Projektantrag

Titel:	Entwicklung einer Schach-Engine auf Basis von Neuronalen Netzen
Autoren:	Frank Zimmerli, Angelo Casanova, Daniel Hamm
Ziele der Arbeit	• Entwicklung einer Schach-Engine, welche in der Lage ist, eine Partie Schach zu spielen. Die Ermittlung der Engine-Züge wird mithilfe eines Neuronalen Netzes umgesetzt. • Experimentieren mit verschiedenen Netz-Architekturen und Lernmethoden, um die vielversprechendsten Ansätze zu finden. • Ziel soll sein, dass die einzelnen Engines am Ende gegen sich selbst spielen und wir so die beste Engine ermitteln können. • Spass am Engineering: neben dem Training und der Inference des NN braucht es für eine lauffähige Engine auch Komponenten der klassischen Software-Entwicklung.
Kurzbeschreibung:	**Datenbeschaffung** In einem reinen “Reinforcement Learning” (RL) Ansatz könnte eine Engine immer wieder gegen sich selbst spielen. Es soll aber auch ein **klassisches NN** trainiert werden, und für dieses Training benötigen wir gespielte Partien mit entweder • (1) bewerteten Stellungen oder • (2) den nächsten besten Zügen Zu (1) gibt es diverse Datensätze: • https://www.kaggle.com/competitions/finding-elo/data - enthält 50.000 gespielte Partien; in jeder Partie ist jede Stellung gewertet. • https://database.lichess.org/ - enthält pro Monat über 90 Millionen gespielte Partien - gemäss eigenen Angaben von lichess enthalten 6% der Partien Stellungsbewertungen. • https://ccrl.chessdom.com/ccrl/4040/games.html - computer chess rating list mit gewerteten Partien zwischen Engines. Zu (2) wurden keine Datensätze gefunden. Es ist vermutlich möglich, mit einer lokal installierten Engine entsprechende Daten zu bekommen, dies müsste aber im Rahmen der Arbeit evaluiert und umgesetzt werden. Zwingend notwendig ist es aus heutiger Sicht nicht.

	Die Daten zu (1) liegen meist im PGN-Format (Portable Game Notation) vor, welches aus der **Reihenfolge der Züge** besteht. Eine Alternative ist das FEN-Format (Forsyth-Edwards Notation), welches eine **Stellung** beschreibt. Es gibt Python-Bibliotheken (bspw. *python-chess*), welche ein Format in das andere umwandeln können. **Entwicklung des NN** Zum Trainieren eines NN eignet sich das FEN-Format besser. Das NN bekommt als Input eine Stellung und muss die “evaluation” ausgeben (falls >0 steht Weiss besser, bei 0 ist es ausgeglichen und falls <0 steht Schwarz besser). Dies lässt sich sowohl als Regressions- wie auch als Klassifikations-Problem designen. Zu einer gegebenen Stellung könnte das NN nun sämtliche gültigen nächsten Stellungen bekommen (grob ~20-40) und die evaluations ausgeben. Es wird der Zug gewählt, der die beste evaluation ergibt. Im Zuge des Reinforcement-Learning Ansatzes kann man bspw. mit https://pettingzoo.farama.org/environments/classic/chess/ ein weiteres NN trainieren. Hierzu braucht es allerdings jeweils “bessere” Gegner; dies könnte entweder das oben beschriebene NN sein oder eine lokal auf dem Computer installierte Engine. Welcher Ansatz am besten funktioniert kann im Rahmen der Arbeit ermittelt werden. **Weitere notwendige Komponenten** Es braucht neben dem NN weitere Komponenten, die “traditionell” entwickelt werden müssen (unter Verwendung vorhandener OS-Libraries): ● Zu einer gegebenen Stellung werden gültige Züge ermittelt. Dies darf nicht einem NN überlassen werden, weil wir in 100% aller Fälle gültige Züge brauchen. ● Zu einer gegebenen Stellung wird ermittelt, ob das Spiel zu Ende ist (matt oder patt). ● Eine Art “Orchestrator”, der Engines gegeneinander spielen lässt. ● Datenaufbereitung: Umwandlung von PGN in FEN und Speichern der Daten in passenden Formaten.

	In der Arbeit steht das **Engineering** im Vordergrund. Daten, Engines und eine Menge Beispiele (https://github.com/thomasahle/fastchess, https://github.com/thomasahle/sunfish, weitere unter “Quellen”) gibt es. Die Arbeit soll **Spass machen** und einen Eindruck vermitteln, wie Schach-Engines auf Basis von Neuronalen Netzen funktionieren und was damit möglich ist.
Rahmenbedingungen:	• Nur öffentlich zugängliche Daten verwenden • Keine Eröffnungs- oder Endspielbibliotheken verwenden
Datenquelle	• Lichess Datenbank https://database.lichess.org/ • https://www.kaggle.com/competitions/finding-elo/data • https://ccrl.chessdom.com/ccrl/4040/games.html
Quellen	• Learning to Evaluate Chess Positions … https://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/ICPRAM_CHESS_DNN_2018.pdf • Chess Position Evaluation Using Radial Basis … https://www.hindawi.com/journals/complexity/2023/7143943/ • Dissecting Stockfish: In-Depth Look at a Chess Engine https://towardsdatascience.com/dissecting-stockfish-part-1-in-depth-look-at-a-chess-engine-7fddd1d83579 • Chess PettingZoo for Reinforcement Learning https://pettingzoo.farama.org/environments/classic/chess/ • Chess Playing Agents: A Problem Analysis https://static.aminer.org/pdf/PDF/000/226/325/genetically_programmed_strategies_for_chess_endgame.pdf
Risiken	• Schlechte Model-Performance, zu lange Trainingszeiten, hohe Komplexität (u.a. beim Reinforcement-Learning)