickelt. Shogi unterscheidet sich in einigen Punkten von herkömmlichem Schach, es hat unter anderem eine andere Spielfeldgröße und erlaubt es geschlagene Figuren wieder einzusetzen. Trotzdem eignet sich Nasus [3] Ansatz für traditionelles Schach, da die Zuggenerierung sowie die Evaluation ähnlich ist. Außerdem gibt es in beiden Varianten einen König, praktisch für die Auswahl eines passenden Feature Sets, wie in Unterabschnitt 2.5.1 genauer erläutert.

Nur zwei Jahre später zeigte eine Portierung des Konzepts starke Verbesserungen in dem Schachcomputer Stockfish, der sich durch NNUE um mehr als 80 elo verbessern konnte [4]. Die größte Verbesserung einer Stockfish-Version jemals. Mit Ausnahme von AlphaZero [5] hatte bis dahin noch kein NN basierter Ansatz erfolge gezeigt.

Ein Schachcomputer besteht aus drei Teilen: Suche, Zuggenerierung (Boardrepräsentation) und Evaluation [6]. Als Basis für diese Arbeit wird ein simpler Schachcomputer, der in dem Modul Künstliche Intelligenz für autonome Systeme (KIS) entwickelt wurde, verwendet. Dieser Schachcomputer verfügt über eine simple Suche und eine HCE [7]. Gegenstand dieser Arbeit ist es die HCE, des 2021 im Modul KIS entwickelten Schachcomputer, durch ein eigens trainiertes NNUE zu ersetzen. Ziel ist es hierbei nicht eine neue NNUE Architektur zu präsentieren. Es wird die Architektur verwendet, die von Nasu [3] vorgestellt wurde und auch in der ersten Version der Stockfish NNUE verwendet wurde. Der Grund dafür ist, dass sie mit minimalem Domänenwissen auskommt und so ein besseres Bild der Kernelemente der NNUE Architektur vermittelt. Außerdem sollte sie, gemessen an dem Erfolg

in Stockfish, ausreichen um die Spielstärke des in KIS entwickelten Schachcomputers zu steigern. Die NNUE Implementierung, soll ein Proof of Konzept sein. Die Erstellung neuer Eingabedaten für NNUEs ist nicht teil dieser Arbeit.

Kapitel 2

Grundlagen

In diesem Kapitel wird das Wissen vermittelt, welches benötigt wird um zu Verstehen, wie NNUEs im Rahmen von Schachcomputern funktionieren. Zuerst wird die Evaluation, wie sie in herkömmlichen Schachcomputern funktioniert erklärt, auch HCE genannt. Weiterhin werden die grundlegenden Bestandteile, die für überwacht lernende Feedforward Neural Networks (FNNs) von Bedeutung sind, eingegangen. Außerdem wird erläutert was SIMD ist und wie diese Vektoroperationen in C/C++ verwendet werden können. Zuletzt wird die grundlegende Funktionsweise von NNUEs vermittelt, die auf den davor gelegten Grundsteinen aufbaut.

2.1 Hand-crafted Evaluation

Es ist wichtig zu wissen wie die HCE eines Schachcomputers funktioniert, da sie nicht nur die Variante die in jedem starken Schachcomputer vor 2017 eingesetzt wurde, sondern auch heute noch in Kombination mit NNUE eingesetzt wird. Bei NNUE Schachcomputern wird sie oft in Kombination mit der NN evaluation genutzt, weil sie besser in extremen Stellungen funktioniert. Besitzt beispielsweise Weiß in einer Position eine Dame mehr, muss nicht die teurere Berechnung des NNUEs durchgeführt werden, um zu entscheiden, dass Weiß im Vorteil ist.

Die HCE einer Schachposition ist eine heuristische Methode der Position einen numerischen Wert zuzuordnen. Vor der Verbreitung von NNs, war HCE die einzige Form der Positions-Evaluation. Gäbe es unendliche Ressourcen könnten aus jeder Position alle mögliche Zugfolgen per Brute-Force bestimmen und der Positionen einer der drei Werte: -1 (Verlust), 0 (remis), 1 (Gewinn) geben. In der Realität ist es

nicht möglich den exakten Wert der Stellung zu kennen, deshalb wird in der HCE versucht anhand von Menschen festgelegten Kriterien einen Wert der Position zuzuordnen. Die so gewonnene Bewertung wird in der Zugsuche verwendet, um den besten Zug, abhängig von den per Hand gewählten Kriterien, zu finden. Die Evaluation wird aus Sicht der Seite, die gerade am Zug ist angegeben. Das ist wichtig für den verwendeten Suchalgorithmus (Alpha-Beta-Suche) [8].

Die HCE eines Schachcomputers ähnelt in einigen Aspekten mehr eine Philosophie als eine Funktion. Schach ist ein Spiel, das es seit über 1000 Jahren gibt. In dieser Zeit haben Menschen regeln überlegt, um besser Schach zu spielen. All diese Regeln in die Evaluationsfunktion zu integrieren, ist nicht ratsam. Es ist ein Abwägen zwischen Wissen und Geschwindigkeit. Je mehr regeln dem Computer geben werden, umso weniger weit kann er Vorausschauen.

Wenn ein Mensch Schach spielen lernt, ist der Wert der Figuren eines der ersten Erkenntnisse. Das ist ebenfalls der wichtigste Faktor für einen Schachcomputer [9]. Die Angabe der Materialwertung wird bei Computern als Centipawn angegeben, um so mehr Spielraum für feingranulare Faktoren zu lassen. Figuren werden ebenfalls anhand ihrer Position bewertet. Dafür gibt es sogenannte Piece Square Tables, die jeder Figur abhängig von ihrer Position einen Wert zuordnen. Beispielsweise ist ein Springer am Rand des Brettes deutlich weniger Wert als einer im Zentrum, auch bekannt als „ein Springer am Rand bringt Kummer und Schand“. Weitere nennenswerte Aspekte der HCE sind

2.2 Neuronale Netze

künstliche neuronale Netze (KNNs) oder einfach NNs genannt, sind Computer Systeme, die dem biologischen Vorbild des Gehirns nachempfunden sind. Analog zu seinem biologischen Vorbild besteht ein NN aus Neuronen die miteinander Vernetzt sind. Jedes Neuron reagiert auf eingehende Signal mit einer bestimmten Reaktion. Diese Reaktion kann sich durch neu gewonnene Erfahrungen anpassen, das ermöglicht es zu lernen und zukünftig besser zu reagieren.

In Abbildung Abbildung 2.1 ist ein einfaches neuronales Netz zu sehen. Es besteht aus drei Schichten. Die erste Schicht, die Eingabeschicht, nimmt Eingabedaten entgegen. Eingabedaten können ganz unterschiedliche daten Repräsentieren, ist der Eingabedatensatz beispielsweise ein 100×100 Schwarzweiß-Bild, bestände die

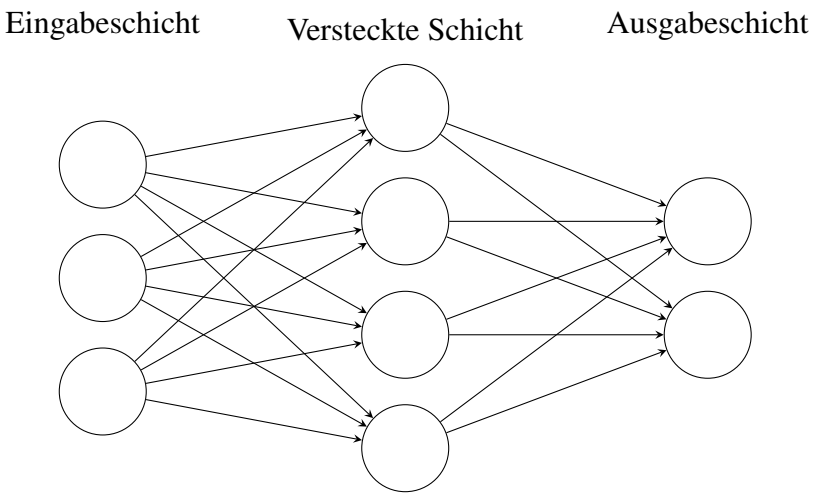


Abbildung 2.1: Ein einfaches neuronales Netz

Eingabeschicht aus 1000 Neuronen die pro Neuron den Zustand eines Pixels (0 = weiß, 1 = Schwarz) des Bildes gefüttert bekommen. Die zweite Schicht heißt versteckte Schicht, weil von außen nur die Eingabedaten und das Ergebnis sichtbar ist. Sie empfängt die Informationen der Eingabeschicht, gewichtet sie und gibt sie an der Ausgabeschicht weiter. Die versteckte Schicht kann aus mehreren Schichten bestehen. Ein NN mit mehreren versteckten Schichten heißt Deep Neural Network (DNN). Die letzte Sicht, die Ausgabeschicht, spiegelt das Ergebnis des NNs dar. Ein Netz, das versucht Bilder zwischen Hunden und Katzen zu unterscheiden kann zwei Ausgabeneuronen enthalten, eins für die Wahrscheinlichkeit das auf dem gegebenen Bild ein Hund ist und eins für die Wahrscheinlichkeit das es eine Katze ist. Ein NN kann auch nur ein Ausgabeneuron besitzen, wie z. B. bei der Evaluation einer Schachposition nötig ist. Die Verbindungen der einzelnen Neuronen stellen deren Zusammenhang dar. Wie stark dieser Abhängigkeit ist, wird durch Gewichte definiert [10, S. 2–7].

Es gibt verschiedene Modelle neuronale Netze, für diese Thesis sind lediglich FNNs relevant. FNNs basieren auf dem von Rosenblatt [11] beschriebenen mehrlagigen Perzeptron. Das FNN zeichnet sich durch seinen Zyklen freien Aufbau aus. Der Datenfluss führt immer von der Eingabeschicht zu Ausgabeschicht. Das FNN gilt als die einfachste Netzwerkarchitektur [12].

In der Praxis, so auch in dieser Arbeit, werden für die Entwicklung neuronaler Netze Frameworks verwendet. Sie abstrahieren große Teile der Komplexität. Trotzdem ist es wichtig ihre Funktionsweise zu kennen, um Entscheidungen zu treffen und Probleme zu beheben. In den folgenden Unterabschnitten wird Grundlegend auf

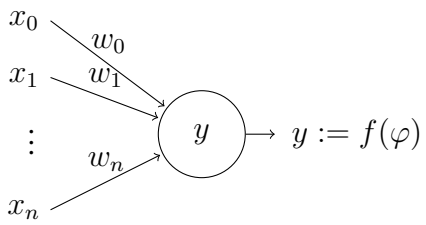


Abbildung 2.2: Ein einzelnes Neuron mit seinen eingabe- und Ausgabekomponenten

die Einzelteile neuronaler Netze eingegangen. Erst wird das Neuron beschrieben und wie sich seine Aktivität berechnen lässt. Das Unterkapitel Backpropagation beschreibt, wie NNs lernen können.

2.2.1 Das Neuron

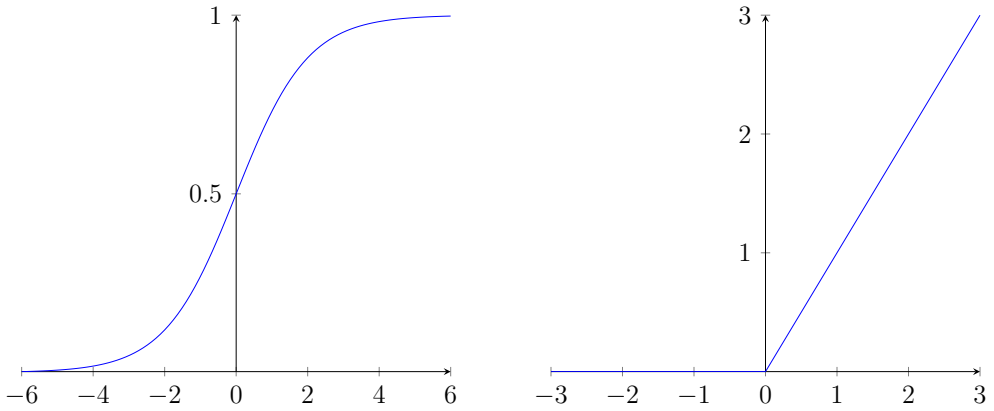
Das Neuron ist der elementare Bestandteil eines NNs. Es wurde 1943 von McCulloch und Pitts [13] eingeführt. Neuronen sind in einem NN mit anderen Neuronen verbunden und bilden so beliebig komplexe Funktionen ab. In Abbildung 2.2 ist ein einzelnes Neuron zu sehen. Die Eingänge x_0 bis x_n, werden mit den Gewichten w_0 bis w_n multipliziert und aufsummiert und mit der Aktivierungsfunktion $f(\varphi)$ wird die Ausgabe des Neurons. Für gewöhnlich ist immer $x_0 = 1$, was ihn zu dem Bias des Neurons mit $w_0 = b$ macht. Das bedeutet, dass es nur n tatsächliche Eingabewerte gibt: von x_1 bis x_n. Konkret lässt sich die Aktivität eines Neurons mit der Gleichung 2.1 und die Ausgabe y mit Gleichung 2.2 bestimmen:

$$f(\varphi) = \varphi(\sum_{i=0}^{n} w_j x_j) \tag{2.1}$$

$$y = f(\varphi) \tag{2.2}$$

Die Aktivierungsfunktion, oder auch Transferfunktion, eines Neurons kann linear oder nicht linear sein. Ist die Transferfunktion linear, ergibt ein mehrschichtiges NN keinen Sinn, da sie zu einer Schicht vereinfacht werden können. Lineare NNs sind nicht in der Lage, nicht lineare Probleme zu lösten [14]. Nicht lineare Transferfunktionen sind interessanter, da sie für nicht lineare Probleme antworten liefern können. In Abbildung 2.3 sind zwei Aktivierungsfunktionen zu sehen. In Abbildung 2.3a

Rectified Linear Unit (ReLU)



(a) Standardsigmoide Aktivierungsfunktion **(b)** Rectified Linear Aktivierungsfunktion

Abbildung 2.3: Beispiele für Aktivierungsfunktionen

2.2.2 Backpropagation

Als Backpropagation wird das Verfahren der Fehlerrückführung beschrieben. Es gehört zu der Familie der überwachten Lernverfahren

2.2.3 Loss Functions

2.2.4 Optimierer

2.2.5 Quantisierung

Quantisierung ist ein Signalverarbeitungsverfahren, bei welchem Eingabewerte auf eine vorher festgelegte kleinere Menge von Ausgabewerten abgebildet wird. Ein simples Beispiel für Quantisierung ist das Abbilden von rationalen Zahlen auf ganze Zahlen, hierfür müssen die rationalen Zahlen zu der nächsten ganzen Zahl gerundet werden. Im Bereich der Informatik werden für Gleitkomma Eingabewerte oft Festkommazahlen oder Ganzzahlen als Ausgabewerte gewählt [15]. Egal wie und welche die Quantisierung stattfindet, das Ziel ist es weniger Speicherkapazität und weniger Berechnungszeit zu benötigen mit minimalen Präzisionsverlust. Welches Quantisierungsschema verwendet wird, hängt von dem Anwendungsfall ab und kann nicht allgemein bestimmt werden. Es ist immer ein abwägen von Leistung und Präzision.

Dieses Verfahren eignet sich gut für Anwendungsgebiete mit wenig Speicher- und Rechenkapazität, wie beispielsweise der Einsatz von NNs bei Mobilgeräten [15], [16]. Der Grund dafür ist zweierlei. Erstens sorgt Quantisierung dafür, dass weni-

ger Platz im cache der CPU gebraucht wird, wodurch weniger Schreib- und Lesezugriffe ausgeführt werden und somit die Berechnung schneller ist. Zweitens ermöglicht die Abbildung auf kleinere Datentypen einen Performance-Gewinn, durch die effizientere Verwendung von Prozessor internen Recheneinheiten die beispielsweise SIMD unterstützen. Zudem ermöglicht die Abbildung auf Ganzzahl Typen die Nutzung von CPU internen Ganzzahl-Recheneinheiten, die effizienter als die Gleitkommazahl äquivalente Funktionieren, falls überhaupt vorhanden [17].

Das Problem der Quantisierung ist das Einbauen von „Fehlern“. Bei NNs wird oft von Fehler-Kumulierung gesprochen, da bei der Aktivierung eines NNs in jedem Quantisierten Neuron der Fehler wächst [18].

2.3 Training

2.4 SIMD

Der Begriff SIMD kommt von der flynnschen Klassifikation, die Rechnerarchitekturen in vier gebiete aufteilt [19]. Die Aufteilung orientiert sich an der Anzahl vorhandener Befehls- und Datenströme. Es gibt Single und Multiple Instructions, sowie Single und Multiple Data. Die daraus entstehenden Klassen heißen: SIMD, Single Instruction, Single Data (SISD), Multiple Instruction, Multiple Data (MIMD) und Multiple Instruction, Single Data (MISD).

In diesem Abschnitt geht es um SIMD. SIMD ermöglicht in einem Prozessor Befehlszyklus, eine Instruktion auf mehere Elemente eines Vektors gleichzeitig durchzuführen. Es gibt je nach Mikroprozessor-Architektur, verschiedene Erweiterungen, um SIMD zu implementieren. In dieser Arbeit sind alle Beispiele mit dem Advanced Vector Extensions 2 (AVX2) Befehlssatz geschrieben. Der Grund dafür ist, dass AVX2 von Modernen Intel und AMD Mikroprozessoren unterstützt werden.

2.4.1 Memory Alignment

2.5 NNUE

Die NNUE Evaluationsfunktion evaluiert eine Schachposition auf einer CPU ohne eine Notwendigkeit für eine GPU. Das die NNUE Evaluation eine Chance hat, besser als die HCE zu sein, muss sie schnell berechenbar sein. Anderenfalls sieht sie nicht weit genug in die Zukunft. Eine Untersuchung der Relation zwischen Suchtiefe und Spielstärke des Schachcomputer Houdini 2013 hat ergeben, dass Suchtiefe einen sehr großen Einfluss auf die Spielstärke hat, aber auch das dieser Effekt mit zunehmender Tiefe kleiner wird [20]. Ein weiter Vorteil von NNUEs ist es, das sie ein Eins-zu-eins-Ersatz für HCEs sind, es wird lediglich ein Netz und der CPU optimierte code zur Verwendung des Netzes gebraucht.

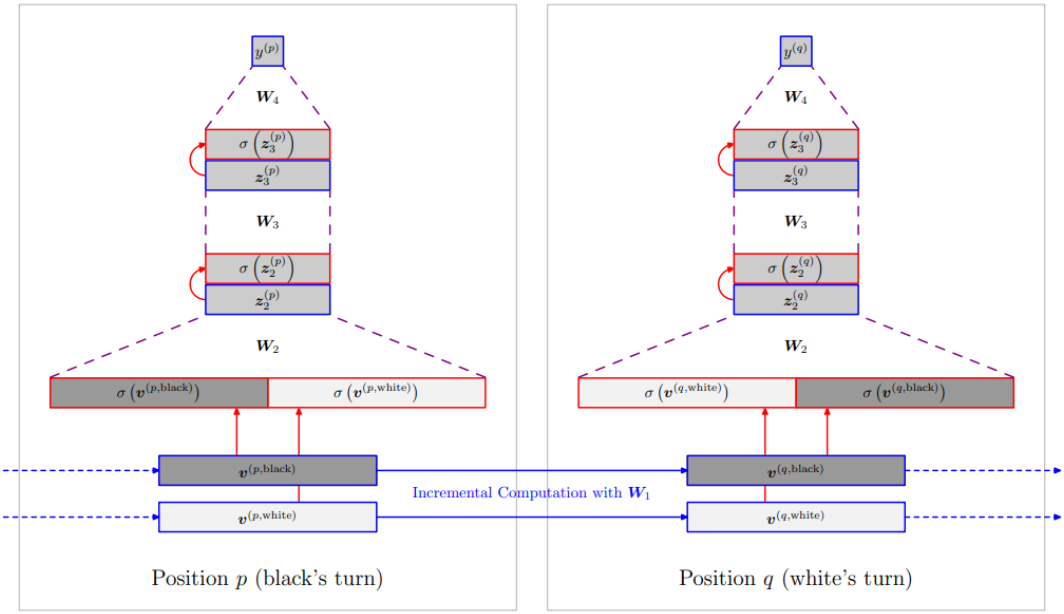


Abbildung 2.4: NNUE Evaluationsfunktion für die Evaluation von Position q, dabei unterscheidet sich q von p nur um einen Zug. Abbildung für die Evaluation des Shogicomputers „the end of genesis T.N.K.evolution turbo type D“ [3]

In Abbildung 2.4 ist der Aufbau der NNUE Evaluationsfunktion, wie sie von Nasu [3] entwickelt wurde, gezeigt. Sie eignet sich für die schnelle Berechnung auf einer CPU. In den folgenden Unterkapiteln wird genauer darauf eingegangen, warum das der Fall ist.

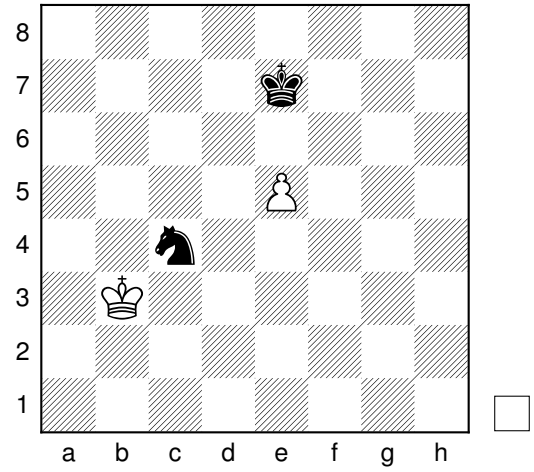


Abbildung 2.5: Exemplarische Schachposition. Weiß am Zug

2.5.1 Feature Set

Das Feature Set bestimmt die Form des Vektors, die der Eingabeschicht des NNs gegeben wird. Ein simples Feature Set setzt sich aus der Position, dem Figurentyp und seiner Farbe zusammen, mit 64 Feldern, sechs verschiedenen Figurentypen und zwei Farben, gibt es $64 * 6 * 2 = 768$ Merkmale. Ein Merkmal ist entweder 0 oder 1, je nachdem ob es für auf dem Feld eine Figur mit der Endsprechenden Farbe steht. Da im Schach maximal 32 Figuren im Spiel sind, kann es nur 32 gleichzeitig aktive Merkmale geben. In Abbildung 2.5 gibt es vier aktive Features: (B3, König, Weiß), (C4, Springer, Schwarz), (E5, Bauer, Weiß), (E7, König, Schwarz). Wenn der weiße König den Springer schlägt, ändern sich drei Features. Die Features (B3, König, Weiß) sowie (C4, Springer, Schwarz) werden inaktiv und ein neues Feature (C4, König, Weiß) wird aktiv. Bei diesem Feature Set ändern sich von einer Position p zu einer Position q vier Features (Rochade) maximal und im Durchschnitt drei Features [21]. Das Feature Set erfüllt die zwei Voraussetzungen, die für ein NNUE gelten:

1. Die Anzahl der aktiven Merkmale ist klein.
2. Die Anzahl der unterschiedlichen Merkmale von Position p nach Position q ist minimal.

Anhand dieser zwei Regeln lässt sich auch die Frage, warum sind nicht Elemente, die schon in der HCE verwendet werden (wie z. B. Rochade-Rechte) Teil des Feature Sets, beantworten. Es erhöht die Anzahl aktiver Merkmale und die Anzahl

durchschnittlicher Änderungen. Der Gewinn an Evaluationsgenauigkeit rechtfertigt nicht die Geschwindigkeitseinbußen [21].

In der Praxis bessere Feature Sets, als das eingangs erklärte Beispiel. Das am weitesten verbreitete Feature Set ist das sogenannte HalfKP Feature Set [3], [21]. Es besteht aus dem Tupel (Feld des eigenen Königs, Feld der Figur, Figurentyp, Farbe der Figur), wobei der Figurentyp kein König sein kann. Die Anzahl aktiver Merkmale ist hier ebenfalls maximal 32. Die gesamte Anzahl der Merkmale ist $64 * 64 * 5 * 2 = 40960$. Von einer zu der anderen Position ändert sich im Schnitt öfter etwas, da bei einem Zug des Königs alle aktiven geändert werden. Das ist eine bessere Aufteilung der Merkmale, da sich im Schach der König selten bewegt und durch dieses Feature Set das NN besser versteht wie die Figuren in Relation zum König stehen [21]. Es ist bekannt, dass überparametrisierte Netze, als Netze mit mehr Parametern als theoretisch nötig, besser lernen und gut generalisieren. Normalerweise sorgen mehr versteckte Schichten oder größere versteckte Schichten für die Überparametrisierung **Du2018**, **allen2019learning**. In dem fall Schach Evaluation, ist das aufgrund der nötigen Geschwindigkeit nicht möglich.

HalfKP alleine spiegelt nicht die gesamte Position wider, wie der Name impliziert, fehlt der gegnerische König. Deshalb werden die zwei Seiten separat behandelt, es gibt einen Vektor pro Seite. Das bedeutet es gibt doppelt so viele aktive Merkmale und doppelt so viele Änderungen, insgesamt zahlt sich der Kompromiss immer noch aus [21]. Wie die zwei Vektoren kombiniert werden, ist in im nächsten Unterkapitel erläutert.

HalfKP stammt aus der Shogi Welt, in der es keine Rochade gibt und somit die Relation der Figuren zum König wichtiger ist. Für Schach gibt es keine logische Begründung warum, HalfKP eine gute Repräsentation ist. HalfKP ist nur empirisch zu rechtfertigen und bildet die Grundlage für alle anderen verwendeten Feature Sets [21].

2.5.2 Akkumulator

Wie in Unterabschnitt 2.5.1 angesprochen und in Abbildung 2.4 zu sehen, werden für die Darstellung einer Schachposition mit HalfKP zwei Vektoren gebraucht. Ein Vektor $v^{(p,white)}$ für Weiß und einer für $v^{(p,black)}$ für Schwarz. Die zwei Vektoren müssen kombiniert werden, um sie in die nächste Schicht weiterzugeben. Für ge-

wöhnlich geben Schachcomputer die Evaluation immer aus der Sicht, der Seite die gerade am Zug ist an. Deshalb konkatenieren wir $v^{(p,white)}$ mit $v^{(p,black)}$, wenn Weiß am Zug ist und $v^{(q,black)}$ mit $v^{(q,white)}$, wenn Schwarz im nächsten Zug dran ist.

Es gibt verschiedene Wege für die zwei Eingabevektoren gehandhabt werden können [21]. Entweder beide Seiten verwenden die gleichen Gewichte oder die Gewichte sind Seiten spezifisch. Für den ersten Ansatz muss das Brett für Schwarz (kann auch Weiß sein) gespiegelt werden, weil ein weißer König auf E1 anders als ein schwarzer König auf E1 zu deuten ist. Alternativ sind die Gewichte Seiten spezifisch. Dieser Ansatz, scheint logischer, da Weiß und Schwarz nicht gleich Spielen. Die Nachteile ist ein größeres NN und eine längere Trainingszeit.

Kapitel 3

Verwandte Arbeiten

Auch wenn NNUEs erst seit 2020 existiert, hat es einen großen Einfluss auf die Schachcomputerlandschaft. In der letzten Saison (Saison 22) der Top Chess Engine Championship (TCEC) [22] spielen fünf der acht Teilnehmer der höchsten Division mit einer hybriden NNUE Evaluation, die restlichen drei nutzen einen von AlphaZero etablierten NN Ansatz, der später in diesem Kapitel genauer erläutert ist.

Stockfish war der erste Schachcomputer mit NNUE und manifestierte so seine Stellung als stärkster Schachcomputer. Die Entwicklung ist ein Community-Projekt. Das auf SETI@home [23] basierende Testing Framework Fishtest ermöglicht das Testen tausender Versionen. Alle Änderungen der Codebasis und neue NNUEs werden durch die Plattform getestet. Stand August 2022 gibt es 289 Entwickler und 1747 Tester, die 126.000 Tests seit der Entstehung der Plattform 2013 durchgeführt haben [24]. Dieser Ansatz ist ein großer Faktor dafür, wie Stockfish der beste Schachcomputer wurde und auch zukünftig bleibt. Die meisten anderen NNUE Schachcomputer bauen auf der Variante von Stockfish auf.

Die Architektur der Stockfish NNUE ist aktuell in seiner fünften Version. Sie besteht aus einem Feature Set mit 45.056 Eingabeparametern, namens HalfKAv2_hm, die inkrementell in zwei Farben abhängigen Akkumulatoren aktualisiert werden. Die Ausgabe der Eingabeschicht besteht aus jeweils 520 Ausgabewerten, welche in einen Vektor mit acht Werten und einen mit den restlichen 512 Werten geteilt wird. Die zwei Vektoren mit acht Werten werden basierend auf der Seite, welche am Zug ist, angepasst. Anhand der Phase des Spiels wird einer der sogenannten Buckets gewählt. Konkret bestimmt die Anzahl der im Spiel stehenden Figuren die Spielphase: $\lfloor \frac{pieceCount-1}{4} \rfloor$. Das gleiche Verfahren wird auch zur Auswahl der Schichten der versteckten Schichten verwendet. Die Buckets beinhalten zwei Schichten, die

1024 Werte gewichten und mit einer Clipped ReLU oder der Quadratwurzel einer Clipped ReLU aktivieren. Alle Schichten sind linear und die Quantisierung wird schon während des Trainings angewandt [21].

Der Unterschied von HalfKAv2_hm zu dem in dieser Arbeit verwendeten HalfKP Feature Set ist, dass der König selbst als Figur enthalten ist. Jedoch werden die Könige egal welcher Farbe als ein Figurentyp angesehen, da die Belegung ihrer Felder disjunkt ist. Sie können also nie auf demselben Feld stehen. So werden acht Prozent der Eingabeparameter gespart. „hm“ steht für „horizontally mirrored“, auf Deutsch horizontal gespiegelt. Das bedeutet, das Brett wird vor der Erstellung der Eingabeparameter gespiegelt, sodass der eigene König immer auf einem der e bis h (je nach Konvention auch a bis d) Ränge ist. Das hört sich unintuitiv an und spiegelt nicht die Realität wider, eignet sich trotzdem, da es die Größe des Netzes stark reduziert, aber ein Unterschied in Spielstärke kaum messbar ist [21].

Schon vor der Entwicklung von NNUEs gab es einen NN basierten Schachcomputer namens AlphaZero [5], der den schon damals den noch HCE basierten Schachcomputer Stockfish vernichtend schlagen konnte. AlphaZero wurde 2017 von zu Google gehörenden Forschungsunternehmen DeepMind entwickelt. Es erlernte laut DeepMind bereits nach vier Stunden Self-Play reinforcement Training die nötige Spielstärke, um gegen Stockfish zu gewinnen. DeepMinds nennt den Trainingsansatz *tabula rasa*. Nennenswert ist das zur Suche Monte Carlo tree search (MCTS), statt der normalerweise genutzten Alpha-Beta Suche, verwendet wird. AlphaZero kennt nur die Spielregeln und trainiert sein Convolutional Neural Network (CNN) durch Self-Play reinforcement Training. Der Ansatz ist generell anwendbar und hat in den Spielen Schach, Shogi und Go gegen die führenden Computer-Programme gewonnen.

Dem erstmals durch einen Schachcomputer geschlagenen damalige Schachweltmeister Kasparov [25] gefällt der dynamische und offene Spielstil von AlphaZero, der anders als der von HCE basierten Schachcomputern auf konventionellem Wissen aufbauende Spielstil. Er beschrieb AlphaZero als Experten und nicht als das Werkzeug eines Experten. Damit deutet Kasparov darauf hin, dass ein Schachspieler, besonders Super-Großmeister, diesen Schachcomputer nicht nur für die Analyse ihrer Züge nehmen kann, sondern auch für das Entdecken neuer Spielweisen, die vorher nicht in Betracht gezogen wurden.

Leider ist AlphaZero nicht öffentlich zugänglich und wird auch von DeepMinds nicht weiter entwickelt. Andere Entwickler nutzen jedoch die Herangehensweise von AlphaZero und entwickeln ihr eigenes NN nach diesem Ansatz. Der aktuell stärkste Nachfolger heißt Leela Chess Zero (Lc0) [26]. Lc0 hat gemessen an der letzten TCEC Saison eine Elo von 3586 [22] und ist somit wahrscheinlich stärker als AlphaZero. Lc0 belegte in der 22ten TCEC Saison den dritten Platz hinter den zwei NNUE Schachcomputern Stockfish und KomodoDragon [27]. Es bleibt spannend, welcher Ansatz sich durchsetzen wird.

Kapitel 4

NNUE Implementierung

Ziel dies Kapitels ist es, Architektur und Implementierung der im Rahmen dieser Arbeit entwickelten NNUE Evaluationsfunktion zu erläutern. Es wird anfangs auf die Architekturendscheidungen eingegangen. Danach wird geklärt wie diese Entscheidungen

Kapitel Abschnitt 2.1 zeigt wie die herkömmliche Art und Weise der Positions-Evaluation funktioniert. Nach kurzer Überlegung wird aber klar, dass die HCE nur so gut sein kann wie die Schachspieler die sie Entwickeln. Natürlich können die darin verwendeten Parameter durch Optimierungsalgorithmen wie genetische Algorithmen oder Simulated Annealing maximiert werden, letztendlich bleibt der limitierende Faktor das Spielverständnis der Entwickler. Die NNUE Evaluation ist nicht an solche Limitierungen gebunden und kann auf eine ganz andere Art und Weise entscheiden welche Faktoren wichtig für die Evaluation einer Schachposition sind. Die Entwicklung der Schachcomputerlandschaft zeigt, das diese Herangehensweise der HCE überlegen ist.

4.1 Architektur

Aufgrund des zeitlich begrenzten Rahmen dieser Arbeit, ist die verwendete Architektur keine neue

Das Resultierende NN hat 10,5 Millionen Trainierbare Parameter. Tatsächlich werden bei einer aktivierung Maximal

4.1.1 Eingabeschicht

4.1.2 Versteckte Schicht

4.1.3 Ausgabeschicht

4.2 Training

4.2.1 Eingabedaten

Die Erzeugung der Eingabedaten ist nicht teil dieser Arbeit. Jedoch ist es wichtig zu wissen wie die Eingabedaten generiert werden und wie sie in den Trainer geladen werden, um zu verstehen, wie das neuronale Netz lernt. Im Training für diese Arbeit wurden drei verscheiden generierte Datensätze verwendet. Diese Datensätze wurden von Stockfish für das Training der neusten Variante ihres NNUEs verwendet [28]. Sie

4.3 Integration in einen Schachcomputer

4.3.1 Eingabeschicht

4.3.2 Versteckte Schicht

- simple -> quantization

Kapitel 5

Ergebnisse

5.1 Testaufbau

Der Testaufbau soll dafür sorgen, dass die hier erreichten Ergebnisse reproduzierbar sind. Außerdem wird in diesem Kapitel die Auswahl der verwendeten Komponenten erläutert.

Schachcomputer sind von Natur aus deterministisch, deshalb wird zur Vermeidung des immer gleichen Spielablaufs ein Eröffnungsbuch verwendet. Dafür wird das Unbalanced Human Openings (UHO) V3 von Pohl [29] zusammengestellte Eröffnungsbuch verwendet. UHO enthält Eröffnungen aus Spielen starker Schachspieler (2300+ Elo), bei denen eine Analyse durch KomodoDragon [27] ein Vorteil für Weiß vorliegt. Diese Eröffnungen eignen sich gut für Schachcomputer, da so weniger remis gespielt werden, als bei ausgeglichenen Eröffnungen. Konkret werden Eröffnungen mit sechs Zügen und einem Vorteil von +0.90 bis +0.99 für Weiß verwendet. Jede Eröffnung wird von beiden Computern mit beiden Farben gespielt.

5.2 Verbesserungen

5.3 Probleme

Kapitel 6

Fazit und Ausblick

Die Verwendung des UHO Eröffnungsbuch ergibt normalerweise bei dem Computer gegen Computer Vergleich sinn, da die hier getesteten Schachcomputer aufgrund fehlender Tiefe und Tablebase Schwierigkeiten im Endspiel haben, ist es vermutlich von kleiner Bedeutung welche Eröffnungen gewählt worden.

6.1 Persönliche Bemerkungen

Final möchte ich meine persönlichen Gedanken zu diesem Thema schildern und welche Probleme mir während der Erstellung dieser Arbeit entgegengekommen sind.

Abkürzungsverzeichnis

NNUE Efficiently Updatable Neural Network
SIMD Single Instruction, Multiple Data
SISD Single Instruction, Single Data
MIMD Multiple Instruction, Multiple Data
MISD Multiple Instruction, Single Data
AVX2 Advanced Vector Extensions 2
HCE hand-crafted evaluation
KNN künstliches neuronales Netz
NN neuronales Netz
CNN Convolutional Neural Network
DNN Deep Neural Network
FNN Feedforward Neural Network
KIS Künstliche Intelligenz für autonome Systeme
Lc0 Leela Chess Zero
TCEC Top Chess Engine Championship
MCTS Monte Carlo tree search
ReLU Rectified Linear Unit
UHO Unbalanced Human Openings

Tabellenverzeichnis

Abbildungsverzeichnis

Literatur

[1] Alan Turing, *Faster Than Thought - A Symposium on Digital Computing Machines*. London: Sir Isaac Pitman & Sons, 1953, 1953.

[2] Claude E. Shannon, „XXII. Programming a computer for playing chess“, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Jg. 41, Nr. 314, S. 256–275, März 1950. DOI: 10.1080/14786445008521796.

[3] Yu Nasu. „NNUE: Efficiently Updatable Neural-Network-based Evaluation Functions for Computer Shogi“. (2018), Adresse: https://www.apply.computer-shogi.org/wcsc28 /appeal/the_end_of_genesis_T.N.K.evolution_turbo_type_D/nnue.pdf (besucht am 28. 04. 2018).

[4] Tord Romstad, Marco Costalba, Joona Kiiski und Gary Linscott. „Introducing NNUE Evaluation“. (2020), Adresse: https://stockfishchess.org/blog/2020/introducing-nnue-evaluation/ (besucht am 07. 08. 2020).

[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan und Demis Hassabis, *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*, 2017. DOI: 10.48550/ARXIV.1712.01815.

[6] Eduardo Vazquez-Fernandez und Carlos A. Coello Coello, „An adaptive evolutionary algorithm based on tactical and positional chess problems to adjust the weights of a chess engine“, in *2013 IEEE Congress on Evolutionary Computation*, IEEE, Juni 2013. DOI: 10.1109/cec.2013.6557727.

[7] Joel Staubach und Marvin Karhan. „Ein Algorithmenbasierter Schachcomputer“. (2022), Adresse: https://github.com/marvinkarhan/chess-engine/blob/master/Karhan_

Staubach_Ein_algorithmenbasierter_Schachcomputer.pdf (besucht am 17. 08. 2022).

[8] James R. Slagle und John E. Dixon, „Experiments With Some Programs That Search Game Trees“, *Journal of the ACM*, Jg. 16, Nr. 2, S. 189–207, Apr. 1969. DOI: 10.1145/321510.321511.

[9] David Levy, Hrsg., *Computer Chess Compendium*. Springer New York, 1988. DOI: 10.1007/978-1-4757-1968-0.

[10] Maciej Krawczak, *Multilayer Neural Networks*. Springer, 2013.

[11] Frank Rosenblatt, „The perceptron: a probabilistic model for information storage and organization in the brain.“, *Psychological review*, Jg. 65, Nr. 6, S. 386, 1958.

[12] Jürgen Schmidhuber, „Deep learning in neural networks: An overview“, *Neural Networks*, Jg. 61, S. 85–117, Jan. 2015. DOI: 10.1016/j.neunet.2014.09.003.

[13] Warren S. McCulloch und Walter Pitts, „A logical calculus of the ideas immanent in nervous activity“, *The Bulletin of Mathematical Biophysics*, Jg. 5, Nr. 4, S. 115–133, Dez. 1943. DOI: 10.1007/bf02478259.

[14] Marvin Minsky und Seymour Papert, *Perceptron: an introduction to computational geometry*, 1969.

[15] Philipp Gysel, Mohammad Motamedi und Soheil Ghiasi, *Hardware-oriented Approximation of Convolutional Neural Networks*, 2016. DOI: 10.48550/ARXIV.1604.03168.

[16] Jiali Ma, Zhiqiang Zhu, Leyu Dai und Songhui Guo, „Layer-by-Layer Quantization Method for Neural Network Parameters“, in *Proceedings of the International Conference on Industrial Control Network and System Engineering Research*, Ser. ICNSER2019, Shenyang, China: Association for Computing Machinery, 2019, S. 22–26. DOI: 10.1145/3333581.3333589.

[17] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam und Dmitry Kalenichenko, *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*, 2017. DOI: 10.48550/ARXIV.1712.05877.

[18] Eunhyeok Park, Dongyoung Kim, Sungjoo Yoo und Peter Vajda, *Precision Highway for Ultra Low-Precision Quantization*, 2018. DOI: 10.48550/ARXIV.1812.09818.

[19] Michael J. Flynn, „Some Computer Organizations and Their Effectiveness“, *IEEE Transactions on Computers*, Jg. C-21, Nr. 9, S. 948–960, 1972. DOI: 10.1109/TC.1972.5009071.

[20] Diogo R. Ferreira, „The Impact of the Search Depth on Chess Playing Strength“, *ICGA Journal*, Jg. 36, Nr. 2, S. 67–80, Juni 2013. DOI: 10.3233/icg-2013-36202.

[21] Joost VandeVondele Thomasz Sobczyk Hisayori Noda, *NNUE*, https://github.com/glinscott/nnue-pytorch/blob/master/docs/nnue.md, 2022.

[22] TCEC. „TCEC Leagues Season 22 Engines“. (2022), Adresse: https://wiki.chessdom.org/TCEC_Leagues_Season_22_Engines (besucht am 28. 04. 2018).

[23] E. Korpela, D. Werthimer, D. Anderson, J. Cobb und M. Leboisky, „SETI@home-massively distributed computing for SETI“, *Computing in Science & Engineering*, Jg. 3, Nr. 1, S. 78–83, 2001. DOI: 10.1109/5992.895191.

[24] Stockfish. „Users | Stockfish Testing“. (2022), Adresse: https://tests.stockfishchess.org/users (besucht am 24. 08. 2022).

[25] Garry Kasparov, „Chess, a *Drosophila* of reasoning“, *Science*, Jg. 362, Nr. 6419, S. 1087–1087, Dez. 2018. DOI: 10.1126/science.aaw2221.

[26] UCT/NN AI Community. „Leela Chess Zero: Open source neural network based chess engine“. (2022), Adresse: https://lczero.org/ (besucht am 17. 08. 2022).

[27] Mark Lefler Don Dailey Larry Kaufman. „Dragon by Komodo Chess“. (2022), Adresse: https://komodochess.com/ (besucht am 24. 08. 2022).

[28] Joost VandeVondele, *Update default net to nn-3c0054ea9860.nnue*, https://github.com/official-stockfish/Stockfish/pull/4100, 2022.

[29] Stefan Pohl. „Anti Draw Openings - The future of Computerchess“. (2022), Adresse: https://www.sp-cc.de/anti-draw-openings.htm (besucht am 26. 08. 2022).