1 Einführung

Willkommen zu meiner Presentation mit dem Titel “Generierung von Schachkommentaren mittels maschinellem Lernen”.

Zu Beginn werde ich Ihnen einen Überblick darüber geben, wie die Themen Schach und künstliche Intelligenz zusammenhängen, damit die Notwendigkeit meiner Forschungsfrage deutlich wird. Dann werde ich kurz den Aufbau zur Lösung der Forschungsfrage vorstellen.

In den Abschnitten 2 und 3 wird der Aufbau näher erläutert, und Abschnitt 4 schließt die Präsentation mit einem Fazit ab.

2 Überblick

Schach gehört zu einem der am längsten erforschten Themen der Künstlichen Intelligenz. Das liegt vor allem daran, dass Pioniere der KI wie Alan Turing Schachspieler waren und das Schachspiel als ideales Modell für maschinelles Lernen betrachteten.

Damals wie heute wurde hauptsächlich auf dem Gebiet der Schachengines geforscht. Das sind Schachprogramme. Das Ziel der Forschung ist es, die Spielstärke zu optimieren. Heute liegt die Spielstärke der Engiens weit über der der besten Profispieler.

Ein Problem ist, dass dieses hohe Spielniveau zu einer Intransparenz von Zügen bzw. Zugfolgen führt. Das bedeutet, dass man professionelle Spieler oder Kommentatoren braucht, um die Ideen hinter den Zügen des Computers zu verstehen, und selbst deren Analysen sind nicht immer korrekt.

Meine Forschungsfrage "Wie kann maschinelles Lernen genutzt werden, um Kommentare zu Schachpartien zu generieren?" versucht, dieses Problem der Intransparenz von Zügen zu lösen, indem versucht wird, menschliche Kommentatoren durch virtuelle zu ersetzen. Außerdem können Kommentare nicht nur für Partien von Schachengines, sondern auch für menschliche Partien erstellt werden, so dass die eigene Spielstärke verbessert werden kann.

Um die Forschungsfrage zu lösen, wird sie dazu in zwei Teile aufgeteilt. Zum einen die Informationsbeschaffung und zum anderen die Kommentarerstellung. Die Informationsbeschaffung befasst damit *woher* man die Informationen bekommen die zum übersetzen nötig sind. Die Kommentargenerierung nimmt diese Informationen und beschäftigt dann damit *wie* und *was* für Kommentare erzeugt werden sollen.

Für den ersten Teil wird eine bereits erwähnte Schach Engine verwendet, für den zweiten Teil das sogenannte Encoder-Decoder-Modell.

3 Schach Engine

Wie bereits erwähnt, sind Engines Programme zum Schachspielen. Damit diese funktionieren, müssen sie eine Reihe von Anforderungen erfüllen. Diese sind die

- Brett Darstellung
- Zugsuche
- Positionsbewertung

Der Vorteil von Schachengines ist, dass diese Anforderungen genau die Informationen bereitstellen, die später benötigt werden.

3.1 Brett Darstellung

Für den Computer muss das Schachbrett und die Figuren in eine verarbeitbare Form gebracht werden.

Eine Standardimplementierung sind sogenannte Bitboards.

Jedes Feld kann entweder den Wert 0 oder 1 haben, wobei 0 bedeutet, dass sich keine Figur auf dem Feld befindet, und 1, dass ein Feld belegt ist.

Für jeden Figurentyp, Figurenfarbe und für die Rochade werden Bitboards erstellt. Für die Leute die nichts mit dem Begriff Rochade anfangen können, dabei handelt es sich um einen speziellen Spielzug, um den König in eine sichere Position zu bringen.

Die Anzahl der Bitboards können sich je nach Implementierung variieren. Hier sind es insgesamt 16.

Eine mögliche Implementierung sind 8×8 Arrays. Es gibt jedoch auch andere Implementierungen, z.B. mit einem eindimensionalen Array oder als Objekte.

Hier habe ich als Beispiel ein Bitbrett für die weißen Bauern mitgebracht. Durch das "Übereinanderlegen" der verschiedenen Bitboards können die Züge mit Hilfe logischer Operationen berechnet werden.

Ein Vorteil dieser Darstellung ist, dass sie als Input für neuronale Netze verwendet werden kann. Ich werde Ihnen gleich zeigen, wie das aussieht.

3.2 Zugsuche und Positionsbewertung

Schach ist bisher ein ungelöstes Spiel, d.h. es ist nicht bekannt, ob es eine Methode gibt, die immer zum Sieg oder, wenn sie von beiden Seiten angewendet wird, zum Unentschieden führt.

Deshalb basiert die Qualität der Suche nach Zügen und der Bewertung immer auf der Programmierung der Engine und Rechenleistung des Systems.
Bekannte Implementierungen, die keine Methoden des maschinellen Lernens enthalten, sind der MiniMax-Algorithmus bzw. die Erweiterung Alpha-Beta Pruning für die Zugsuche und eine handgeschriebene Bewertungsfunktion.

Die Implementierung, die ich euch zeigen werde, basiert auf maschinellem Lernen. Sie verwendet einen Monte-Carlo-Suchbaum für die Zugsuche und ein neuronales Netz für die Positionsbewertung.

Der Ansatz basiert auf der AlphaZero-Engine, die zum ersten Mal in der Lage war, die stärksten bekannten Engines zu schlagen, die ohne maschinelles Lernen arbeiten.

Wie genau der Ansatz funktioniert erkläre ich an einer Abbildung.

1. Wir haben eine Positon, diese liegt ich erzählt habe in der BitBoard Darstellung vor.
2. Die Codeierte Position wird als Input für ein neuronales Netzwerk verwendet, welche die Position bewertet. Als Output gibt dieser eine Bewertung (hier v) aus und Wahrscheinlichkeiten für folgende Züge (hier p_1 bis p_3 für drei Züge).
3. Der Monte Carlo Suchbaum, generiert dann alle möglichen legalen Züge.
4. Der ursprüngliche Position wird dann die Bewertung v zugeordnet und die berechneten Zügen bekommen die Zugwahrscheinlichkeiten p. Der Suchbaum wählt dann Pfade, teils zufällig, aus, simuliert Partien und bewerte all diese Positionen und Züge mit Hilfe des Neuronalen Netzwerks.
5. Für alle Pfade werden Gesamtstatistiken erstellt und der Pfad mit der besten Gesamtstatistik wird gespielt.

Damit das Neuronale Netzwerk zuverlässige Bewertungen erzeugt muss es trainiert werden. In traditionellen Engines wurde Trainingsdaten, aus Datenbanken, welche Profipartien speichern, benutzt. Die Daten können allerdings auch durch Selbstspiel generiert werden.

Dabei exitsiert Anfangs eine Engine besteht aus den Spielregeln, einem untrainiertes neuronales Netz und der MCTS-Algorithmus

Um das Netzwerk zu trainieren werden nun 4 Schritte immer wieder wiederholt.

1. Das Programm spielt gegen sich selbst, zeichnet jede Partie auf und weißt dann einer Partie verloren, unentschieden oder gewonnen zu.
2. Das Neuronale Netzwerk wird geklont und bekommt die Partien, zusammen mit dem Wert, um die Parameter anzupassen
3. Das Programm mit dem neuem Neuronale Netzwerk spielt dann gegen das alte Programm
4. Das Programm das gewinnt wird dann ausgewählt und es geht von vorne los

Durch diese Trainingsmethode konnte die damals beste Engine *Stockfish*, in nur 4 Stunden übertroffen werden.

Aber welche Informationen haben wir nun von der Engine bekommen.
Zum einen haben wir eine verarbeitbare Brettdarstellung. Außerdem können wir erkennen, ob eine Stellung gut oder schlecht ist, und wissen anhand der Zugwahrscheinlichkeiten, was gute und schlechte Züge sind. Darüber hinaus können wir Zugpfade nehmen, um die Absicht hinter einer Strategie in einer Partie zu bestimmen oder Züge zu vergleichen.