1 Einführung

- Willkommen zu meiner Presentation mit dem Titel “Generierung von Schachkommentaren mittels maschinellem Lernen”
- Zu Beginn Überblick darüber geben, wie Themen Schach und künstliche Intelligenz zusammenhängen damit Notwendigkeit meiner Forschungsfrage deutlich wird
- Dann Aufbau zur Lösung der Forschungsfrage vorstellen
- Abschnitten 2 und 3 Aufbau näher erläutert
- Schließt mit Fazit ab

2 Überblick

- Schach eines am längsten erforschten Themen der KI
- Pioniere der KI wie Alan Turing oder Claude Shannon waren selber Schachspieler
- Betrachteten Schachspiel als ideales Modell für maschinelles Lernen
- Forschung hauptsächlich auf Gebiet der Schachengines
- Schachengines sind Schachprogramme
- Ziel ist Optimierung der Spielstärke
- Heute liegt Spielstärke der Engiens weit über der der besten Profispieler
- Hohes Spielniveau führt zu Intransparenz von Zügen bzw. Zugfolgen
- Professionelle Spieler oder Kommentatoren werden benötigt, um Ideen hinter Zügen des Computers zu verstehen
- Auch deren Analysen sind oft nicht korrekt

- Forschungsfrage versucht, Problem der Intransparenz von Zügen bzw. Zugfolgen zu lösen
- Menschliche Kommentatoren durch virtuelle ersetzt werden
- Selbstverständlich nicht nur Partien von Schachengines, sondern auch menschliche Partien kommentieren
- Kann auch genutzt werden um eigene Spielstärke zu verbessern

- Um Forschungsfrage zu lösen, wird in zwei Teile aufgeteilt
- Zum einen Informationsbeschaffung
- Zum anderen Kommentarerstellung
- Informationsbeschaffung befasst sich *woher* Informationen kommen die zum übersetzen nötig sind
- Kommentargenerierung nimmt Informationen und beschäftigt *wie* und *was* für Kommentare erzeugt werden
- Für ersten Teil bereits erwähnte Schach Engine verwendet
- Für zweiten Teil Encoder-Decoder-Modell verwendet

3 Schach Engine

- Engines sind Schachprogramme
- Müssen Reihe von Anforderungen erfüllen
  - Brett Darstellung
  - Zugsuche
  - Positionsbewertung
- Anforderungen stellen Informationen bereit die später benötigt werden

3.1 Brett Darstellung

- Für Computer muss Schachbrett und Figuren in verarbeitbare Form gebracht werden
- Standardimplementierung Bitboards
- Jedes Feld kann Wert 0 oder 1 haben
  - 0: Figur belegt Feld nicht
  - 1: Figur belegt Feld
- Für jeden Figurentyp, Figurenfarbe und für die Rochade werden Bitboards erstellt.
- Rochade ist spezieller Spielzug, um König in sichere Position zu bringen
- Anzahl Bitboards kann je nach Implementierung variieren
- Hier insgesamt 16

- Mögliche Implementierung sind 8×8 Arrays
- Alterantiv eindimensionalen Array oder als Objekte
- Beispiel: Bitboard für weiße Bauern
- Durch "Übereinanderlegen" der Bitboards können Züge mit Hilfe logischer Operationen berechnet werden
- Vorteil ist, dass Darstellung als Input für neuronale Netze verwendet werden kann

3.2 Zugsuche und Positionsbewertung

- Schach ist bisher ungelöstes Spiel
- Nicht bekannt, ob es Methode gibt, die immer zum Sieg oder, wenn sie von beiden Seiten angewendet wird, zum Unentschieden führt
- Qualität Zugsuche und Positionsbewertung immer Basis der Programmierung und Rechenleistung des Systems
- Bekannte Implementierungen, ohne maschinelles Lernen:
  - MiniMax-Algorithmus bzw. Erweiterung Alpha-Beta Pruning für Zugsuche
  - handgeschriebene Bewertungsfunktion
- Zeigen Methode basierend auf maschinellem Lernen
  - Monte-Carlo-Suchbaum für Zugsuche
  - Neuronales Netz für Positionsbewertung
- Ansatz basiert auf AlphaZero-Engine
- War in der Lage, stärksten bekannten Engines zu schlagen, die ohne maschinelles Lernen arbeiten

Wie genau der Ansatz funktioniert erkläre ich an einer Abbildung.

1. Haben Positon gegeben, die in BitBoard Darstellung vorliegt
2. Codeierte Position wird als Input für neuronales Netzwerk verwendet, welche die Position bewertet. Als Output gibt es Bewertung (hier v) und Wahrscheinlichkeiten für folgende Züge (hier p_1 bis p_3 für drei Züge) aus.
3. Monte Carlo Suchbaum, generiert 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.

1. Zum einen verarbeitbare Brettdarstellung
2. Bewertung ob Stellung gut oder schlecht ist
3. Zugwahrscheinlichkeiten
4. Zugpfade durch MCTS um Absicht hinter Strategie zu bestimmen oder Züge zu vergleichen

4 Schachkommentator

Jetzt werden ich den Teil besprechen, der für die Erzeugung von Kommentaren verantwortlich ist.

4.1 Encoder-Decoder Modell

- Erzeugung von Kommentaren fällt in Bereich Sequenz-zu-Sequenz-Verarbeitung
- Input (Entsprechende Informationen) wird auf Output (d.h. menschliche Sprachen) dargestellt
- Länge von Input und Output können sich dabei unterscheiden
- Architektur die dies ermöglicht ist das Encoder-Decoder Modell
- Basiert auf speziellen neuronalen Netzwerkarchitektur, bidirektionalen Long-Short Term Memory-Architektur (LSTM)

- LSTMs sind Erweiterung von Rekurrenten Neuronalen Netzwerken
- RNNs neuronale Netze, zur Verarbeitung von Sequenzen
  - Generierte Outputs wieder mit neuem Input in Netzwerk geben
  - Zustand des Netzes repräsentiert Neuronen zu bestimmten Zeitpunkt
  - Durch zurückführen des Outputs kann sich Netzwerk an laufende Muster erinnern und entsprechend reagieren
- Bidirektionale RNNs sind Erweiterung von RNNs
- Eingaben nicht nur in positive Zeitrichtung, sondern auch in negative Zeitrichtung berücksichtigt
- Sowohl vorherigen als auch nachfolgenden Eingabedaten einbezogen
- Problem von RNNs: Bei langen Sequenzen hat interne Gedächtnis Probleme laufende Muster zu merken
- Problem nennt man "Problem des verschwindenden Gradienten"
- Lösung sind spezielle Neuronen, den Long-Short Term-Memory Neuronen
- Verfügen über spezielles Kurzzeitgedächtnis und Langzeitgedächtnis

Bidirektionale LSTMs sind also bidirektionale RNNs, die Long-Short Term Memory Neuronen verwenden.

- Encoder-Decoder Modell besteht aus zwei Teilen: Encoder, Decoder
- Encoder und Decoder sind bidirektionale LSTMs
- Encoder empfängt als Input Informationen von Schach-Engine
- Wandelt diese in andere Darstellung um
- Darstellung ist eine Zusammenfassung der Informationen
- Zusammenfassung mit Hilfe von Aufmerksamkeitsmechanismus erstellt
- Decoder verwendet Darstellung und erstellt entsprechende Kommentare

- Abbildung eines Encoder-Decoder Modells mitgebracht
- Blöcke stellen Netzwerk zu bestimmten Zeitpunkt dar
- Entsprechende Sequenz wird von links nach rechts, zeitlich aufsteigend, eingegeben
- Hidden Layers verarbeiten die Sequenz
- Fassen durch Aufmerksamkeitsmechanismus bestimmte Teile der Eingabe in Endzustand zusammen
- Zustand wird als Kontextvektor bezeichnet
- Kontextvektor wird zur Initialisierung der Neuronen des Decoders verwendet
- Generiung beginnt mit Startzeichen
- Ausgabe stellt erzeugten Kommentar dar
- Ausgabe wird als neue Eingabe in den Decoder gegeben
- Passiert solange bis Endzeichen erreicht wird

4.2 Generationsmodelle

- Um zu wissen zu was genau man Kommentare generieren soll, wurde 5 Kategorien festgelegt.

- Einziges Encoder-Decoder Modell reicht nicht aus
- Für jede Kategorien existiert ein Encoder-Decoder Modell
- Unterscheiden sich im Training
- Kategorien werden deshalb als Generationsmodelle bezeichnet
- Encoder die mehrere Züge bekommen als Multi-Move-Encoder bezeichnet
- Encoder die nur einen Zug bekommen als Single-Move-Encoder bezeichnet
- Merkmale die die Modelle erhalten, lassen sich als Funktion zusammenfassen
- Zug mit m (move) abgekürzt und Position mit b (board).

Nun da man weiß was für Merkmale man zum trainieren brauch, wurden Datensätze zum Trainieren, erstellt, welche basierend auf Kommentaren aus den fünf Kategorien aus einem Schachforum stammen.

5 Fazit

Schließlich noch das Fazit. Generell lässt sich sagen, dass die Forschungsfrage zur Generierung von Schachkommentaren zwar beantwortet werden konnte, sie aber den Menschen noch nicht ersetzen kann. Die generierten Kommentare konnten in verschiedenen Verfahren zur Messung der Genauigkeit gute Ergebnisse erzielen, menschliche Kommentatoren können allerdings noch deutlich tiefere Einblicke geben.

Dennoch bietet der vorgestellte Ansatz eine gute Grundlage für weitere Forschungen.