Beating humans in complex board games

Seminar „Neueste Trends in Big Data Analytics“

Wintersemester 2017/18

Dozenten: Dr. Julian Kunkel, Jakob Lüttgau

B.A. Informatik

Universität Hamburg

Eike Nils Knopp
Osterstraße 68
20259 Hamburg
Matrikelnr.: 6809069
eike.knopp@studium.uni-hamburg.de

Inhaltsübersicht

1 Einleitung

Künstliche Intelligenzen finden immer mehr Anwendung in unserem Alltag. Ob in unseren Smartphones, Smarthome-Gegenständen wie Amazons Echo oder auch in der medizinischen Forschung. Doch dieser Vormarsch der künstlichen Intelligenzen ist erst kürzlich durch einige Durchbrüche in dem Gebiet der neuronalen Netzwerke ermöglicht worden. In dieser Arbeit wird die Geschichte der Entwicklung der künstlichen Intelligenz anhand von verschiedenen Brettspielen aufgezeigt. Es werden die Möglichkeiten und Grenzen ausgelotet und vor allem ihre Funktionsweisen erläutert. Abschließend werden in einem Ausblick die Einsatzmöglichkeiten dieser Form von künstlichen Intelligenzen diskutiert.

2 Historie

Die Geschichte der künstlichen Intelligenz in Brettspielen kann im Wesentlichen an verschiedenen Meilensteinen festgemacht werden. Diese Meilensteine werden im Folgenden kurz vorgestellt. Künstliche Intelligenzen in Brettspielen sind dabei insofern von Interesse, da sich dort Handlungen in klar definierten Räumen und Regeln bewegen und so sich mögliche Handlungen nicht im endlosen verlaufen. Dennoch kann von solchen Spiele-Szenarios abstrahiert werden und die künstliche Intelligenz (KI) so Anwendung in der realen Welt finden.

2.1 IBM DeepBlue

Der Supercomputer namens DeepBlue war der erste Computer, der je einen amtierenden menschlichen Weltmeister unter Wettkampfbedingungen in einem Schachspiel schlug. Im Jahre 1997 schlug der Supercomputer den damals amtierenden Schachweltmeister Garry Kasparov mit einem Endergebnis von 3½ zu 2½. Das Projekt begann als Dissertationsprojekt des Studenten Fenghsiung Hsu im Jahre 1985 und wurde später vom Unternehmen IBM übernommen. Die Komposition aus TreeSearch und einer von Schachprofis feinjustierten Evaluate-Funktion war zur damaligen Zeit revolutionär und konnte so, als erste Maschine, einen menschlichen Weltmeister schlagen.

2.2 IBM Watson

Der zweite Meilenstein wurde ebenfalls von der Firma IBM erreicht, mit der künstlichen Intelligenz, genannt Watson. Diese künstliche Intelligenz konnte 2011 gleich mehrere Champions im wissensbasierten Spiel Jeopardy schlagen. Das interessante an Watson ist dabei nicht, dass sie in der Lage war die Antworten auf die Fragen zu finden, da Watson unter anderem Zugriff auf ca. 200 Millionen strukturierte und unstrukturierte Dokumente hatte, inklusive der gesamten Datenbasis der online Enzyklopädie Wikipedia. Das interessante an Watson ist, dass die künstliche Intelligenz in der Lage war, Fragen – gestellt in natürlicher Sprache – zu verstehen und die Antworten basierend auf einem Confidence-Level abzuwägen. Dieses Confidence-Level ist dabei entscheidend: Watson weiß, was es nicht weiß. Ist Watson sich in einer Antwort nicht sicher (der Threshold des Confidence-Levels wurde also nicht erreicht), so wird lieber keine Antwort ausgegeben als eine falsche.

2.3 DeepMind AlphaGo

Der letzte große Meilenstein ist die von Googles Tochterfirma DeepMind entwickelte künstliche Intelligenz AlphaGo aus dem Jahre 2016. Im März dieses Jahres schlug sie den 18-fachen Go-Weltmeister Lee Sedol mit einem deutlich überlegenen Endergebnis von vier zu eins. Im Anschluss erhielt die künstliche Intelligenz als erste Software den höchsten Go Rang „Honory 9 Dan“, verliehen von der Korean Baduk Association (KBA).

3 Hauptteil

3.1 Relevanz von AlphaGo

AlphaGo setzte mit seiner Technik neue Maßstäbe für künstliche Intelligenzen und konnte das Spiel knacken, was lange Zeit als für Computer nicht lösbar galt: Das Jahrhunderte alte Go. Um zu verstehen, warum dies so war, muss man sich den bisherigen Ansatz für das Lösen solcher Brettspiele anschauen. Traditionell verwendet man einen so genannten Full-Game-Tree um eine Lösung zu finden. Dafür werden alle möglichen Kombinationen der Figuren auf dem Spielfeld aufgeschlüsselt und die aktuelle Brettkombination in diesem Baum gesucht. Von diesem aktuellen Knoten werden alle möglichen Pfade bis zu den Blättern traversiert. Auf diesen Pfaden werden alle möglichen Kombinationen von Zügen

durchgespielt, bis einer der Spieler letztendlich an den Blättern als Gewinner feststeht. Sind alle Pfade durchlaufen, wird der Zug ausgewählt, dessen Pfad die höchste Gewinnchance für den Spieler darstellt.

Verdeutlichen lässt sich dieses Verfahren am besten an dem bekannten Brettspiel Tic Tac Toe, welches anhand der folgenden Abbildungen kurz erklärt wird:

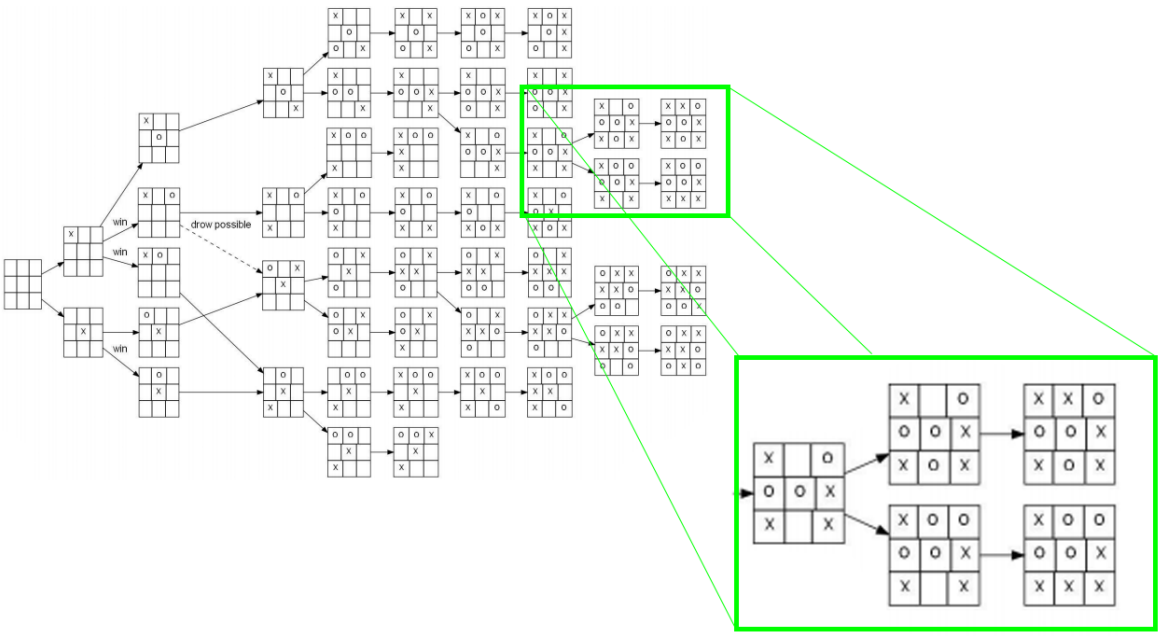

Abbildung 1: Wir betrachten einen Tic Tac Toe Game-Tree und legen dabei unsere Aufmerksamkeit auf den hervorgehobenen Bereich. Wir betrachten also die möglichen Züge des Spiels als Baum.[22]

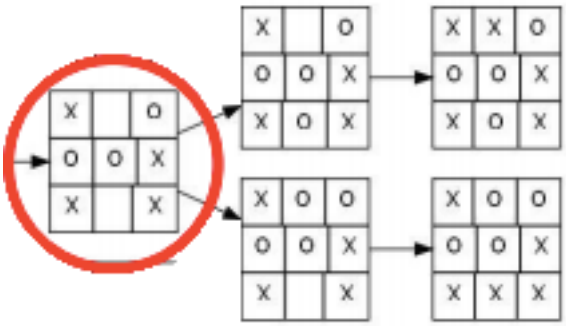

Abbildung 2: Der markierte Knoten ist unsere aktuelle Brettkombination. [22]

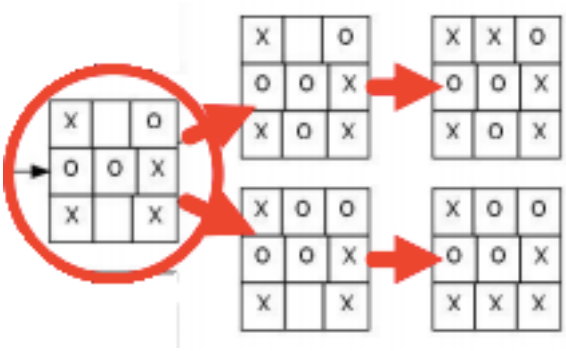

Abbildung 3: Wir durchsuchen nun alle möglichen Pfade ausgehende von diesem Knoten. Jeder mögliche Zug ausgehend vom aktuellen Knoten stellt einen eigenen Pfad dar. [22]

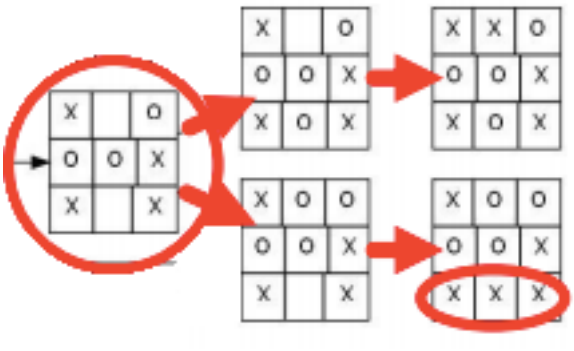

Abbildung 4: Angekommen an den Knoten überprüfen wir die möglichen Endergebnisse. Wir stellen fest, dass der obere Pfad zu einem Unentschieden führt, der unterer Pfad führt zu einem Sieg für den Kreuz-Spieler. [22]

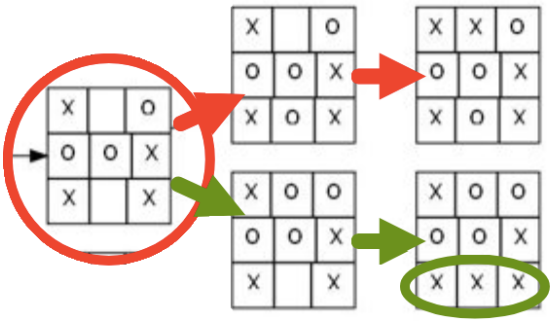

Abbildung 5: Da der untere Pfad also wesentlich sinnvoller für uns ist, wählen wir diesen Pfad. [22]

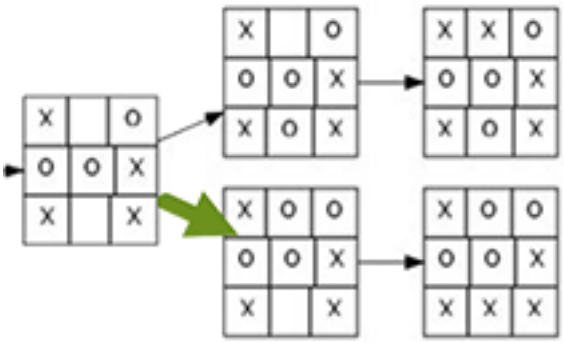

Abbildung 6: Nachdem wir uns für einen Pfad entschieden haben, wählen wir als aktuellen Zug jenen aus, der unseren ausgewählten Pfad initiiert. [22]

Vor dem Hintergrund dieses Verfahrens stellen wir nun also fest, dass der benötige Speicherplatz und die benötigte Rechenleistung in starker Abhängigkeit zur Komplexität des Spiels steht. Je komplexer ein Spiel ist, desto mehr mögliche Brettkombination gibt es. Je mehr Kombinationen es gibt, desto größer wird der Complete-Game-Tree. Da wir zur Lösung unseres nächsten Zugs den gesamten Baum durchsuchen müssen, steht die Größe des Baumes also im direkten Verhältnis zur benötigten Rechenleistung. Für kleine, einfache Spiele wie zum Beispiel Tic Tac Toe ist dieses Verfahren kein Problem, da das Tic Tac Toe Spielfeld nur aus 9 Feldern besteht, welche mit nur 2 Typen von Figuren befüllt werden. Eine simple obere Schranke für die Größe des Baumes ist daher 9!, also gerade einmal 362.880 Knoten (der erste Spieler hat 9 Möglichkeiten, seine Figur zu platzieren, der zweite 8 usw. bis das komplette Spielfeld befüllt ist). Für Game-Trees dieses Ausmaßes ist die benötigte Rechenleistung definitiv noch überschaubar. Jedoch sind Spiele, die diese Komplexität deutlich überschreiten, mit solch einem Ansatz nicht mehr zu bewältigen.

3.2 Wie spielte DeepBlue Schach?

Der Supercomputer DeepBlue war mit seiner Rechenleistung in der Lage bis zu 200 Millionen Schachpositionen zu berechnen. Auf Basis der Regeln des Schachspiels konnte DeepBlue, anhand des aktuellen Spielbretts, Züge berechnen. Jeder Zug wird danach von der speziellen Evaluate-Funktion evaluiert. Anhand der Ergebnisse kann der bestmögliche Zug ausgewählt werden.

Da die Größe des Game-Trees für Schach die Grenzen des Möglichen in Bezug auf die Reichenleistung überschreitet, wird der Baum, ausgehend von einer Brettkombination, nur so weit wie möglich durchsucht. Je weiter er durchsucht wird, desto genauer wird dabei

das Ergebnis. Durchschnittlich wird der Baum dabei 6 Züge weit durchsucht. Der nachfolgende Sub-Baum wird ersetzt durch die spezielle Evaluate-Funktion. Dabei wird die aktuelle Brettkombination als Input verwendet. Viele Faktoren werden dabei mit einbezogen: Die verfügbaren Figuren und ihre Positionen, die Position des Königs und seine Verteidigung, der Fortschritt des Spiels und zusätzliches Expertenwissen. Das Ergebnis der Funktion ist eine Zahl, die den Wert eines möglichen Zugs beschreibt. Nach der Prüfung von mehreren Zügen wird der Zug ausgewählt, der den höchsten Wert erhalten hat.

3.2.1 Die Evaluate-Funktion

Die Evaluate-Funktion besteht aus über 8.000 Teilen, dabei sind viele Teile extra auf bestimmte Positionen abgestimmt. Alleine das Opening-Book besteht aus über 4.000 Positionen aus über 700.000 Spielen von Grandmaster Spielern. Auch für den späteren Verlauf des Spiels hat DeepBlue Vorgaben für feste Positionen: Für viele Boardkombinationen von sechs oder weniger Figuren sind die nachfolgenden Züge genau festgelegt.
Wir stellen also fest, dass DeepBlue seine Lösung mit einem Verfahren erreicht, das eher an Brute-Force erinnert als an eine smarte künstliche Intelligenz, die durch "überlegen" und abschätzen eine Lösung findet. Da die Evaluate-Funktion zudem auch speziell auf DeepBlues Gegner Garry Kasparov und seinen Spielstil abgestimmt wurde, kommt es zu einem starken overfitting. Einen anderen Gegner, über den DeepBlue keine Informationen bezüglich seines Spielstils besitzt, hätte die künstliche Intelligenz möglicherweise nicht geschlagen. Zudem ist der Algorithmus nicht in der Lage zu lernen. Seine Stärke zieht das System aus der von Menschen geschaffenen Evaluate-Funktion. Im Umkehrschluss kann also das System nie stärker werden, als die Menschen, die es entwickelt haben. Auch wenn diese Menschen über großes Schachwissen verfügten kann dieser Ansatz nur schwer zu neuen Erkenntnissen führen. Auch kann diese künstliche Intelligenz auf keine andere Situation als auf Schach angewendet werden, da sie nicht generalisierbar ist. Sie ist nur darauf ausgelegt Schach zu spielen.

3.3 Die Unterschiede zwischen Schach und Go

Betrachten wir als nächstes die Unterschiede zwischen den Spielen Schach und Go, insbesondere die Unterschiede in der Komplexität. Bei Schach hat jeder Spieler 16 Figuren aus 6 verschiedenen Arten zur Verfügung. Dabei darf jede Art von Figur sich auf unterschiedliche Art auf dem Spielfeld bewegen. Das Spielfeld besteht dabei aus 64 einzelnen Feldern.

Bei Go hingegen besitzt jeder Spieler nur eine Art von Spielfiguren, welche auf einem Spielbrett mit 361 einzelnen Feldern gesetzt werden. Wir stellen also fest, dass Go wesentlich weniger Einschränkung in Bezug auf die Kombinationsmöglichkeiten hat. Das Spielfeld ist fast 6-mal so groß wie das Schachfeld und die Figuren können zu jeder Zeit auf jedes freie Feld auf dem Spielfeld platziert werden.
Ein Schach-Spiel kann auf 20 Arten eröffnet werden, ein Go-Spiel auf 361 Arten. Durchschnittlich kann man zu jedem Moment in einem Schach-Spiel aus 35 möglichen Zügen auswählen, bei Go sind es durchschnittlich 250. Auch die Spiellänge ist bei einem Go-Spiel durchschnittlich wesentlich länger als bei einem Schach-Spiel. 80 Züge beim Schachspiel stehen ungefähr 150 Züge bei Go gegenüber.
Diese Unterschiede manifestieren sich schlussendlich in der Komplexität der Spiele: Es gibt ungefähr 10^120 verschieden Möglichkeiten die Figuren auf einem Schachfeld im Laufe eins Spiels zu platzieren. Bei Go sind es überwältigende 10^170 Möglichkeiten. Um kurz einen Überblick zu schaffen, in welchen Größenordnungen wir uns hier bewegen: Es gibt ungefähr 10^80 Atome im Universum. Ein Complete-Game-Tree für Go besitzt also 10^170 Knoten.
Wir stellen also fest, dass Go viel zu komplex ist um von einem ähnlichen Ansatz wie bei IBMs DeepBlue gelöst zu werden, geschweige denn vom traditionellen Ansatz wie bei einem einfachen Spiel wie Tic Tac Toe.

3.4 Wie konnte AlphaGo Lee Sedol schlagen?

Um dieser Frage auf den Grund zu gehen schauen wir uns zuerst den Aufbau der künstlichen Intelligenz an. AlphaGo besteht aus zwei Hauptkomponenten: Einer Monte Carlo Tree Search und drei Deep Neural Networks. Die Komponenten können zwar auch einzeln verwendet werden, um eine Go spielende KI zu erzeugen. Doch einzeln kommen die Systeme nicht über ein Amateurniveau hinaus. Erst die Kombination aus beiden Systemen vermacht AlphaGo die Stärke die es benötigt, um mehrfache Weltmeister wie Lee Sedol zu schlagen. Betrachten wir zuerst die Komponenten im Einzelnen:

3.4.1 Monte Carlo Tree Search

Eine Monte Carlo Tree Search ist eine alternative Art, einen Game Tree zu durchsuchen. Im Gegensatz zum traditionellen Ansatz muss dafür nicht der gesamte Baum vorliegen, was enormen Speicherplatz und Rechenleistung spart. Ausgehend von einer aktuellen Brettkombination werden Spiele simuliert. Dabei werden zuerst zufällige Züge gewählt,

bis letztendlich ein Spieler gewinnt. Nach jeder Simulation werden die Werte gespeichert, welche Züge gewählt wurden und ob sie zu einem Sieg führten oder nicht. Jede Simulation zieht bei der Auswahl eines Zugs die Siegchancen desselben aus den vorigen Simulationen in Betracht. So werden aus den zu Beginn noch zufälligen Zügen nach und nach Züge, die eine sehr hohe Gewinnchance haben. Dabei gilt: Je mehr Simulationen durchgeführt werden, desto besser werden die Züge. Erhöht man die Anzahl an Simulationen ins unendliche, so konvergiert eine Monte Carlo Tree Search tatsächlich zum bestmöglichen Zug. Die Monte Carlo Tree Search lässt sich in folgende Einheiten aufgliedern:

1. Selection: Der Baum wird von der Wurzel bis zu einem Blatt durchlaufen. Dabei werden bei der Auswahl eines Knotens die jeweiligen Siegchancen des Knotens in Betracht gezogen.
2. Expansion: Ist das Blatt kein Sieg für einen der beiden Spieler, so wird ein oder mehrere neue Züge (Knoten) zufällig gewählt.
3. Simulation: Von diesen neuen Knoten wird eine Simulation gestartet. Das Spiel wird simuliert, bis einer der Spieler gewinnt (auch Rollout genannt).
4. Backpropagation: Ist die Simulation beendet, so wird das Ergebnis der Simulation zurück an den Knoten gegeben, der während der Expansion erzeugt wurde und von dort hoch bis zu der Wurzel des Baumes. War die Simulation ein Sieg, so wird dieser Pfad in zukünftigen Durchläufen starker gewichtet werden, als die Knoten eines Pfades, der letztendlich in einer Niederlage endet.

Systeme, welche sich nur auf Monte Carlo Tree Search für eine Go-KI verlassen und solche, die Monte Carlo Tree Search mit menschlichem Fachwissen kombinieren erreichen das Niveau eines starken Amateur Spieles. Ein Beispiel für solch ein System ist die Software Pachi, welche Züge ausschließlich mit ungefähr 100.000 Monte Carlo Tree Search Simulationen pro Zug auswählt.

3.4.2 Deep Neural Networks

AlphaGo besitzt nicht nur ein neuronales Netz, sondern gleich drei: Ein Value-Network und zwei Policy-Networks. Dabei übernimmt jedes Netz unterschiedliche Aufgaben.

3.4.3 Policy Network

Das Policy-Network hat die Aufgabe die Monte Carlo Tree Search bei der Auswahl neuer Züge zu leiten. Dabei wird das aktuelle Brett als Bild als Input verwendet. Jeder Zug, der durchlaufen wird, bekommt eine Wertigkeit von diesem Policy-Network zugewiesen. Anhand dieser Wertigkeit, kann so der bestmögliche Zug ausgewählt werden. Dieses Netz reduziert die Breite des Suchbaums, da so Züge, die nicht zielführend sind, direkt ausgeschlossen werden können und keine Ressourcen auf deren Berechnung verschwendet werden müssen.
Das Policy-Network gibt es in zwei Ausführungen: Als langsame aber dafür sehr präzise Variante, dem Heavy Rollout Policy Network, und als schnelle aber dafür weniger präzise Variante, dem Fast Rollout Policy Network.

3.4.4 Policy Network Training

Das Training des neuronalen Netzes besteht aus 2 Phasen. In der ersten Phase wird das Netz mittels Supervised Learning trainiert. In der zweiten Phase wird das bereits gelernte mittels Reinforcement Learning vertieft. Beim Supervised Learning werden ungefähr 30 Millionen Positionen aus Spielen zwischen menschlichen Spielern verwendet. Diese sind auf dem internationalen KGS Go Server gespeichert und für jedermann einsehbar. Dabei werden die Positionen als Trainings Set verwendet und der vom menschlichen Spieler tatsächlich ausgeführte Zug als Label des Validation Sets. Das Policy Network versucht nun also anhand einer Position den nächsten bestmöglichen Zug des Spiels vorauszusagen und überprüft sein Ergebnis mit dem Label aus dem Validation Set. Anhand der Korrektheit der Voraussage des Netzes passt das Netz seine Gewichtung innerhalb der Synapsen an und kann so seine Voraussagen immer weiter verbessern. Nachdem die 30 Millionen Trainingsdaten verarbeitet wurden, war das Heavy Policy Network in der Lage den nächsten Zug des menschlichen Spielers mit einer Genauigkeit von 57% vorherzusagen. Dabei benötigt das Netz durchschnittlich drei Millisekunden für die Berechnung des Zugs. Das Fast Rollout Policy Network kommt mit seiner wesentlich geringeren Anzahl an Neuronen zwar nur auf eine Genauigkeit von 24%, kann dafür diesen aber auch bereits innerhalb von durchschnittlich 2 Mikrosekunden berechnen und ist damit rund 1500-mal schneller als das Heavy Rollout Policy Network. In der zweiten Phase des Trainings wird die Genauigkeit des neuronalen Netzes mittels Reinforcemente Learning weiter verbessert. Hierbei spielt das Policy Network gegen zufällige Iterationen von sich selbst, um die Wahrscheinlichkeit eines Overfittings zu reduzieren. Overfitting bezeichnet dabei ein

Phänomen, bei dem sich das Netz zu stark an das Trainingsset annähert. Das Netz erzeugt dann zwar für das Trainingsset sehr gute Ergebnisse, doch tauscht man das Trainingsset gegen neue, unbekannte Daten aus, so erzielt das Netz wesentlich schlechtere Ergebnisse. Das Model des neuronalen Netzes hat sich zu stark an das Trainingsset angepasst um gute Ergebnisse zu erzielen, anstatt von den Daten zu abstrahieren und übergeordnete Strukturen zu erkennen. Durch stetig wechselnde Trainingssets unterschiedlicher Qualität lässt sich dieses Phänomen reduzieren. Deswegen führen Gegner aus zufälligen Iterationen des neuronalen Networks zu besseren Ergebnissen als wenn das Network nur gegen eine Version seiner selbst spielen würde. Bei diesen Spielen gegen sich selbst wird das Endergebnis des Spiels als Trainingssignal für den jeweiligen Zug genommen. Insgesamt spielte so das Netz mehr als 1,2 Millionen Partien gegen sich selbst.
So konnten Go-KIs welche nur auf dem Policy Network basieren nach dem Reinforcement Learning bereits eine Winrate von 80% gegen KIs verzeichnen, welche nur auf dem Policy Network nach dem Supervised Learning basieren. Gegenüber der Go-KI Pachi, welche nur auf der Monte Carlo Tree Search mit 100.000 Simulationen pro Zug basiert, konnte sogar eine Winrate von 85% erzielt werden.

3.4.5 Value Network

Das Value Network gibt Aufschluss über die "Qualität" eines aktuellen Bretts, also wie wahrscheinlich es ist, dass der schwarze Spieler, ausgehend vom aktuellen Brett, gewinnt. Dabei wird das Brett als Bild als Input für das neuronale Netz verwendet und gibt eine Zahl zurück, welche die Gewinnchance des schwarzen Spielers repräsentiert. Dieses Netz ist vergleichbar mit der Evaluate-Funktion des DeepBlue Schachsupercomputer von IBM, mit einem entscheidenden Unterschied: Das Value-Network wird trainiert und ist somit erlernt und nicht definiert. So wird zum einen kein direktes menschliches Wissen benötigt und zum anderen wird das Network nicht durch menschliches Wissen limitiert. Dieses Network reduziert die Tiefe des Suchbaums.

3.4.6 Value Network Training

Trainiert wurde das Value Network zuerst ebenfalls mit Positionen aus der Datenbank des KGS Go Servers. Doch schnell stellte man fest, dass es zu einem Overfitting dieser Daten kommt. Das Netz lernte den Ausgang von bestimmten Positionen auswendig, anstatt die Positionen zu generalisieren. Man ging dazu über, das Value Network mit 30 Millionen

Positionen aus Self-Play Data zu trainieren. Diese Positionen wurden aus Spielen zwischen zwei Varianten des Policy Networks entnommen, die sich in der Phase des Reinforcement Learning befanden. Dabei wurde aus jedem Spiel nur eine Position entnommen, um die Diversität der Daten zu erhöhen. Wenige, aufeinanderfolgende Züge innerhalb eines Spiels verändern meist den Ausgang eines Spiels nicht, solange man nicht zufällig einige wenige, kritische Züge betrachtet (die Wahrscheinlichkeit dafür ist sehr gering). So kann es passieren, dass auch hier das Netz den Ausgang des Spiels auswendig lernt, anstatt von den Positionen zu generalisieren. Um solch ein Overfitting zu vermeiden wird also aus jedem Spiel nur eine Position verwendet.
Das Value Network ist dabei wesentlich genauer als eine Monte Carlo Tree Search kombiniert mit dem Fast Rollout Policy Network und ähnlich genau wie eine Monte Carlo Tree Search kombiniert mit dem Heavy Rollout Policy Network, benötigt dabei allerdings fast 15.000-mal weniger Rechenleistung als das Heavy Rollout Policy Network.

Einzeln kommen die Komponenten von AlphaGo also auf das Niveau eines starken Amateurspielers. Erst die Kombination aus der Monte Carlo Tree Search und den neuronalen Netzen verleiht AlphaGo seine Stärke.

3.5 Kombination der Monte Carlo Tree Search und der neuronalen Netze

Die Stärken der verschiedenen neuronalen Netze werden in einem Monte Carlo Tree Search Algorithmus kombiniert. Jeder mögliche Zug, der sich von der Wurzel des Baumes aus ergibt, wird auf zwei Arten evaluiert. Einmal mit dem Value Network und einmal mit dem Fast Rollout Policy Network: Das Value Network evaluiert die Position nach dem Zug und gibt eine entsprechende Wertigkeit aus. Das Fast Rollout Policy Network führt, ausgehend von dem neu ausgewählten Zug, eine Simulation aus, bis das Spiel terminiert ist. Ob der Zug letztendlich zu einem Sieg oder einer Niederlage führt entscheidet über die Wertigkeit dieses Zugs. Die errechneten Werte werden mit den Werten der Kanten verrechnet. Aus diesem Verfahren ergibt sich letztendlich ein von dem System als bestmöglich eingestufter Zug. Doch betrachten wir die einzelnen Schritte etwas genauer:

3.5.1 Schritt 1: Selection

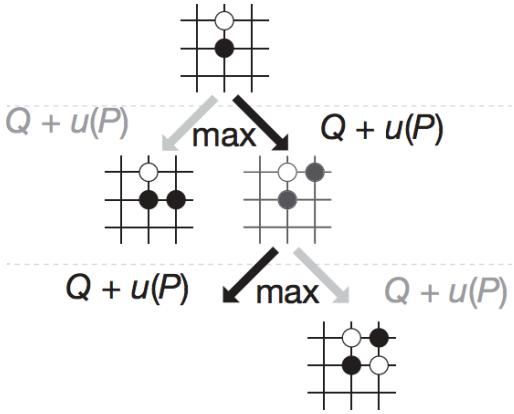

Abbildung 7: Jede Simulation durchläuft den Baum und wählt dabei immer die Kante aus, die den höchsten Wert hat. Der Wert der Kante ist dabei die Summe aus dem berechneten Wert aus vorherigen Simulationen Q und einem initialen probability Wert u(P). Dieser probability Wert wird durch das Heavy Rollout Policy Network bestimmt. Der Zug mit dem höchsten probability Wert spiegelt dabei also den Zug wieder, den ein menschlicher Spieler am wahrscheinlichsten wählen würde. Dabei wird u(P) durch die Anzahl geteilt, wie häufig diese Kante bereits ausgewählt wurde. u(P) wird mit der Zeit also immer geringer, was die Exploration von AlphaGo fördert und verhindert, dass sich das System zu früh auf wenige bestimmte Züge festlegt und andere, möglicherweise bessere Züge außer Acht lässt. [1]

3.5.2 Schritt 2: Expansion

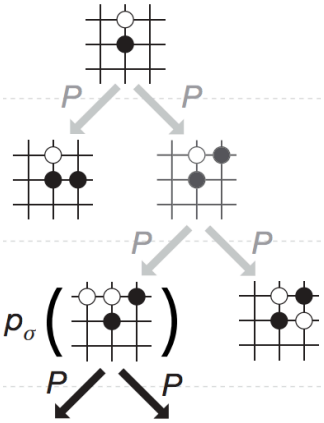

Abbildung 8: Erreichen wir ein Blatt, so wird ein neuer Zug erzeugt und vom Policy Network mit einem initialen probability Wert versehen. [1]

3.5.3 Schritt 3: Evaluation

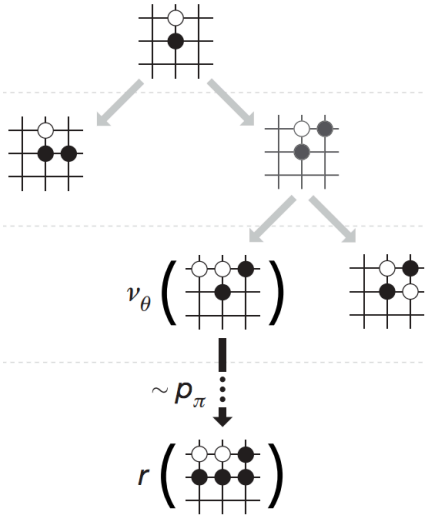

Abbildung 9: Der neu erzeugte Zug wird nun auf zwei Arten evaluiert. Zum einen wird die Position vom Value Network V0 bewertet. Zum anderen wird, ausgehend vom neuen Zug, eine Simulation gestartet, bei der das Spiel mittels des Fast Rollout Policy Network Pπ weitergespielt wird, bis das Ende erreicht wird. Der Ausgang des Spiels bestimmt die Wertigkeit des neu erzeugten Zuges. Anhand dieser beiden Werte wird die Wertigkeit Q des neuen Zugs berechnet. [1]

3.5.4 Schritt 4: Backup

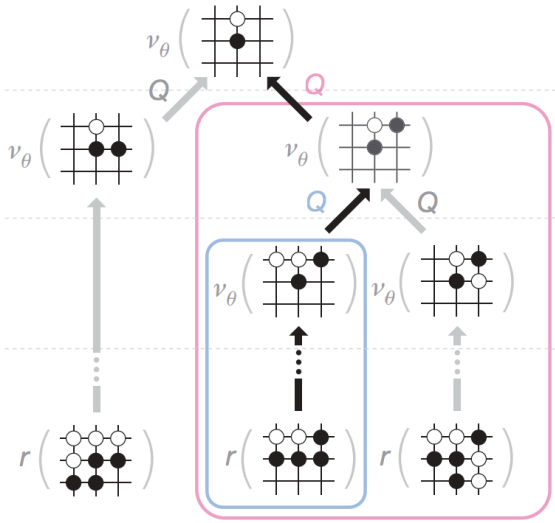

Abbildung 10: Die errechnete Wertigkeit Q des Zuges wird aktualisiert. Des Weiteren werden alle Werte Q der Züge aktualisiert, die in dem Pfad von der Wurzel bis zum neuen Blatt enthalten waren. Führte der Zug zu einem Sieg erhöht sich der Wert, führte der Zug zu einer Niederlage so verringert sich der Wert. Auch wird die aktualisiert, wie häufig eine Kante ausgewählt wurde und so auch der Wert u(P) angepasst. [1]

Nach n Durchläufen dieses Verfahrens ergibt sich also ein Zug, der eine höhere Wertigkeit aufweist als alle anderen Züge. So findet AlphaGo seinen nächsten Zug und der Gegenspieler ist an der Reihe.
Durch diese Kombination aus Monte Carlo Tree Search und der verschiedenen Neuronalen Netze erlangte AlphaGo seine bekannte Stärke, mit der das System auch den 18-fachen Go-Weltmeister Lee Sedol schlagen konnte, und überwindet so die Schranken der Speicher- und Rechenkapazität, die eine solch starke Go-KI für lange Zeit verhinderten: Die neuronalen Netze reduzieren die Breite und Tiefe des Suchbaumes in solch einem Maße, dass AlphaGo durchschnittlich nur 5 Sekunden pro Zug benötigt.

4 Ausblick: Wie kann das neue Wissen weiter genutzt werden?

Anwendung finden die neunen Erkenntnisse aus der künstlichen Intelligenz überwiegend in der medizinischen Forschung. Besonders im Themengebiet der Erforschung der Faltung von Proteinen eröffnet die Problemlösung von AlphaGo neue Pfade. Dabei wird das System zum Erkennen und Verstehen von fehlgeformten Proteinen verwendet um neue Erkenntnisse und Behandlungsformen für beispielsweise Krankheiten wie Alzheimer zu finden. Auch findet die Anfangs erwähnte künstliche Intelligenz Watson von IBM bereits Anwendung Krebsforschung. Watson for Oncology kann eine Vielzahl von Krebsarten anhand von Röntgenbildern erkennen und in Kombination mit einem großen Pool an Krankenakten und Patientendaten speziell auf den Patienten zugeschnittene Behandlungspläne erstellen.

Ein weiterer interessanter Punkt ist die auf AlphaGo basierende Weiterentwicklung AlphaGo Zero. Im Gegensatz zu AlphaGo verwendet AlphaGo Zero ein sogenanntes "Tabula rasa" learning. Dem System wurden nur die Regeln des Spiels Go beigebracht. Ohne jedes weitere menschliche Wissen spielt AlphaGo Zero gegen Iteration von sich selbst und entwickelt so eigenständig eine Spielweise und eigene Taktiken, völlig losgelöst von bereits bekannten Taktiken menschlicher Spieler. So ist es möglich, ohne menschliches Vorwissen neue Erkenntnisse zu gewinnen. Mittels dieses Verfahrens erreichte AlphaGo Zero bereits nach 3 Tagen die gleiche Stärke wie die Version von AlphaGo, die den Weltmeister

Lee Sedol schlug. Nach 40 Tagen übertrumpfte AlphaGo Zero jegliche Versionen von AlphaGo und wird der beste Go Spieler der Welt. All dies ohne menschliches Vorwissen und ausschließlich durch Self-Play.
Dies eröffnet zudem neue Möglichkeiten für "general purpose" KIs, die auf viele verschiedene Anwendungsgebiete anwendbar sind und nicht nur auf einzelne Tätigkeiten.

Quellen

Journals:

[1] Mastering the Game of Go with Deep Neural Networks and Tree Search
Google, Google DeepMind | Pusblished in Nature 529, Januar 2016

[2] Mastering the game of Go without human knowledge
Google, Google DeepMind | Pusblished in Nature 550, April 2017

Online-Quellen:

[3] https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/
[4] https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/
[5] http://ccg.doc.gold.ac.uk/research-mcts/
[6] https://machinelearnings.co/understanding-alphago-948607845bb1
[7] https://deepmind.com/blog/alphago-zero-learning-scratch/
[8] https://deepmind.com/research/alphago/
[9] https://blog.google/topics/machine-learning/alphago-machine-learning- game-go/
[10] http://ccg.doc.gold.ac.uk/research-mcts/
[11] https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/
[12] https://en.wikipedia.org/wiki/Monte_Carlo_tree_search
[13] http://www.businessinsider.com/how-ibm-watson-is-transforming-healthcare-2015-7?IR=T
[14] https://sciencebasedmedicine.org/tag/watson/
[15] https://www.newscientist.com/article/2079871-im-in-shock-how-an-ai-beat-the-worlds-best-human-at-go/

[16] http://www.businessinsider.com/r-ibms-watson-to-guide-cancer-therapies-at-14-centers-2015-5?IR=T
[17] https://www.engadget.com/2017/06/01/ibm-watson-cancer-treatment- plans/
[18] http://www.telegraph.co.uk/science/2017/10/18/alphago-zero-google-deep-mind-supercomputer-learns-3000-years/
[19] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4828734/
[20] https://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov
[21] https://en.wikipedia.org/wiki/Watson_(computer)
[22] https://commons.wikimedia.org/wiki/File:Tic-tac-toe-full-game-tree-x-rational.jpg