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. 13 Seiten.

Karhan, Marvin:
Development of an NNUE for the Evaluation of Chess Positions / Marvin Karhan. –
Bachelor Thesis, Mannheim: University of Applied Sciences Mannheim, 2022. 13 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:

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

Shogi ist die japanische Variante des Schachspiels. 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 was praktisch für die Auswahl eines passenden Feature Sets ist, wie in Unterabschnitt 4.1.1 genauer erläutert ist.

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. 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. Zuletzt erläutert was Single Instruction, Multiple Data (SIMD) ist und wie diese Vektoroperationen in C/C++ verwendet werden können.

2.1 Hand-crafted Evaluation

Die Evaluation 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 die-

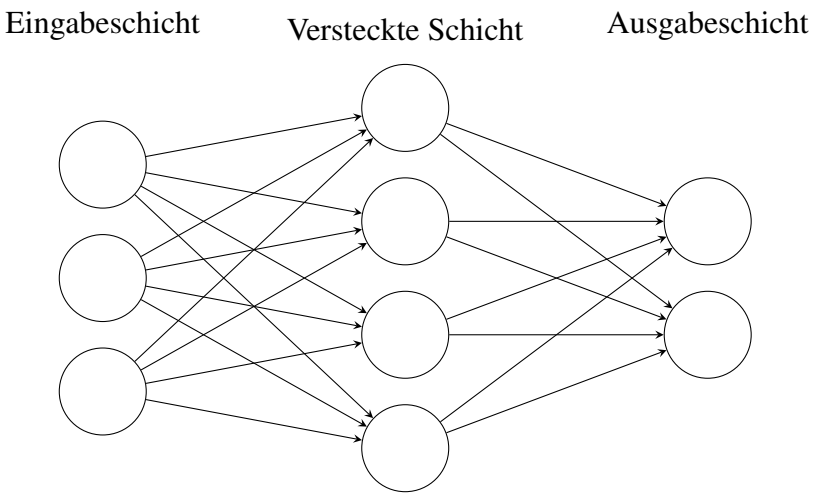


Abbildung 2.1: Ein einfaches neuronales Netz

ser 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 Zeit 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“.

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.

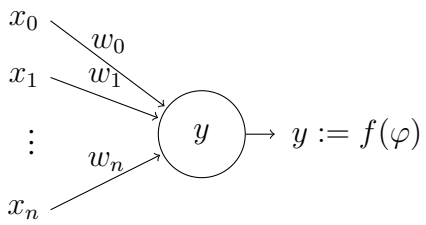
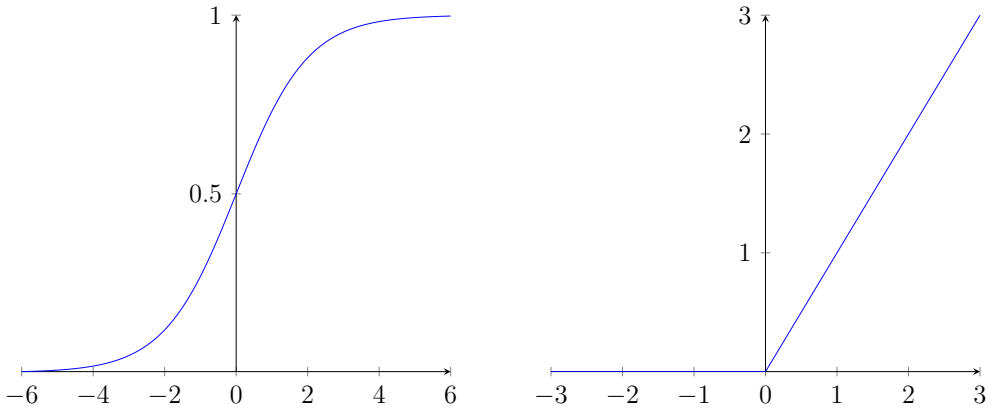


Abbildung 2.2: Ein einzelnes Neuron mit seinen eingabe- und Ausgabekomponenten

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 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 den folgenden Unterabschnitten wird Grundlegend auf die Einzelteile neuronaler Netze eingegangen. To-do.



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

Abbildung 2.3: Beispiele für Aktivierungsfunktionen

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

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 weniger 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, SISD, MIMD und 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

Kapitel 3

Verwandte Arbeiten

Schon vor der Entwicklung von NNUEs gab es einen NN basierten Schachcomputer namens AlphaZero, der den schon damals den noch HCE basierten Schachcomputer Stockfish vernichtend Schlagen konnte [5]. AlphaZero wurde 2017 von zu Google gehörenden Forschungsunternehmen DeepMind entwickelt. Es erlernte laut DeepMind bereit nach vier Stunden Self-Play reinforcement Training die nötige Spielstärke, um gegen Stockfish zu gewinnen. DeepMinds Ansatz

Der erstmals durch einen Schachcomputer geschlagene damalige Schachweltmeister Kasparov [20] gefällt der dynamische und offener Spielstil von AlphaZero, der anders als der von HCE basierten Schachcomputer auf konventionellem wissen aufbauende Spielstil. Er beschrieb AlphaZero als Experten und nicht als das Werkzeug eins Experten. Damit deutet Kasparov darauf hin, dass 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 worden wären.

AlphaZero nicht öffentlich zugänglich und wird auch von DeepMinds Seite nicht weiter entwickelt. Andere Entwickler nutzen jedoch die Herangehensweise von AlphaZero und bauen ihr eigens NN nach diesem Ansatz. Der aktuell stärkste Nachfolger heißt Leela Chess Zero (Lc0). Lc0 hat gemessen an dem letzten Top Chess Engine Championship (TCEC) Superfinal gegen Stockfish eine Elo von 3586 [21] und ist somit wahrscheinlich stärker als AlphaZero. In diesem Kampf gegen Stockfish ging Stockfish mit seiner NNUE basierten Variante siegreich hervor. Also bleibt es spannend welcher Ansatz sich durchsetzen wird.

Kapitel 4

NNUE

Ziel dies Kapitels ist es,

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 ein NN basierte

4.1 Architektur

4.1.1 Feature Set

4.1.2 Eingabeschicht

4.1.3 Versteckte Schicht

4.1.4 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 **StockfishNewestNetJul04**. Sie

4.3 Integration in einen Schachcomputer

4.3.1 Eingabeschicht

4.3.2 Versteckte Schicht

- simple -> quantization

Kapitel 5

Ergebnisse

5.1 Verbesserungen

Kapitel 6

Fazit und Ausblick

Wie in Kapitel 5 festgestellt, hat das NNUE hat nicht die erwarteten Ergebnisse geliefert. Dafür kann es mehrere Gründe geben:

Die Implementierung des Netzes in dem Schachcomputer kann Fehler enthalten. Die rekursive Art und Weise, der Zugsuche macht es nicht einfach mögliche Fehlerquellen zu identifizieren. In einer zeitlich limitierten Anstrengung Fehler in der Implementierung zu finden, wurde das Einlesen der Gewichte und Bias, sowie das Verhältnis zwischen updates und refreshes des Akkumulators überprüft. Es konnten keine Unregelmäßigkeiten festgestellt werden. Fehler an anderen stellen der Implementierung sind wahrscheinlich und können ausschlaggebend für die vorzufindenden Ergebnisse sein. Leider waren weitere Tests im Rahmen dieser Arbeit nicht möglich.

Ein weiter Grund für eine nicht vorhandene Steigerung der Spielstärke kann außerhalb der Evaluation liegen. Die zugrundeliegende Suche von des im Modul KIS entwickelten Schachcomputers ist nur minimal optimiert. Sie besteht aus einem Negamax Depth-First Suchalgorithmus [22] mit Move Ordering, Transposition Tables, Quiescence Search und Iterative Deepening. Es fehlen viele weit verbreitete pruning Techniken wie unter anderem: razoring [9, S. 123-128]

Abkürzungsverzeichnis

NNUE Efficiently Updatable Neural Network
SIMD Single Instruction, Multiple Data
SISD Single Instruction, Single Data
MIMD Multiple Instruction, Multiple Data
AVX2 Advanced Vector Extensions 2
HCE hand-crafted evaluation
KNN künstliches neuronales Netz
NN neuronales Netz
DNN Deep Neural Network
FNN Feedforward Neural Network
KIS Künstliche Intelligenz für autonome Systeme
Lc0 Leela Chess Zero
TCEC Top Chess Engine Championship

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

[21] Guy Haworth und Nelson Hernandez, „The 20th Top Chess Engine Championship, TCEC20.“, *J. Int. Comput. Games Assoc.*, Jg. 43, Nr. 1, S. 62–73, 2021.

[22] Murray S. Campbell und T.A. Marsland, „A comparison of minimax tree search algorithms“, *Artificial Intelligence*, Jg. 20, Nr. 4, S. 347–367, Juli 1983. DOI: 10.1016/0004-3702(83)90001-2.