1 Lernziele

Allgemein Es soll eine Problemstellung aus dem Bereich der wissensbasierten Systeme eigenständig in einer Gruppe bearbeitet werden. Dazu ist eine Einarbeitung in das Themengebiet und das Erstellen eines Projektplans notwendig. Die entwickelten Lösungen sollen implementiert und in einem konkreten Anwendungsfall analysiert werden. Dabei ist das Problem mit Fachwissen unter Anwendung geeigneter Methoden und Verfahren entsprechend des aktuellen Stands der Technik zu lösen, das Projekt zu planen, durchzuführen, abzuschließen und zu dokumentieren.

Reinforcement Learning Lernen durch Belohnung, auch verstärkendes Lernen genannt, wird oft bei Agenten eingesetzt: Ein Agent interagiert mit seiner Umgebung und versucht ein definiertes Ziel zu erreichen, z.B. ein Spiel zu gewinnen. Der Agent kann die Umgebung wahrnehmen und Aktionen ausführen, die die Umgebung verändern. Er soll lernen die Aktionen so auszuführen, dass die erwartete Belohnung maximal wird. In diesem Projekt wollen wir uns das verstärkende Lernen an einem ganz konkreten Beispiel ansehen, zum Beispiel an einem Spiel wie 4-Gewinnt oder Othello. Wenn Sie ganz besonders herausgefordert werden möchten, können Sie sich auch an einem Spiel wie Schach oder Backgammon versuchen.

Hintergrund: Das von Gerald Tesauro 1992 am IBM's Thomas J. Watson Research Center entwickelte Programm TD-Gammon[1] gilt als eine der größten Erfolgsgeschichten des Reinforcement Learning. Es nutzt nur wenig Wissen über Backgammon und spielt dennoch extrem gut, nahe am Niveau der stärksten Großmeister der Welt.

Monte Carlo Tree Search Bei diesem Verfahren wird eine Suche in Baumstrukturen mit Monte-Carlo-Algorithmen kombiniert[2]. Dabei wird zunächst ein Knoten v im Suchbaum ausgewählt, anschließend ein noch nicht durchgeführter Zug auf diesen Zustand ausgeführt, das Spiel (z.B. zufällig) zu Ende gespielt und anhand des Ausgangs des Spiels wird der Knoten v und seine Vorgängerknoten neu bewertet. Dieser Vorgang wird sehr häufig wiederholt. Einfluss auf die Güte haben die Auswahl der Knoten und die Art, ein Spiel zu Ende zu spielen.

Hintergrund: Das Programm AlphaGo[3], dass auf Monte-Carlo-Tree-Search basiert, schlug im Jahr 2016 den damaligen Go-Weltmeister, was bis dahin noch keine künstliche Intelligenz geschafft hatte. Allerdings wurden dazu 1.920 CPUs und 280 GPUs genutzt.

Parallel Tree Search Bei Verfahren wie Minimax oder Monte-Carlo-Tree-Search müssen riesige Suchbäume analysiert bzw. traversiert werden. Hier können parallele Algorithmen helfen, die Laufzeiten wesentlich zu verkürzen bzw. eine deutlich bessere Spielstärke zu erreichen.

[1] `https://bkgm.com/articles/tesauro/tdl.html`
[2] `https://en.wikipedia.org/wiki/Monte_Carlo_tree_search`
[3] `https://en.wikipedia.org/wiki/AlphaGo`

2 Kombinatorische Spiele

4-Gewinnt Bei 4-Gewinnt[4] (englisch: Connect Four) wird auf einem senkrecht stehenden hohlen Spielbrett gespielt, bei dem die Spieler abwechselnd ihre Spielsteine in das Brett fallen lassen. Das Ziel ist es, als erster vier der eigenen Spielsteine in eine Reihe (horizontal, vertikal oder diagonal) zu bringen. Es gibt etwa $4.5 \cdot 10^{12}$ verschiedene, erreichbare Zustände[5] bei diesem Spiel.

Othello Das Spiel Othello[6], oft auch Reversi genannt, wird auf einem Schachbrett mit 64 Feldern gespielt. Die Spielsteine sind auf einer Seite weiß, auf der anderen Seite schwarz. Ein Spieler muss seinen Stein auf ein leeres Feld legen, das horizontal, vertikal oder diagonal an ein bereits belegtes Feld angrenzt. Wird ein Stein gelegt, werden alle gegnerischen Steine, die sich zwischen dem neuen Spielstein und einem bereits gelegten Stein der eigenen Farbe befinden, umgedreht. Spielzüge, die zu keinem Umdrehen von gegnerischen Steinen führen, sind nicht erlaubt. Das Ziel des Spiels ist es, am Ende eine möglichst große Anzahl von Steinen der eigenen Farbe auf dem Brett zu haben. Bei diesem Spiel gibt es etwa 10^{54} verschiedene, erreichbare Zustände.

3 Maschinelles Lernen

Reinforcement Learning In der Vorlesung betrachten wir das sehr einfache Spiel Tic-Tac-Toe. Die möglichen Zustände können dort in einer Hash-Tabelle S abgelegt werden, da es weniger als $3^9 = 19.683$ Zustände gibt. Jedem Zustand $s \in S$ wird eine Zahl $v(s) \in [0, 1]$ zugewiesen. Diese Zahl soll die Chance des Gewinns aus der Sicht eines der Spieler für den Zustand s angeben. Initial erhalten alle Zustände $s \in S$, die einen Gewinn darstellen, den Wert $v(s) = 1$, alle Zustände, die einen Verlust darstellen, erhalten den Wert $v(s) = 0$, alle anderen Zustände erhalten den Wert $v(s) = 1/2$, weil wir hier über den Ausgang des Spiels nichts wissen.

Nun müssen sehr viele Spiele gegen einen Gegner gespielt werden. Um bei einem solchen Spiel einen Zug auszuwählen, betrachten wir die Werte der möglichen Nachfolgezustände. Wir wählen dabei meistens den Zug, der in den Zustand s mit dem größten Wert $v(s)$ führt (greedy move). Gelegentlich wählen wir allerdings auch einen Zug zufällig aus (Exploration). Das Verhältnis von Greedy-Move und Exploration kann während des Trainings variieren.

Während des Trainings versuchen wir nach einem Sieg die Werte der Zustände $v(s)$ iterativ zu verbessern. Bei einem Greedy-Zug, nicht bei einem explorativen Zug, wird der Wert des alten Zustands $v(s_t)$ in Richtung des aktuellen Zustandswertes $v(s_{t+1})$ angepasst:

$$v(s_t) \;:=\; v(s_t) + \alpha \cdot [v(s_{t+1}) - v(s_t)] \;=\; (1 - \alpha) \cdot v(s_t) + \alpha \cdot v(s_{t+1})$$

Dabei ist $\alpha \in (0, 1)$ eine Lernrate, die im Laufe des Trainings geändert werden kann. Dieses Lernverfahren wird als *Temporal-Difference-Learning* bezeichnet, da es auf der Differenz der Werte zweier aufeinanderfolgender Zustände basiert.

Bei den Spielen 4-Gewinnt und Othello sind die Suchräume bereits so groß, dass die Wertefunktion v nicht mehr als Tabelle im Speicher gehalten werden kann und daher nicht mehr exakt ermittelt werden kann. Für so große Suchräume muss v durch adaptive Abbildungen, wie sie

[4] `https://de.wikipedia.org/wiki/Vier_gewinnt`

[5] Edelkamp, Stefan; Kissmann, Peter: Symbolic classification of general two-player games. In: Annual Conference on Artificial Intelligence, Springer, 2008

[6] `https://de.wikipedia.org/wiki/Computer-Othello`

z.B. neuronale Netze liefern, approximiert werden. Wie das Training solcher neuronalen Netze erfolgen kann, ist für das Spiel Tic-Tac-Toe in dem Artikel von C.J. Gatti und M. Embrechts[7] im Detail nachzulesen.

Monte Carlo Tree Search Da bei diesem Verfahren keine Funktion gelernt wird, sondern der Suchbaum tatsächlich aufgebaut wird, ergibt sich ein Problem, wenn der Baum so groß ist, dass er nicht mehr im Speicher gehalten werden kann. Wie groß der Baum wird, hängt vor allem von der Anzahl Wiederholungen ab, die durchgeführt werden müssen, um eine hohe Spielstärke zu erreichen. Dann müssen geeignete Verfahren angewendet werden, um den Suchraum zu verkleinern, z.B. indem nur vielversprechende Zweige betrachtet werden oder die Suche vorzeitig beendet wird. Auch die hohe Laufzeit der Suche ist ein Problem, daher müssen ggf. parallele Ansätze untersucht werden.

4 Projekt

In diesem Projekt soll untersucht werden, wie gut Reinforcement-Learning und Monte-Carlo-Tree-Search im Vergleich zur Minimax-Strategie sind, wobei Minimax in Kombination mit der Alpha-Beta-Beschneidung eingesetzt wird. Um eine hohe Spielstärke zu erreichen, muss bei Minimax eine große Suchtiefe eingestellt werden, was die Laufzeit stark erhöht. Bei Monte-Carlo-Tree-Search müssen viele Wiederholungen durchlaufen werden, was ebenfalls die Laufzeit erhöht und auch die Größe des Suchbaums enorm vergrößert. Es sollen daher parallele Algorithmen implementiert werden.

Fragestellungen, die untersucht werden sollen: Wie viele Trainingsschritte sind notwendig, um eine gute Spielstärke zu erreichen? Welche neuronalen Netzwerke eignen sich für die Approximation der Value-Funktion beim Reinforcement-Learning? Welche parallelen Algorithmen eignen sich zur Suche?

Dieses Projekt sollte von mindestens vier Studierenden bearbeitet werden.

5 Literatur

Ein frei verfügbares Buch zum Reinforcement-Learning von Richard S. Sutton und Andrew G. Barto finden Sie unter `http://incompleteideas.net/sutton/book/ebook/`

An der HAW Hamburg ist eine Bachelorarbeit von Herrn Konstantin Böhm[8] zu dem Thema „Maschinelles Lernen: Vergleich von Monte Carlo Tree Search und Reinforcement Learning am Beispiel eines Perfect Information Game“ verfügbar.

Zum Thema „Parallel Tree Search“gibt es eine Web-Seite[9] speziell zum Thema Schachprogramme. Außerdem findet man im Internet verschiedene Konferenz-Papiere[10] zum Download.

[7] `https://www.researchgate.net/publication/292851696_Reinforcement_Learning_with_Neural_Networks_Tricks_of_the_Trade`

[8] `https://reposit.haw-hamburg.de/bitstream/20.500.12738/7938/1/BA_Boehm.pdf`

[9] `https://www.chessprogramming.org/Parallel_Search`

[10] `https://dke.maastrichtuniversity.nl/m.winands/documents/multithreadedMCTS2.pdf`