

Generierung von Schachkommentaren mittels maschinellen Lernen

Max Semdner

Frankfurt University of Applied Sciences

31. Januar 2023

Inhaltsverzeichnis

- 1 Überblick
 - Schach und KI
 - Forschungsfrage
 - Aufbau
- 2 Schach Engine
 - Anforderungen
 - Brett Darstellung
 - Zugsuche und Positionsbewertung
 - Zusammenfassung
- 3 Schachkommentator
 - Encoder-Decoder Modell
 - Generationsmodelle
- 4 Fazit

1 Überblick

Schach und KI

- Eines der am längsten erforschten Teilgebiete der KI
- Häufig Forschung in Bezug auf Schach Engines
 - ▶ Ziel ist die Optimierung der Spielstärke eines Schachprogramms
 - ▶ Spielstärke von Engines liegt weit über der von Menschen
- Professionelle Schachspieler oder Kommentatoren werden benötigt um Absicht hinter Zügen zu verstehen
- Problem: Züge werden nicht immer richtig verstanden

Forschungsfrage

*Wie kann maschinelles Lernen genutzt werden,
um Kommentare zu Schachpartien zu generieren?*

- Der Prozess der Kommentarerzeugung wird in zwei Teile aufgeteilt
 - ▶ Bereitstellung von Informationen (Schach Engine)
 - ▶ Computerverständliche Schachbrettdarstellung
 - ▶ Zugsuche
 - ▶ Positionsbewertung
 - ▶ Generierung von Kommentaren (Virtueller Schachkommentator)
 - ▶ Festlegen was man übersetzen möchte
 - ▶ Architektur zur Erzeugung von Schachkommentaren

2 Schach Engine

Anforderungen

- Jede Schach Engine muss bestimmte Anforderungen erfüllen
 - ▶ Darstellung des Schachbretts
 - ▶ Suche nach den möglichen Spielzügen
 - ▶ Bewertung der Position

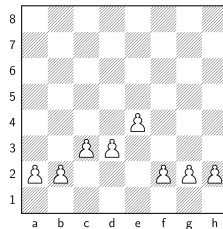
Brett Darstellung

- Brett und Figuren müssen in eine Computer verständliche Form gebracht werden
- Eine Möglichkeit der Darstellung sind Bitboards
- Jedes Feld wird dargestellt und kann entweder den Wert 0 oder 1 enthalten
 - ▶ 0 = Keine Figur auf Feld, 1 = Figur auf Feld
- Für jeden Figurtyp (6), Figurfarbe (2) und für Rochaden (4) werden Bitboards erstellt
 - ▶ 16 Bitboards insgesamt

Brett Darstellung

- Mögliche Implementierung durch 8×8 Arrays
- Züge können mit Hilfe von logischen Operationen berechnet werden
- Vorteil: Können als Input für neuronale Netze verwendet werden können

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	1	0	0	0
0	0	1	1	0	0	0	0
1	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0



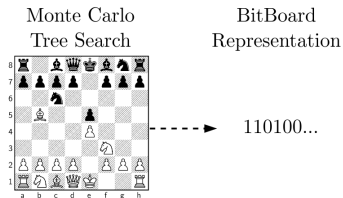
Zugsuche und Positionsbewertung

- Schach ist bislang ungelöst
- Suche nach dem besten Zug und Bewertung auf der Grundlage von Rechenfertigkeiten und Programmierung
- Um den besten Zug zu finden, muss die Engine zwei Aufgaben erfüllen
 - ▶ Legale, möglichen Züge in aktuellen und folgenden Positionen finden (Zugsuche)
 - ▶ Positionen bewerten
- Bekannte Implementierungen: MiniMax/Alpha-Beta Pruning und handgeschriebene Evaluationsfunktion

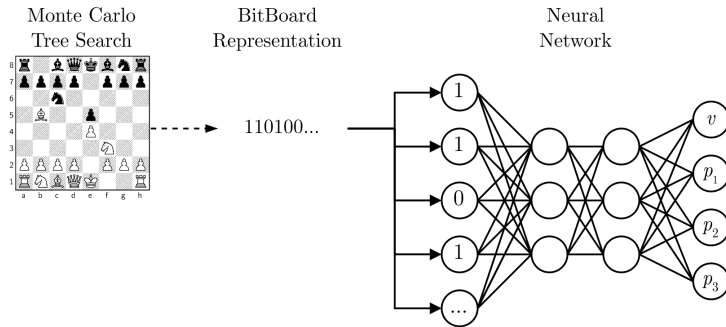
Zugsuche und Positionsbewertung

- Neue Implementierungen: Monte Carlo tree search (MCTS) und Neuronales Netzwerk
 - ▶ Ansatz mittels maschinellem Lernen
- Neuronales Netzwerk zum bewerten einer Position
- MCTS zum suchen von Zügen
- Pfad mit der besten Gesamtbewertung wird gespielt

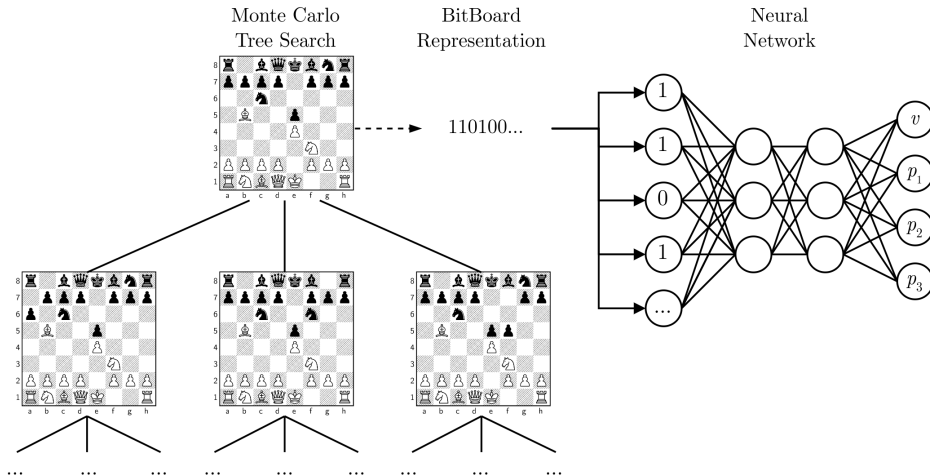
Zugsuche und Positionsbewertung



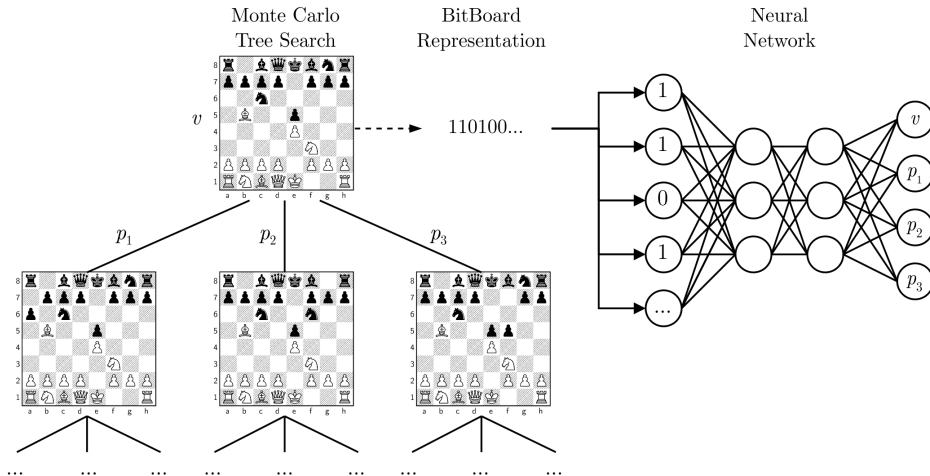
Zugsuche und Positionsbewertung



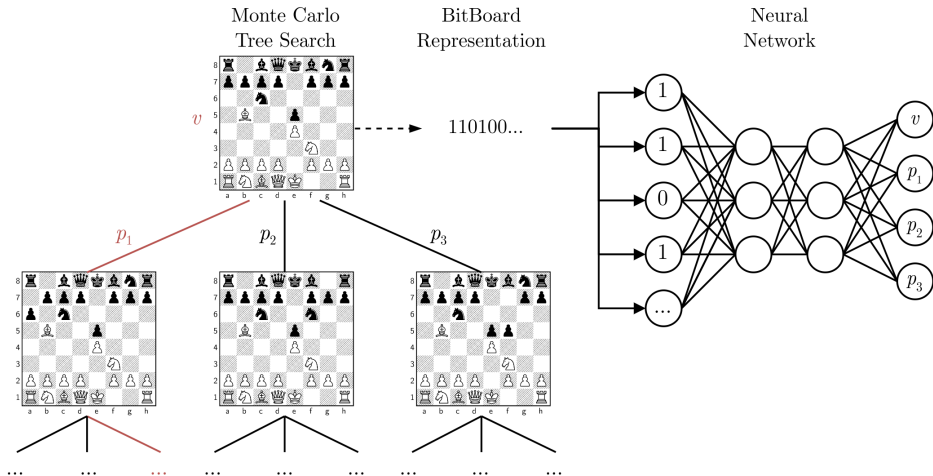
Zugsuche und Positionsbewertung



Zugsuche und Positionsbewertung



Zugsuche und Positionsbewertung

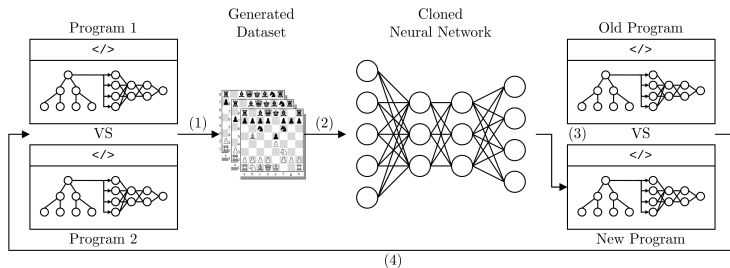


Zugsuche und Positionsbewertung

- Das Neuronale Netzwerk muss trainiert werden, um eine zuverlässige Ausgabe zu erzeugen
- Daten für das Training werden durch Selbstspiel generiert
- Zu Beginn existiert ein Programm mit Spielregeln, ein untrainiertes neuronales Netz und der MCTS-Algorithmus

Zugsuche und Positionsbewertung

1. Programm spielt gegen sich selbst und zeichnet jedes Spiel auf
 - Jede Partie bekommt verloren (-1), unentschieden (0) oder gewonnen (+1) zugerodnet
2. Neuronales Netzwerk wird geklont und die Parameter werden angepasst
3. Das neue Programm, spielt gegen das alte Programm
4. Das Programm, das gewinnt, wird ausgewählt und es beginnt wieder bei 1.



Zusammenfassung

■ Von der Schachengine bereitgestellte Informationen:

- ▶ Brettdarstellung (Bitboards)
- ▶ Positionsbewertung (v)
- ▶ Zugwahrscheinlichkeiten (p)
- ▶ Zugpfade

3 Schachkommentator

Encoder-Decoder Modell

- Schachkommentator nimmt Informationen und übersetzt sie in Kommentare
- Bereich: Sequence-to-sequence processing
 - ▶ Abbildung einer Eingabe auf eine Ausgabe
- Architektur: Encoder-Decoder Modell
 - ▶ Basierend auf bidirektionalen Long Short-Term Memory (LSTM)

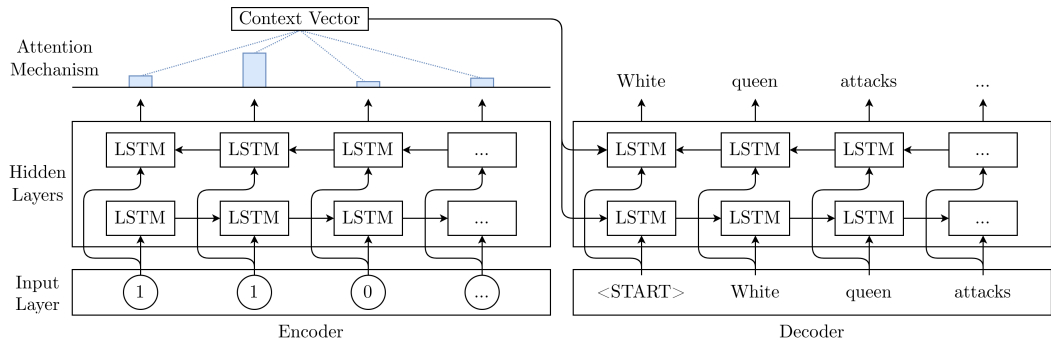
Encoder-Decoder Modell

- LSTMs sind eine spezielle Art von Rekurrenten Neuronalen Netzwerken
- RNNs sind Neuronale Netzwerke zur Verarbeitung von Datenfolgen
 - ▶ Outputs werden mit neuen Inputs an das Netzwerk gegeben
 - ▶ Zustand des Netzes repräsentiert Neuronen zu einem bestimmten Zeitpunkt
 - ▶ Netzwerk kann sich an laufende Muster erinnern und auf diese reagieren
- Bidirektionale RNNs (Bi-RNN) sind eine Erweiterung von RNNs
 - ▶ Berücksichtigen sowohl die vorherigen als auch die nachfolgenden Eingabedaten
- LSTMs sind spezielle Neuronen
 - ▶ Bei langen Sequenzen werden die Informationen aus der Vergangenheit nicht korrekt berücksichtigt
 - ▶ LSTMs lösen das Problem

Encoder-Decoder Modell

- Encoder-Decoder Modell benutzt Bi-LSTMs
 - ▶ in der maschinellen Übersetzung verwendet
- Besteht aus zwei Teilen: Encoder, Decoder
 - ▶ Encoder erhält Eingabe von Engine und wandelt diese in andere Darstellung um
 - ▶ Decoder verwendet diese Darstellung und erstellt entsprechende Kommentare
- Aufmerksamkeitsmechanismus wird verwendet, um auf wichtige Teile der Sequenz zu konzentrieren

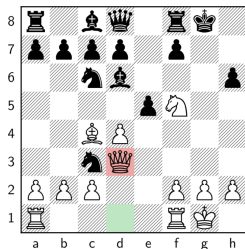
Encoder-Decoder Modell



Generationsmodelle

- Es muss definiert werden, welche Kategorien von Kommentaren generiert werden sollen

- ▶ Beschreibung
- ▶ Qualität
- ▶ Vergleich
- ▶ Planung
- ▶ Kontext



- Zug Start Zug Ende Zug: Dd3
- Beschreibung:** Die bedrohte weiße Dame entkommt und greift den Springer auf c3 an
- Qualität:** Nicht der beste Zug
- Vergleich:** Dd4+ ist ein besserer Zug
- Planung:** Weiß beschließt, die Dame zu schützen und den schwarzen Springer auf c3 anzugreifen.
- Kontext:** Weiß will seine Königin nicht verlieren

Generationsmodelle

- Einzelner Kommentator nicht aus
- Für jede Kategorie ein Kommentator in Form von Encoder-Decoder Modell (sog. Generationsmodelle)
- Generationsmodelle unterscheiden sich im Training
 - ▶ Unterschiedliche Merkmale und Anzahl an Zügen werden berücksichtigt

Generationsmodelle

■ Zug m , Position b

Generationsmodelle

- Zug m , Position b
- **Beschreibung:** $f_{\text{Decoder}}(f_{\text{SME}}(b_0, m_0)) \rightarrow C_{\text{Beschreibung}}$

Generationsmodelle

- Zug m , Position b
- **Beschreibung:** $f_{Decoder}(f_{SME}(b_0, m_0)) \rightarrow C_{Beschreibung}$
- **Qualität:** $f_{Decoder}(f_{Encoder}(b_0, b_1, v_1 - v_0)) \rightarrow C_{Qualitaet}$

Generationsmodelle

- Zug m , Position b
- **Beschreibung:** $f_{\text{Decoder}}(f_{\text{SME}}(b_0, m_0)) \rightarrow C_{\text{Beschreibung}}$
- **Qualität:** $f_{\text{Decoder}}(f_{\text{Encoder}}(b_0, b_1, v_1 - v_0)) \rightarrow C_{\text{Qualität}}$
- **Vergleich:** $f_{\text{Decoder}}(f_{\text{MME}}(b_1, m_0, b_2, m_1)) \rightarrow C_{\text{Vergleich}}$

Generationsmodelle

- Zug m , Position b
- **Beschreibung:** $f_{\text{Decoder}}(f_{\text{SME}}(b_0, m_0)) \rightarrow C_{\text{Beschreibung}}$
- **Qualität:** $f_{\text{Decoder}}(f_{\text{Encoder}}(b_0, b_1, v_1 - v_0)) \rightarrow C_{\text{Qualität}}$
- **Vergleich:** $f_{\text{Decoder}}(f_{\text{MME}}(b_1, m_0, b_2, m_1)) \rightarrow C_{\text{Vergleich}}$
- **Planung:** $f_{\text{Decoder}}(f_{\text{MME}}((b_2, m_1), (b_3, m_2), (b_4, m_3), \dots)) \rightarrow C_{\text{Planung}}$

Generationsmodelle

- Zug m , Position b
- **Beschreibung:** $f_{\text{Decoder}}(f_{\text{SME}}(b_0, m_0)) \rightarrow C_{\text{Beschreibung}}$
- **Qualität:** $f_{\text{Decoder}}(f_{\text{Encoder}}(b_0, b_1, v_1 - v_0)) \rightarrow C_{\text{Qualität}}$
- **Vergleich:** $f_{\text{Decoder}}(f_{\text{MME}}(b_1, m_0, b_2, m_1)) \rightarrow C_{\text{Vergleich}}$
- **Planung:** $f_{\text{Decoder}}(f_{\text{MME}}((b_2, m_1), (b_3, m_2), (b_4, m_3), \dots)) \rightarrow C_{\text{Planung}}$
- **Kontext:** $f_{\text{Decoder}}(f_{\text{MME}}((b_1, m_0), (b_2, m_1), (b_3, m_2), (b_4, m_3), \dots)) \rightarrow C_{\text{Kontext}}$

4 Fazit




Fazit

- Forschungsfrage zur Generierung von Kommentaren konnte beantwortet werden
- Problem: Kommentare noch keine Alternative zum Menschen
- Bildet Basis für weitere Forschung




Ende

Vielen Dank für eure Aufmerksamkeit!




Literatur I

-  Allis, L. V. (1994, July).
Searching for Solutions in Games and Artificial Intelligence.
Ph. D. thesis, University of Limburg.
-  Boskovic, B., S. Greiner, J. Brest, and V. Zumer (2005).
The representation of chess game.
In 27th International Conference on Information Technology Interfaces, 2005., pp.
359–364.
-  Cho, K., B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk,
and Y. Bengio (2014).
Learning phrase representations using rnn encoder-decoder for statistical machine
translation.





Literatur II

-  Cho, K., B. van Merriënboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio (2014).
Learning phrase representations using RNN encoder-decoder for statistical machine translation.
CoRR abs/1406.1078.
-  Czech, J., P. Korus, and K. Kersting (2021, May).
Improving alphazero using monte-carlo graph search.
In Proceedings of the International Conference on Automated Planning and Scheduling, 31, pp. 103–111. arXiv.
-  Hochreiter, S. and J. Schmidhuber (1997, 11).
Long Short-Term Memory.
Neural Computation 9(8), 1735–1780.




Literatur III

-  Jhamtani, H., V. Gangal, E. Hovy, G. Neubig, and T. Berg-Kirkpatrick (2018, July).
Learning to generate move-by-move commentary for chess games from large-scale social forum data.
In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Melbourne, Australia, pp. 1661–1671.
Association for Computational Linguistics.
-  Keen, B. (2009, November).
A history of computer chess.
In ECM3401, pp. 1–14.
-  Klein, D. (2022).
Neural networks for chess.



Literatur IV

-  Levy, D., D. Broughton, and M. Taylor (1989).
The sex algorithm in computer chess.
pp. 10–21. ICCA Journal, Vol. 12, No. 1.
-  Levy, D. and M. Newborn (1982).
All About Chess and Computers.
Springer Berlin, Heidelberg.
-  Lipton, Z. C., J. Berkowitz, and C. Elkan (2015).
A critical review of recurrent neural networks for sequence learning.
-  Luong, M.-T., H. Pham, and C. D. Manning (2015).
Effective approaches to attention-based neural machine translation.



Literatur V

-  Mohajerin, N. and S. L. Waslander (2017).
State initialization for recurrent neural network modeling of time-series data.
In 2017 International Joint Conference on Neural Networks (IJCNN), pp.
2330–2337.
-  Salehinejad, H., S. Sankar, J. Barfett, E. Colak, and S. Valaee (2018).
Recent advances in recurrent neural networks.
-  Schuster, M. and K. K. Paliwal (1997).
Bidirectional recurrent neural networks.
IEEE Trans. Signal Process. 45, 2673–2681.

Literatur VI

-  Segundo, P. S. and R. Galán (2005).
Bitboards: A new approach.
In M. H. Hamza (Ed.), *IASTED International Conference on Artificial Intelligence and Applications, part of the 23rd Multi-Conference on Applied Informatics, Innsbruck, Austria, February 14-16, 2005*, pp. 394–399. IASTED/ACTA Press.
-  Silver, D., T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis (2018).
A general reinforcement learning algorithm that masters chess, shogi, and go through self-play.
Volume 362, pp. 1140–1144.

Literatur VII

-  Sutskever, I., O. Vinyals, and Q. V. Le (2014).
Sequence to sequence learning with neural networks.
In *Proc. NIPS*, Montreal, CA.
-  Zang, H., Z. Yu, and X. Wan (2019, July).
Automated chess commentator powered by neural chess engine.
In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, pp. 5952–5961. Association for Computational Linguistics.