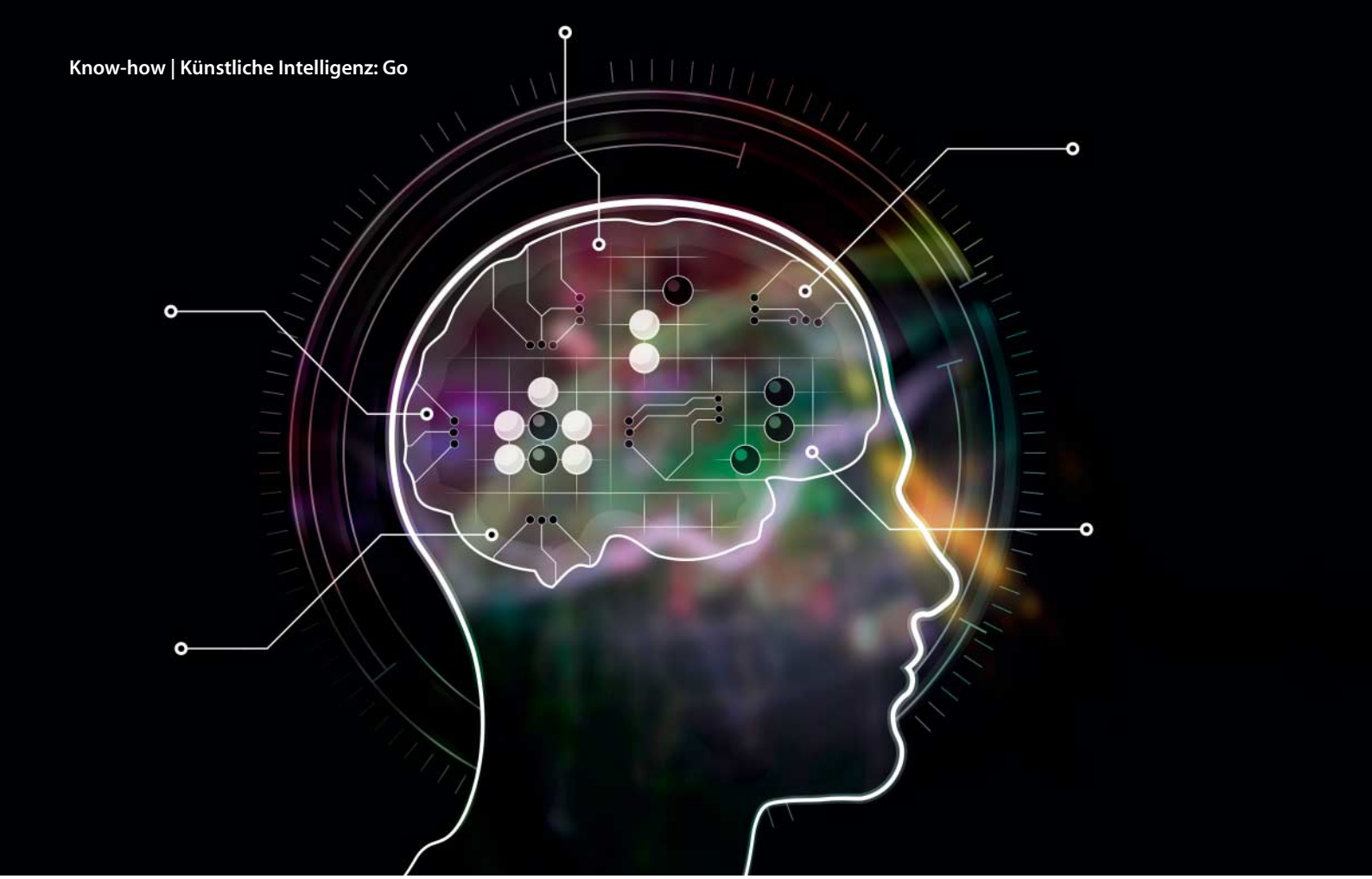


Harald Bögeholz

Mysteriöse Tiefe

Wie Google-KI den Menschen im Go schlagen will

Am 9. März ist Showdown: Die Google-Tochter DeepMind fordert den weltstärksten Go-Spieler der letzten zehn Jahre heraus, gegen ihre künstliche Intelligenz AlphaGo anzutreten. Unter den Strategiespielen gilt Go als die letzte Bastion menschlicher Intelligenz – und sie könnte fallen.

Was dem Westler das Schach, ist dem Asiaten das Go: Jeder kennt es irgendwie, in der Zeitung gibt es eine Go-Ecke, aber natürlich spielen es nur wenige auf Meisterniveau. In China, Japan und Korea gibt es Profi-Spieler, die ihr Leben lang Go studieren und ihren Lebensunterhalt damit verdienen; Titelkämpfe werden live im TV übertragen.

Für Computer ist Go ausgesprochen schwierig; es gilt unter den Strategiespielen als die letzte große Herausforderung an die künstliche Intelligenz. Im Schach hat bereits 1997 der IBM-Rechner Deep Blue den Schachweltmeister Kasparov geschlagen. Die Go-Programme dieser Zeit spielten dagegen erbärmlich und konnten selbst mittelmäßigen Amateuren nicht das Wasser reichen.

Das hat sich nun geändert. Zunächst gab es 2006 einen Durchbruch mit der Erfindung der Monte-Carlo-Suche, seitdem eine kontinuierliche Verbesserung der Go-Programme bis auf das Niveau starker Amateure und dann im Januar 2016 einen großen Knall: Die Google-Tochter DeepMind hat eine künstliche Intelligenz geschaffen, die erstmals einen menschlichen Profi im Go geschlagen hat.

Es gibt verschiedene Faktoren, die Go so schwierig machen. Zum einen ist es die schiere Größe des Suchbaums. Für den ersten Zug von Schwarz gibt es 361 Möglichkeiten, für die Antwort von Weiß 360 … grob gesagt im Mittel 200 Möglichkeiten pro Zug. Eine Partie kann schon mal 250 Züge oder mehr dauern; selbst wenn man die Tiefe des Baums konservativ auf 150 schätzt, ist seine Größe mit ungefähr 200^{150} ungefähr unendlich.

Beschneiden und bewerten

Den Baum vollständig zu durchsuchen, ist also völlig unmöglich – aber das ist es auch schon beim Schach mit seinem vielleicht 35^{80} Knoten großen Suchbaum. Wenn man den Baum schon nicht vollständig abgrasen kann, ist es eine Idee, nur eine gewisse Zahl von Zügen vorauszurechnen und die entstehende Stellung dann zu bewerten. Dann hat kein Spieler klar gewonnen, aber die Bewertungsfunktion kann doch eine Einschätzung in Form einer Gewinnwahrscheinlichkeit liefern.

Beim Schach gibt es naheliegende Kriterien für solch eine Bewertungsfunktion, allen voran der Materialwert: Eine Dame ist mehr wert als ein Turm oder gar ein Bauer. Beweglichkeit von Figuren, Einfluss etc., da kommen schon ein paar Ideen zusammen.

Und beim Go? Schaut man sich ein Go-Brett nach etwa 20 Zügen an, so kann man fast noch gar nichts sagen. Klar, der Profi wird es erkennen, wenn ein Amateur völligen Mist aufs Brett gelegt hat, aber wenn zwei gleich starke Spieler eine Partie spielen,

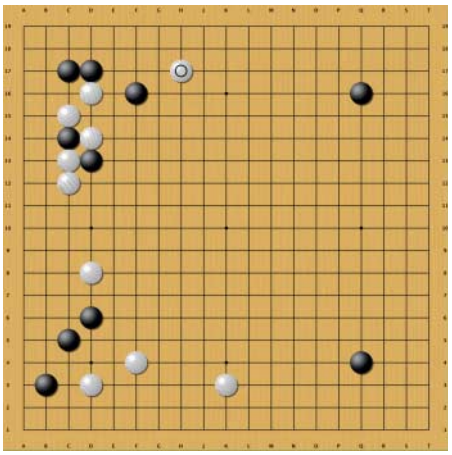
wird nach 20 Zügen noch alles offen sein. Jeder hat gleich viele Steine auf dem Brett und jeder Stein ist gleich viel wert. Beziehungsweise kann man einem Stein überhaupt nicht ansehen, ob er am Ende der Partie als tragender Teil einer wichtigen Mauer dasteht oder ob er als unbedeutender Gefangener vom Brett geht. Unscharfe Kriterien wie Stärke, Schwäche und Einfluss können sich im Lauf der Partie ins Gegenteil verkehren, und sie tun es regelmäßig. Material ist nicht unbedingt entscheidend, man kann immer noch gewinnen, wenn man ein paar Steine verloren hat.

In Ermangelung einer guten Bewertungsfunktion versuchten sich die Go-Programmierer an allerlei Heuristiken, legten Datenbanken lokaler Muster an, die gute nächste Züge vorschlugen, Joseki-Bibliotheken (bewährte Zugfolgen in der Eröffnung) und dergleichen. Doch den großen Durchbruch brachte der Zufall: eine Reihe von Verfahren, die auf dem Auswürfeln von Partien beruhen, heute unter dem Namen Monte Carlo Tree Search (MCTS) zusammengefasst.

Affen am Zug

Die zentrale Idee dabei ist: Wenn man von einer Stellung aus die Partie mit zufälligen Zügen zu Ende spielt, ist das Ergebnis zufällig. Tut man das aber sehr oft und gewinnt dabei Schwarz öfter als Weiß, dann wird die Stellung wohl besser für Schwarz sein. Diesem Auswürfeln verdankt das Verfahren das „Monte Carlo“ im Namen. Statt völlig zufälliger Züge kann man auch eine Strategie nehmen (englisch „policy“), die ausgehend von einer Stellung eine Wahrscheinlichkeitsverteilung für den nächsten Zug erzeugt. Wenn beide Spieler mit derselben Strategie – immer noch zufallsgesteuert – sehr oft zu Ende spielen und einer häufiger gewinnt als der andere, dann kann die Ausgangsstellung für diesen Spieler als besser gelten.

Ein MCTS-Algorithmus spielt nun eine große Menge von Zufallspartien („Playouts“) und baut dabei im Speicher einen Teil des Spielbaums auf, dessen Wurzel die aktuelle Stellung ist. Jeder Knoten entspricht einer Stellung. Er speichert für jeden möglichen



Partie: Fan Hui (2p, Schwarz) gegen AlphaGo (Weiß)

Nach 20 Zügen ist der Ausgang der Partie (siehe Kasten) unmöglich vorherzusehen.

Zug Statistiken darüber, wie oft die Simulation diesen Zug gewählt hat und wie viele der Partien gewonnen wurden. Solange die Simulation innerhalb des im Speicher gehaltenen Spielbaums verläuft, tragen die bisher

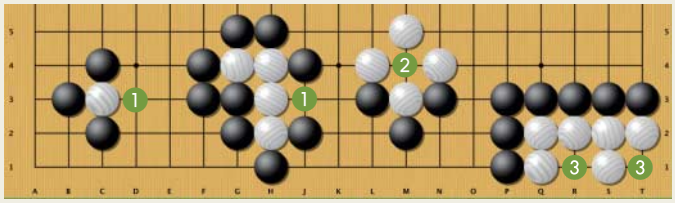
Die Go-Regeln

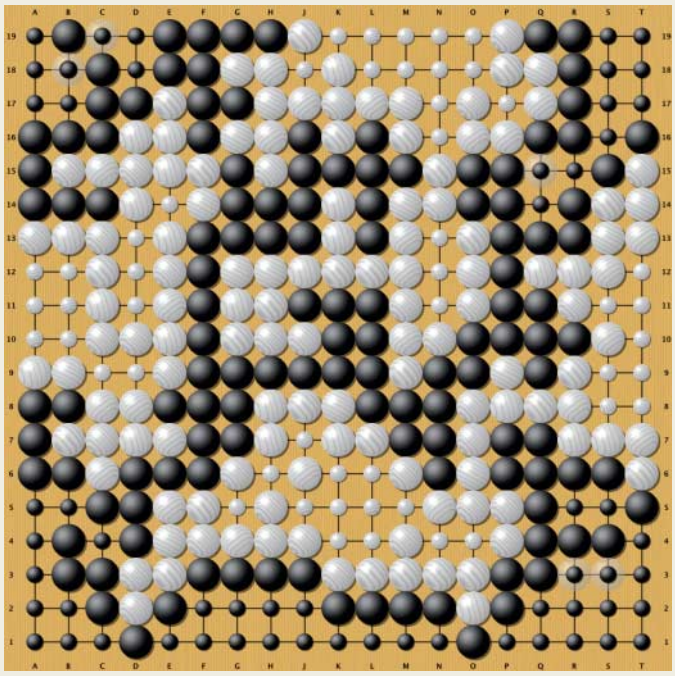
Go wird auf einem Brett mit 19 × 19 Linien gespielt; Anfänger beginnen auf kleineren Brettern mit 9 × 9 oder 13 × 13. Die Spieler setzen abwechselnd Steine auf die Kreuzungspunkte, nicht in die Kästchen, Schwarz beginnt. Einmal gesetzte Steine werden nicht bewegt, können aber geschlagen werden. Ziel des Spieles ist es, möglichst viele freie Kreuzungspunkte mit seinen eigenen Steinen zu umschließen.

Jede zusammenhängende Kette von Steinen und auch jeder Einzelstein muss mindestens einen angrenzenden freien Punkt besitzen, eine sogenannte Freiheit. Zusammenhang ist dabei entlang der Linien gemeint, nicht diagonal. Besetzt man mit einem eigenen Stein die letzte Freiheit einer gegnerischen Kette (1), so wird diese geschlagen und vom Brett genommen. Selbstmord ist verboten: Man darf nicht die letzte Freiheit einer eigenen Kette besetzen. Ausnahme: Man schlägt dadurch gegnerische Steine und schafft so Freiheit für den gesetzten Stein (2). Eine Regel namens Ko verbietet Endlosschleifen, in denen die Spieler durch Hin- und Herschlagen immer dieselben Stellungen wiederholen.

Aus diesen Regeln folgt, dass Formationen mit zwei nicht zusammenhängenden Löchern („Augen“) (3) nicht geschlagen werden können. Denn dazu müsste man beide Freiheiten besetzen, jeder dieser beiden Züge wäre aber Selbstmord und somit verboten. Langfristig, wenn das Brett sich füllt, überleben also nur Formationen mit mindestens zwei Augen.

Jeder Spieler darf passen; die Partie endet, wenn beide nacheinander passen. Dann wird gezählt. Dazu nehmen die Spieler zunächst alle gegnerischen Steine als Gefangene vom Brett, die keine zwei Augen haben (wenn Anfänger sich in diesem Punkt nicht einig sind, müssen sie es ausspielen). Die Punktzahl eines Spielers ergibt sich jetzt als Anzahl der freien Punkte, die er mit eigenen Steinen umschlossen hat, plus der Anzahl der Gefangenen. Es zählen also nur die Löcher, nicht die gelegten Steine. Weiß erhält dann noch 7,5 Punkte gutgeschrieben, um den Anzugsvorteil von Schwarz auszugleichen. Der Spieler mit der höheren Punktzahl gewinnt; die Höhe der Differenz ist unerheblich.

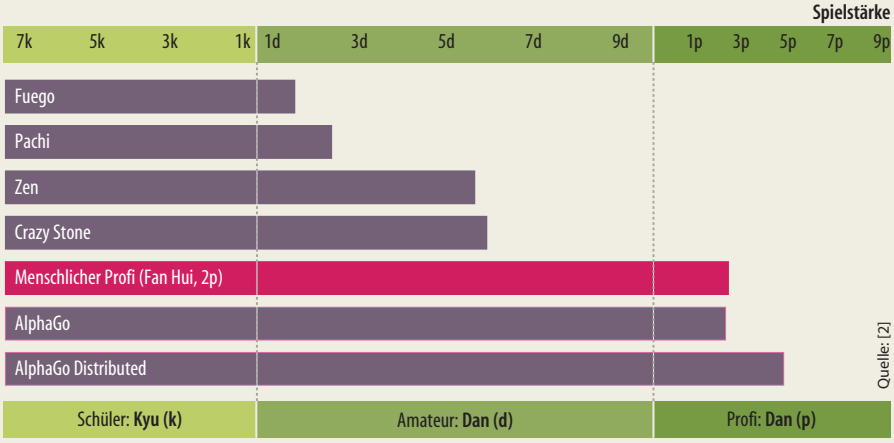




Endstand der Partie Fan Hui (2p, Schwarz) gegen AlphaGo (Weiß). Als Punkte zählen die freien Felder (kleine Punkte). Weiß gewinnt mit 2,5 Punkten Vorsprung.

Spielstärken in Go – Mensch vs. Maschine

Im Go werden die Spielstärken ähnlich wie bei den Kampfsportarten in Schülergraden (Kyu, kleinere Zahlen besser) und Meistergraden (Dan, größere Zahlen besser) angegeben. Die Grafik zeigt die Stärke verschiedener Go-Programme im Vergleich zum Profi-Spieler Fan Hui (2p).



ermittelten Statistiken zur Wahl des nächsten Zuges bei. Nach Verlassen des Baums simuliert der Algorithmus die Partie einfach nach einer festen Strategie zu Ende und findet heraus, wer gewinnt. Mit dieser Information aktualisiert er dann auf dem Weg zurück durch den Baum hinauf die Statistiken. Der Baum wächst dabei im Speicher nach und nach an den vielversprechendsten Stellen: Wo viele Simulationen vorbeikommen, lohnt es sich, Statistiken über die Folgezüge differenzierter zu speichern.

Ein Go-Programm wiederholt nun einfach diese Berechnungen so lange, bis die zur Verfügung stehende Bedenkzeit aufgebraucht ist, und wählt dann den Zug, den laut Statistik die meisten Simulationen gemacht haben. Das ist die Grundidee, und seit etwa 2006 haben alle namhaften Go-Programme in irgendeiner Form MCTS implementiert.

Neuronale Netze

Dann kamen die neuronalen Netze. Im November 2015 horchte die Computer-Go-Szene auf, als Forscher von Facebook ein neuronales Netz präsentierten, das auf dem Go-Server KGS aus dem Stand auf Amateur-1-Dan-Niveau spielte [1]. Sie experimentierten dann damit, das Ganze mit MCTS anzureichern und veröffentlichten im Januar eine überarbeitete Version ihres Papers. Hierin konnten sie schon eine Spielstärke von 3 Dan auf KGS vermelden – durchaus eine beachtliche Steigerung.

Doch sie ahnten wohl, dass Google DeepMind sie längst in den Schatten gestellt hatte. Dort muss es allen Beteiligten verdammt schwergefallen sein, bis zur Veröffentlichung des Artikels [2] im renommierten Wissenschaftsmagazin Nature dichtzuhalten.

Und mit diesem, am 28. Januar, platzte dann die Bombe: Schon im Oktober 2015 hatte AlphaGo, die Go-KI von Google DeepMind, den Profi-Spieler Fan Hui (2p) in einem formalen Match 5:0 geschlagen. Fan Hui ist ein chinesischer Profi, der in Europa lebt und dreifacher Europameister ist. Damit war erwiesen, was interne Tests gegen die stärksten anderen Go-Programme schon ergeben hatten: AlphaGo spielt erheblich stärker als alle anderen Programme und hat definitiv Profi-Niveau erreicht.

AlphaGo zieht alle Register: Deep Convolutional Networks, Supervised Learning, Reinforcement Learning … gleich zwei neuronale Netze spielen eine Rolle. Diese wurden mit MCTS-Ideen zu einem Algorithmus kombiniert, den die Macher APV-MCTS nennen: Asynchronous Policy and Value MCTS. Und nicht zuletzt kam eine gehörige Portion Rechenleistung zum Einsatz. Für die Partien gegen Fan Hui lief eine verteilte Version von AlphaGo auf 1202 CPUs und 176 GPUs.

Strategie und Stellungsbewertung

Die Eckpfeiler von AlphaGo sind zwei neuronale Netze namens Policy Network und Value Network. Ein Policy Network macht ausgehend von einer Stellung Vorschläge für gute Züge, und zwar in Form einer Wahrscheinlichkeitsverteilung über alle möglichen Züge. In seiner ersten Version wurde es mit von Menschen gespielten Partien trainiert, sagt also voraus, was für einen Zug ein starker Mensch in der betreffenden Stellung machen würde.

Ein Value Network soll eine Bewertungsfunktion liefern: Wie gut ist eine bestimmte Stellung, ausgedrückt als Wahrscheinlichkeit, aus dieser Stellung heraus die Partie zu gewinnen?

Um diese neuronalen Netze zu trainieren, gingen die Forscher in mehreren Schritten vor. Ausgangsbasis waren 30 Millionen Stellungen aus Partien zwischen starken Spielern (6 bis 9 Dan) auf dem Go-Server KGS. In die Eingabeschicht flossen dabei nicht nur Ebenen mit der nackten Stellung ein (Schwarz, Weiß, leer), sondern weiteres, auf einfache Weise errechenbares Go-Grundwissen, zum Beispiel:

- Wie lange liegt ein Stein schon da?
- Wie viele Freiheiten hat eine Kette?
- Wie viele gegnerische Steine würde ein Zug an dieser Stelle schlagen?
- Wie viele Freiheiten würde ein Zug an dieser Stelle schaffen?

Allein der Lernaufwand war kein Pappenstiel: 50 GPUs ackerten daran drei Wochen lang. Das Ergebnis nennen die Forscher das SL Policy Network, SL für Supervised Learning. In einem weiteren Schritt ließen sie dieses Netz dann in leicht unterschiedlichen Versionen gegen sich selbst spielen und daraus weiter lernen (Reinforcement Learning). Dies optimierte das Netz weg von der Aufgabe „menschlichen Zug vorhersagen“ hin zum eigentlichen Spielziel, gewinnen. Nach 50 weiteren GPU-Tagen war die RL-Version (Reinforcement Learning) des Policy Network fertig. Sie spielt für sich allein genommen stärker Go als die SL-Version.

Um das Value Network zu trainieren, kann man nicht einfach alle Stellungen aus einer Partie hernehmen, denn diese sind viel zu stark korreliert (jede entsteht aus einer vorigen durch Hinzufügen eines Steins, alle Stellungen führen zum selben Partie-Ausgang). Für die Trainingsdaten des Value Network zogen sich die Googler daher am eigenen Schopf aus dem Sumpf, indem sie die RL-Version des Policy Network gegen sich selbst spielen ließen. Aus jeder so generierten Partie wählten sie nur eine Stellung zufällig aus und gewannen daraus, zusammen mit der Information, wer die Partie gewonnen hat, einen Datenpunkt für das Training des Value Network. Auch dieser Datensatz umfasste insgesamt mehr als 30 Millionen Stellungen und das Training des Value Network dauerte eine Woche auf 50 GPUs.

MCTS mit Schuss

Der Clou bei AlphaGo liegt nun darin, wie die beiden neuronalen Netze in den MCTS-Suchalgorithmus integriert wurden. MCTS muss schnell gehen, denn nur wenn man sehr viele Partien simuliert, erwächst aus dem Zufall nützliches Wissen. 1000 Zufallspartien schafft die einfache Playout Policy von AlphaGo pro Sekunde. Eine einzelne Auswertung von Policy Network oder Value Network dauert dagegen selbst auf einer GPU Millisekunden; würde man dies in jeden simulierten Zug einbauen, würde das die Suche um Größenordnungen verlangsamen. Daher wertet AlphaGo diese Netze asynchron aus.

Jeder Knoten des Suchbaums speichert für jeden möglichen Zug außer den beiden MCTS-üblichen Zählern (wie oft probiert, wie oft gewonnen) weitere Informationen: Eine Anfangs-Wahrscheinlichkeit, die aus dem Policy Network stammt, sowie die mittlere Stellungsbewertung des Value Network für den gesamten Unterbaum.

Viele Threads – in der Match-Version 40 – führen nun parallel Monte-Carlo-Suchen ab der Wurzel aus, nach folgendem Algorithmus: Innerhalb des Baums wird der nächste Zug gewählt mit einer Mischung der An-

fangswahrscheinlichkeiten und der bisher gewonnenen Erkenntnisse, wobei sich das Gewicht im Laufe der Zeit von den Anfangswahrscheinlichkeiten weg hin zu Letzteren verschiebt. Das sind sowohl die nackten Monte-Carlo-Simulationszähler als auch die Einschätzungen der Value Network für die bisher im Baum gespeicherten Stellungen. Nach Verlassen des Baums geht es wie gehabt mit einer schnellen Playout Policy weiter; auf dem Rückweg werden die Playout-Zähler aktualisiert.

Spannend wird es, wenn der Baum wächst. Wenn von einem Blatt des Baums aus ein Zweig eine gewisse Anzahl von Besuchen übersteigt, dann wächst dort ein neues Blatt. Für dieses wird jetzt eine Einschätzung des Policy Network gebraucht, um die Anfangswahrscheinlichkeiten für die nächsten Züge ab hier zu speichern, und außerdem eine Einschätzung über den Wert dieser Stellung vom Value Network. Beides würde aber lange dauern, deshalb wirds jetzt asynchron: Die Stellung kommt in eine Warteschlange für die Auswertung von Policy und Value Network auf einer Grafikkarte. Als Anfangswahrscheinlichkeiten werden erst mal näherungsweise die der schnellen Playout Policy eingesetzt. Die Bewertung bleibt leer.

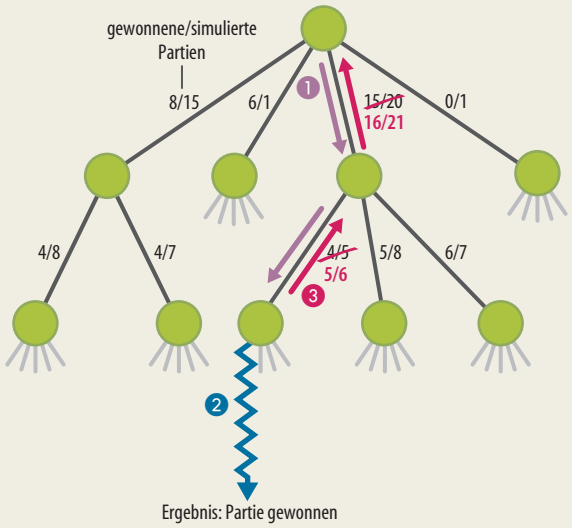
So kann der MCTS-Algorithmus erst einmal praktisch ungebremst wie gehabt den Baum verlassen, zufällig zu Ende simulieren und auf dem Rückweg nach oben im Baum die Zähler aktualisieren. Sollten weitere Simulationen an dem neuen Blatt vorbeikommen, arbeiten sie mit den vorläufigen Wahrscheinlichkeiten, aber das ist besser als Warten. Gleiches gilt für die Stellungsbewertung: Aus den Zufallspartien entsteht ja nach und nach bereits eine Einschätzung des Stellungswerts. Wenn das Value Network dann fertig ist, wird dessen Ergebnis vom neuen Blatt aus den Baum hinauf nachgetragen. Ebenso trägt ein Hintergrundthread die Ergebnisse des Policy Network als Anfangswahrscheinlichkeiten in das neue Blatt ein, sobald es fertig ist.

Die Wachstumsgeschwindigkeit des Baums steuert der Algorithmus so, dass die GPUs für die Berechnung von Policy und Value Networks voll ausgelastet sind. Schnelleres Wachstum hieße auf die neuronalen Netze verzichten, langsameres Rechenleistung verschenken. In der verteilten Version von AlphaGo, die auf mehreren Rechnern lief, führte ausschließlich der Master-Rechner die MCTS-Suchthreads aus, alle anderen waren nur mit der Auswertung von Policy und Value Networks befasst.

Es ist die Kombination all dieser Techniken, die AlphaGo zu seiner Spielstärke verhilft: In Tests schalteten die Forscher wechselweise Policy Network, Value Network oder Playouts ab und erlebten mehr oder weniger drastische Einbrüche der Spielstärke. Die vielleicht größte Neuerung gegenüber früheren Ansätzen sind die Value Networks, aber etwa im Vergleich mit den Bemühungen von Facebook ist auch die asynchrone Ausgestaltung des Algorithmus bemerkenswert.

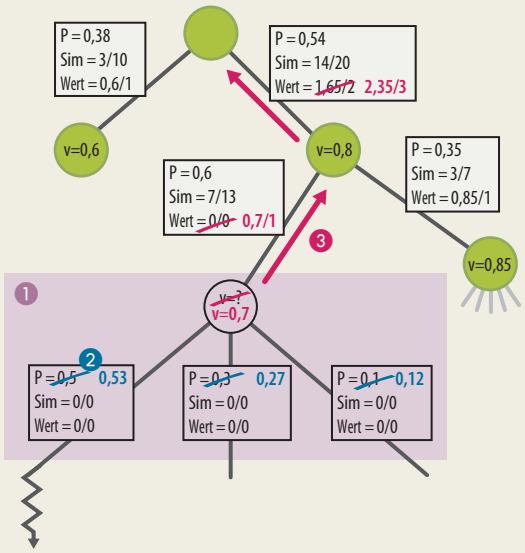
Monte Carlo Tree Search (MCTS)

1. Die Partie wird ab der aktuellen Stellung (Wurzel) simuliert. Die Zugwahl orientiert sich an den bisher ermittelten Statistiken.
2. Nach Verlassen des Baums spielt der Algorithmus mit einer festen Playout Policy zu Ende.
3. Auf dem Rückweg durch den Baum werden die Statistiken entsprechend dem Ergebnis der simulierten Partie aktualisiert.



MCTS + Policy + Value = AlphaGo

1. Ein neuer Knoten wird initialisiert mit Nullen in den Zählern und Schätzwerten für die Anfangswahrscheinlichkeiten. Im Hintergrund beginnt die Auswertung von Policy und Value Network für diese Stellung, während die Monte-Carlo-Simulation ungebremst weiterläuft.
2. Wenn die Ergebnisse des Policy Network vorliegen, ersetzen sie asynchron die geschätzten Wahrscheinlichkeiten.
3. Die Stellungsbewertung des Value Network wird asynchron im Baum nachgepflegt, sobald sie vorliegt.



Showdown ab 9. März

Nach dem Sieg gegen Fan Hui (2p) greift Google nun nach den Sternen und hat den weltstärksten Go-Spieler der letzten zehn Jahre herausgefordert, den Koreaner Lee Sedol. Es geht um eine Million US-Dollar Preisgeld. Fünf Partien werden ab dem 9. März gespielt, mit zwei Stunden Kernbedenkzeit und dann einer Minute pro Zug. Die Partien dürften also jeweils mindestens vier bis fünf Stunden dauern. Sie beginnen um 13 Uhr koreanischer Zeit, also 5 Uhr morgens MEZ. Die Partien werden live auf YouTube übertragen, auf Englisch kommentiert vom amerikanischen Go-Profi Michael Redmond (9p).

Lee Sedol gab sich in einem Interview optimistisch. Aber es sind fünf Monate vergangen, seit AlphaGo Fan Hui geschlagen hat – viel Zeit für das AlphaGo-Team, sein Baby weiter hochzupäppeln. Am 22. Februar schrieb einer der Haupt-Schöpfer von AlphaGo, Aja Huang, auf der Computer-Go-Mailingliste: „AlphaGo is getting stronger and stronger. I hope you all will enjoy watching the games.“ Wir werden auf heise online darüber berichten. (bo@ct.de)

Literatur

[1] Yuandong Tian, Yan Zhu, Better Computer Go Player with Neural Network and Long-term Prediction: http://arxiv.org/abs/1511.06410

[2] David Silver, Aja Huang et al, Mastering the game of Go with deep neural networks and tree search, Nature 529, 484–489, 28. 1. 2016

c't Weitere Infos, Live-Übertragung: ct.de/yuk6