Gymnasium am Markt Bünde

Facharbeit

- QAGI -

Universell lernende Agenten

2019 / 2020

Verfasser: Linus Schmidt

Kurs: Leistungskurs Informatik

Betreuer: Oliver Nienhüser

Abgabetermin: 21.02.2020

Inhaltsverzeichnis

1 Einleitung

Künstliche Intelligenz *(engl. artificial intelligence, AI)* ist in den letzten Jahren ein populäres Thema geworden. Immer mehr Firmen und technische Produkte basieren auf Algorithmen des maschinellen Lernens *(engl. machine learning, ML)*. Es handelt sich dabei um ein Teilgebiet der Informatik, welches sich mit der Erforschung von Mechanismen des intelligenten menschlichen Verhaltens befaßt. Viele bekannte Produkte wie Amazon Alexa, Siri von Apple oder auch Cortana von Windows verwenden maschinelles Lernen. Maschinelles Lernen bedeutet, die Generierung von Wissen aus Erfahrung. Das heißt, dass Algorithmen beim maschinellen Lernen ein statistisches Modell aufbauen, welches auf Trainingsdaten (Erfahrung) basiert. Systeme, wie die oben genannten, sind durch ihre Leistung für viele sehr beeindruckend. Man nennt diese Systeme schwache künstliche Intelligenz. Sie sind oftmals besonders gut in einem Themengebiet, jedoch können sie ihre Erfahrung, ihr Wissen oder ihre Schlüsse nicht auf andere unbekannte Umgebungen *(engl. Environment)* übertragen. Zum Beispiel kann Amazon Alexa sehr gut mit Menschen kommunizieren, jedoch wäre es nicht in der Lage ein Auto zu fahren oder einen Industrieroboter zu steuern. Eine KI, die das könnte, nennt man künstliche allgemeine Intelligenz (engl. artificial general intelligence, AGI). Es ist das aktuelle Forschungsthema und wäre eine Weltrevolution. Das erreichen von einer künstlichen allgemeinen Intelligenz würde die Welt verändern. Wenn schon der Intelligenzsprung von einem Gorilla zum Menschen die Welt so verändert, was kann dann eine künstliche allgemeine Intelligenz bewirken? In dieser Arbeit werde ich versuchen, mit meiner eigenen künstlichen Intelligenz *HPAGI* versuchen einen schnellen Algorithmus für universelles Lernen zu finden, der effizient und leistungsfähig ist.

2 AIXI

AIXI ist ein theoretischer mathematischer Formalismus für künstliche allgemeine Intelligenz. AIXI ist hierbei ein Agent, der in einer Umgebung in einem Zyklus $k = 1,2,...,m$ agiert. In einem Zyklus *k* führt dieser dann eine Aktion aus. Diese basiert auf vergangene Wahrnehmungen *(engl. perception)* $o_1 r_1 \dots o_{k-1} r_{k-1}$. Anschließend, bekommt der Agent eine neue Beobachtung *(engl. observation)* o und Belohnung *(engl. reward)* r. Diese werden zu den Vergangenen Handlungen hinzugefügt. Nun versucht AIXI die totale zukünftige Belohnung zu maximieren.

$$\text{AIXI} \qquad a_k := \arg\max_{a_k} \sum_{o_k r_k} \dots \max_{a_m} \sum_{o_m r_m} [r_k + \dots + r_m] \sum_{q\,:\,U(q,a_1..a_m)=o_1 r_1..o_m r_m} 2^{-\ell(q)}$$

Ist die Umgebung q deterministisch, so können die zukünftigen Wahrnehmungen $\ldots o_k r_k \ldots o_m r_m = U(q, a_1 \ldots a_m)$ durch eine universelle monotone „Turing machine“ U berechnet werden. Diese Berechnung erfolgt durch q mit gegeben Aktionen $a_1 \ldots a_m$. Da q jedoch unbekannt ist, muss AIXI die zukünftig erwartete Belohnung maximieren. Heisst, der Durchschnitt von allen möglichen zukünftigen Belohnungen $r_k + \ldots + r_m$, die durch alle möglichen Umgebungen q erzeugt werden, müssen mit den früheren Wahrnehmungen übereinstimmen. Der Beitrag einer Umgebung wird dabei mit $2^{-l(q)}$ berechnet. Das bedeutet, je einfacher die Umgebung, desto höher der Beitrag. Die Komplexität des Programms wird hierbei mit der Länge l beschrieben. Das Problem des Formalismus ist, dass dieser nicht berechnet werden kann. AIXI muss approximiert werden. Eine berechenbare Approximation[1] mit dem Monte Carlo Algorithmus zeigt, dass der Formalismus bei einer approximation funktioniert. Ein Problem welches auffällt ist, dass die Rechenzeit exponentiell zu der Größe des erstellten Baumes wächst. Das Bedeutet, dass die Approximation nur in übersichtlich kleinen Umgebungen Sinnvoll ist.

3 HPAGI

HPAGI[2] („High Performance Artificial General Intelligence“) soll eine Alternative zu AIXI sein. Es soll ein berechenbarer und effizienter Algorithmus sein, der in überschaubarer Zeit lernen kann. HPAGI basiert auf verschiedenen Algorithmen des maschinellen Lernens und repräsentiert hauptsächlich ein stochastischen Agenten. Das Ziel des Agenten ist es, möglichst viel Erfahrung aus verschiedenen Bereichen zu Sammeln, um dann die Belohnung einer Umgebung zu maximieren. Anschließend soll es dann das gesammelte Wissen aus der Historie auf neue unbekannte Umgebungen anwenden. Hierfür greift HPAGI auf verschieden approximierte oder kombinierte Algorithmen des maschinellen Lernens zurück. HPAGI ist ein theoretischer unvollständiger Ansatz zur künstlichen allgemeinen Intelligenz.

3.1 Algorithmische Struktur

3.1.1 Suchraumproblem

Das Problem, welches sich jedem Algorithmus stellt, ist das der Berechenbarkeit und Rechenzeit. In dem Zusammenhang möchte ich auf das Suchraumproblem eingehen.

[1] Veness, Ng, Hutter, Uther, Silver, 2009: A Monte Carlo AIXI Approximation.

[2] Schmidt, 2020: HPAGI. GitHub: https://github.com/Lxnus/hpagi

Da auf der Suche nach einem Beweis immer in jedem Schritt (je nach Kalkül) unendlich viele Möglichkeiten für die Anwendung von Inferenzregeln gibt, wurden Algorithmen mit heuristischen Verfahren entwickelt. Heuristische Verfahren müssen unter beschränkten Ressourcen Realzeitentscheidungen herbeiführen. Dabei können sie den Weg zum Ziel oftmals verkürzen, indem sie über eine heuristische Bewertungsfunktion verfügen. Andererseits können heuristische Verfahren, bezogen auf Suchbäume, mehr Rechenleistung benötigen mit einer falschen Lösung.

3.1.2 Monte Carlo Tree Search

Der Monte Carlo Tree Search[3] Algorithmus beschreibt einen Suchbaum des bestärkten Lernens mit heuristischer Funktion. Standardmäßig definiert die „Upper Confidence Bound 1“ die heuristische Funktion von einem Knoten x. Dafür wird die durchschnittliche Erfolgsrate v berechnet.

$$v = \frac{v}{w}$$

Hier bezeichnet v die Anzahl, wie oft x besucht worden ist und w die Anzahl der Erfolge. Die Konstante $C = \sqrt[2]{(2)}$ ist der explorations Parameter und empirisch festgelegt. Die Wahrscheinlichkeit, wie oft ein untergeordneter Knoten n besucht wurde, wird mit $\sqrt[2]{(\frac{log_2 N}{n_i})}$ verwendet. Dabei ist N die Anzahl der Besuche von x.

Zusammen ergibt es dann die UCB1 Funktion.

$$u_i = v_i + C \cdot \sqrt[2]{(\frac{log_2 N}{n_i})}$$

Sie beschreibt die Wahrscheinlichkeit, ob der Knoten zu einem Erfolg führt. Erfolg ist dabei definiert mit dem Erlangen einer hohen Belohnung. Damit die Funktion unter der Bedingung $N \leq 0 \vee n \leq 0$ aufgeht, wird unter diesen Umständen u wie folgt berechnet[4]:

$$u_i = \varepsilon * r = 1e - 6 * rand * k$$

Die zufällige Zahl $rand = [0,1] = \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ wird hier als Parameter benutz, damit nicht jeder Knoten die gleiche Wahrscheinlichkeit erhält, der richtige zu sein. Ansonsten würde der Algorithmus verharren, weil es keinen „besten“ Knoten

[3] Salloum, 2019: Monte Carlo Tree Search in Reinforcement Learning.

[4] Anhang, Teil A. 1.4

findet, da alle gleiche Werte besitzen. Zudem kann mit der möglichen zukünftigen Belohnung k die Rechenzeit minimieren werden, da die UCT (*Upper Confidence Tree, Formel für Bäume von UCB1)* Funktion durch den beobachtbaren Parameter einen theoretischen zweiten heuristischen Wert bekommt. Das Problem ist, dass es, wie das Suchraumproblem es beschriebt, nicht bedingt zu der richtigen Lösung kommen muss. Zudem kann der Knoten standardmäßig nur einen Wert abspeichern. Heisst, es könnte nicht generalisiert in verschiedenen Umgebungen agieren. Eine Lösung wäre eine approximiertes Modell für die Universalität. Alpha Go Zero[5] ein generalisierte Algorithmus basierend auf Monte Carlo Tree Search und Convolutional Neural Networks[6], zeigt eine mögliche Approximation. Dieses Verfahren beharrt nicht auf einer Aktion a. Es ist in der Lage, eigenständig Strategien zu erlernen und diese auf neue unbekannte Umgebungen zu testen (Umgebung hier definiert als neues Spiel). Eine andere Möglichkeit MCTS zu approximieren ist die, dass man a als einen binären Vektor A darstellt. In A gilt: Wenn der Index $i = a$, dann ist $A_i = 1$. Ein anderes Konzept basiert auf der Extrahierung von a, wenn den der Knoten zu einem Erfolg führt. Dann wird a von dem Knoten in den binären Vektor umgewandelt und zu einem Datensatz hinzugefügt. Am Ende von dem Durchlauf wird dann ein stark komprimierter Datensatz erstellt. Dieser repräsentiert den Lösungsweg von dem MCTS-Baum. Das hat den Vorteil, dass man den Baum reproduzieren kann, ohne Knoten, die nicht verwendet worden sind. Das erspart bei zukünftigen Berechnungen viel Rechenzeit[7] und Speicher.

3.1.3 Bayes'sche Entscheidungsnetze

Bayes'sche Netzwerke[8] sind stochastische gerichtete Graphen, welche das Bayes'sche Inferenz[9] zur Wahrscheinlichkeitsberechnung verwendet. Durch die Abhängigkeit der gerichteten Kanten in dem Graphen, versucht ein Bayes'sches Netz eine Abhängigkeit und damit Kausalität zu modellieren. Dabei erfüllen Bayes'sche Netze die Eigenschaft, dass ein Knoten nur bedingt unabhängig von seinen nicht -Nachkommen ist. Diese Eigenschaft nennt man *lokale Markov Eigenschaft*. Die Inferenz kann als eine statistische Ableitung von Eigenschaften aus einer Menge oder als Wahrscheinlichkeitsverteilung aus einem Datensatz definiert werden. Heisst, die

[5] Deepmind, 2017: Mastering the game of Go without human knowledge.

[6] Saha, 2018: A Comprehensive Guide to Convolutional Neural Networks.

[7] Anhang, Teil A

[8] Soni, (2018): Introduction to Bayesian Networks.

[9] Brooks-Bartlett, (2018): Probability concepts explained: Bayesian inference for parameter estimation.

Bayes'sche Inferenz ist die Ableitung von Eigenschaften oder Wahrscheinlichkeitsverteilung aus Daten unter Verwendung des Bayes'sche Theorems. Die Inferenz wird wie folgt beschrieben:

$$P(x \mid e) = a \sum_{\forall y \in Y} P(x, e, Y)$$

Hier beschreibt $P(x \mid e)$ die Wahrscheinlichkeit einer Zuweisung einer Teilmenge x, die Zuweisungen der Evidenz e erhalten haben, zu finden. Um dies zu berechnen nehmen wir an, dass $P(x \mid e) = aP(x, e)$ ist. Dabei ist a eine Normierungskonstante, die so berechnet wird, dass $P(x \mid e) + P(\neg x \mid e) = 1$ ergibt. Um $P(x, e)$ berechnen zu können, ist es notwendig, dass wir die Wahrscheinlichkeitsverteilung, die nicht in x oder e vorkommen marginalisieren. Diese wird als Y notiert. Durch die oben angegebenen Eigenschaften lässt sich durch die Kettenregel die allgemeine gemeinsame Formel vereinfachen. Nach der Vereinfachung lässt sich die Formel wie folgt beschreiben:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid X_1, \ldots, Xi - 1) = \prod_{i=1}^{n} P(X_i \mid Eltern(X_i))$$

Eine Herausforderung hierbei ist die automatische und autonome Generierung von einem Bayes'schem Netzwerk. Mithilfe der Bayesserver[10] API ist es möglich. Die API stellt diversen *Parameter* und *Structural learning* Algorithmen zur Verfügung. Diese Algorithmen kann man kombinieren. Dadurch war es möglich, einen *BayesianNetworkBuilder*[11] zu erstellen, der unter bedingten Parametern und Daten ein Bayes'sches Netz erstellen kann. Dadurch soll es ermöglicht werden, dass der Agent, mithilfe von einer Historie, autonome Entscheidungen durch ein automatisch generiertes Bayes'sches Netz tätigen kann. Ein möglicher Vorteil hierbei: Das Netz kann sich selbstständig optimieren, indem es neue Verzweigungen herstellt.

3.1.4 Der Autoassoziativspeicher Hopfield

Hopfield Netzwerke können auch als Autoassoziativspeicher bezeichnet werden. Sie können als Suchalgorithmus nach einem Minimum der Energiefunktion im Raum der Aktivierungsfunktion beschrieben werden. Hopfield Netze können auf verschiedene Energiefunktionen angewandt werden, solange die Gewichtsmatrix w symmetrisch ist und die Diagonalelemente null sind. Das liegt daran, dass die Hopfield-Dynamik

[10] Bayesserver API: https://www.bayesserver.com/

[11] Anhang, Teil C

äquivalent zu dem physikalischem Modell des Magnetismus ist. Dort gilt $w_{ii} = 0$, denn die Teilchen dort haben keine Selbstwechselwirkung. Die Symmetrie lässt sich ebenfalls aus der Physik ableiten, denn die Wechselwirkungen sind immer symmetrisch. Deswegen gilt ebenfalls $w_{ij} = w_{ji}$.

Dabei wird die Energiefunktion wie folgt beschrieben:

$$E = -\frac{1}{2} \sum_{i,j} w_{ij} x_i x_j$$

Da bei einer Aktualisierung in jedem Schritt die Hopfield-Dynamik von den Zuständen *-1, 1* immer den mit der kleineren Gesamtenergie nimmt, lässt sich der Beitrag eines Neurons i wie folgt beschreiben:

$$-\frac{1}{2} x_i \sum_{j \neq i}^{n} w_{ij} x_j$$

Dabei werden die Neuronen nach folgender Regel aktiviert:

$$x_i = \{ \begin{matrix} -1 \textbf{ falls } \sum_{j=1, j \neq i}^{n} w_{ij} x_j < 0 \\ 1 \textbf{ sonst} \end{matrix}$$

Die Idee bei der Verwendung ist die, dass man Speicher[12] *(engl. Memory)* einbauen können, die Assoziativ arbeiten. Zudem kann man binäre Vektoren verwenden, welches in der Rechenzeit betrachtet bedeutet, dass man kleinere primitive Datentypen verwenden kann. Der kleinstmögliche wäre dann *byte, der sich in einem Zahlenraum von -128, 128* bewegt. Zudem kann man Daten abspeichern, die Assoziativ verwendet werden können. Ein Beispiel hierbei ist die Mustererkennung. Bei Mustern kann es sein, dass nicht jeder Pixel an der gleichen Stelle ist, wie ein Identisches anderes Bild. Durch den assoziativen Speicher kann man diese Muster trotzdem klassifizieren. Somit müssen nicht jede einzelne Daten abgespeichert werden, sondern es kann dynamischer Speicher erstellt werden und viel Computerspeicher gesparrt werden.

3.2 HPAGI Konzept

Das Konzept basiert auf dem Zusammenschluss von allen vorgestellten Algorithmen. MCTS erweist sich als eine gute und starke heuristische Funktion. Bayes'sche Netze als schnelle und einflussreiche autonome Entscheidungsnetze und Hopfield-Speicher

[12] Anhang, Teil B

als Dynamische Assoziative Speicher. Das Problem, warum nicht wie oben vorgestellt, die extrahierten binären Vektoren aus den MCTS-Knoten als Trainingsdaten verwendet werden können ist die, dass der binäre Vektor eine Aktion repräsentiert. Heisst, wenn ein binärer Vektor $X = [0,0,1,0]$ ist, dann würde dieser übersetzt die Aktion $a = 3$ definieren. X ist dabei an die Definition gebunden. Es kann also nicht für eine Bayes'sche Entscheidung verwendet werden, da kein Wert unter der Bedingung eines anderen steht. Dafür müsste man MCTS approximieren. Eine logische und funktionierende Lösung ist die von Deepmind (s. o.). Durch die Generalisierung von MCTS wäre es möglich dem Agenten ein Verhalten beizubringen, welches universell auf unbekannte Umgebungen anwendbar ist. Das würde das Problem der Gebundenheit an eine Umgebung lösen, denn die Knoten würden keine Aktion, sondern ein Verhalten, speichern. Übertragend auf einen Baum wäre eine Verbindung zu Bayes'schen Netzen gegeben. Denn, der Pfad zu einem Knoten würde übersetzt folgendes bedeuten: *Unter folgenden Bedingungen und Aktionen von Zuständen ausgelöst, wurde die Aktion gefordert und als nächste auserwählt.* Wenn man nun diese Definition mit der von Bayes vergleicht:

> *„Durch die Abhängigkeit der gerichteten Kanten in dem Graphen, versucht ein Bayes'sches Netz eine Abhängigkeit und damit Kausalität zu modellieren."*

Man erkennt, dass beides das gleiche aussagt. Sowohl MCTS als auch Bayes beschreiben ein Modell der Kausalität. Eins als Baum, das andere als Graph. Da nicht das Ziel ist, beides zu kombinieren, sondern die Kausalität zu kombinieren, müssen die Kausalitäten sinnvoll extrahiert werden und kombiniert werden. Eine Möglichkeit hier wären die empfohlenen Aktionen, die nach einer Generalisierung und gelernten MC-Baum herauskommen, so zu formulieren, dass eine implementationsfähige Kausalität für Bayes herauskommt. Da ein Pfad zu einem Knoten x bereits eine *Abhängige Kausalität zu einem Knoten über einen gerichteten und gerichteten Pfad* beschreibt, wäre das Problem gelöst. Dadurch könnte ein autonomes Modell, welches autonom Entscheidungen treffen kann entwickelt werden. Es würde in sofern mächtig sein, da durch die Generalisierung von MC-Bäume viele komplexe Umgebungen erlernt werden können. Das Problem dabei ist die Rechenzeit. MC-Bäume expandieren exponentiell, welches bedeutet, dass auch die Rechenzeit exponentiell steigt. Ein Lösungsansatz wäre hierbei die in 3.1.2 vorgestellte Variante der Wiederherstellung eines Baumes durch Vektoren. Da es aber generalisiert wurde, durch Deep learning Algorithmen, müssen diese abgespeichert werden. Stellt dies ein Problem dar? Nein, denn selbst Neuronale Netzwerke *(engl. Neural Networks, NN)* können abgespeichert werden. Dann würde es keinen binären Vektor mehr geben, sondern eine Folge von gespeicherten Neuronalen Netzwerken. Um den Agenten lernfähig zu machen ist es erforderlich, dass er zum einen in möglichst universellen Umgebungen trainiert und zum anderen eine gute Kommunikation zwischen der Umgebung und dem Agenten

besitzt. Hierbei sollte die Umgebung folgendes Umfassen: *Logische Schlussfolgerungen, Lernen von Regeln, Speichern und Wiedererkennung aus der Vergangenheit und kombinieren von Daten.* Eine Umgebung, die diese Aspekte beinhalten würde, wäre Minecraft. Minecraft ist 3D Sandbox Spiel. Der Vorteil hierbei wäre, dass es *open source* ist. Dadurch kann man den Agenten einfach und unkompliziert in das Spiel einbauen. Eine andere Möglichkeit wäre, mehrere Umgebungen selbst zu entwicklen, in denen dann der gleiche Agent zurecht kommen muss. Wie aber kann man erreichen, dass der Agent unbewacht bestärkt lernt? Dazu möchte ich kurz auf die Methoden des Lernens mit Verstärkung *(engl. Reinforcement learning)* eingehen. Dabei ist das Ziel, dass der Agent eine möglichst effiziente Strategie *(engl. Policy)* erlernt. Dabei befindet sich der Agent in einer Umgebung und hat dort die Möglichkeit eine Aktion auszuführen. Daraufhin folgt eine Belohnung. Diese bewertet so gesehen die Aktion und dessen Auswirkungen auf den Agenten. Zudem bekommt der Agent eine neuen beobachtbaren Zustand. Diese Beobachtung wird von der Aktion ausgeführt. Bekannte Verfahren auf dem Gebiet sind das *Q-Learning*[13] und die *Wert Iteration*[14] mit dem *Dynamischen Programmieren*[15]*.* Diese Algorithmen werden meist im Kontext von anderen Algorithmen des maschinellen Lernens angewandt. Sie sind sehr effizient, können allerdings eine gewisse Zeit beanspruchen. Dadurch, dass (beispielsweise Q-Learning) lediglich auf den Zuständen und den Belohnungen der Umgebung basiert und nicht wie überwachte Lernalgorithmen vorgeschrieben bekommt, was es zu tun hat, kann das finden einer optimalen Strategie, je nach Umgebung, sehr lange dauern.

3.2.1 Fluch der Dimensionen

Dieses Problem wird auch als *Fluch der Dimensionen (engl. Curse of dimensionality).* Das Problem dabei wird in den Dimensionen beschrieben. Auf den Agenten übertragen müsste dieser, wie der Mensch es auch tut, langsam und in Schritten lernen. Das heisst, dass der Agent wieder neue kleinere Bereiche erkunden. Kapselförmig. Das Problem dabei ist jedoch das gleiche. Die Dimensionen der Vektoren werden zu lang, was die Rechenzeit und die Komplexität gigantisch erhöht. Das sieht man schon bei dem Versuch HPAGI-MCTS. Die Rechenzeit und Komplexität steigt enorm zu der Größe, der Umgebung. Dies lässt sich gut an dem Beispiel eines Spielbrettes darstellen. Dort muss ein Agent ein Ziel finden. Beide Koordinaten werden zufällig initialisiert unter der Bedingung, dass die Koordinaten nicht mit anderen übereinstimmen.

---

[13] Grundkurs Künstliche Intelligenz: Kapitel 10.6 - 10.7

[14] Grundkurs Künstliche Intelligenz: Kapitel: 10.4 - 10.5

[15] Grundkurs Künstliche Intelligenz: Kapitel: 10.4 - 10.5

4 Fazit

Zusammenfassend kann gesagt werden, dass HPAGI eine theoretische Alternative zu AIXI ist. Beides sind zwar theoretische Ansätze, lassen sich aber bei einer fortgeschritteneren Umsetzung oder Approximation anwenden und berechnen. HPAGI erweist sich als eine gute Alternative, da es auf die verschiedenen Probleme von *Multi-Agent* Systemen eingeht und viele Algorithmen so verändert, das die Rechenzeit logisch vermindert wird. Dies zeigt das vorgestellte Extrahierungsverfahren von Knoten des Monte Carlo Algorithmus. Jedoch verbleiben genauso bei HPAGI, als auch bei AIXI die klassischen Probleme. Gerade der *Fluch der Dimensionen*, das Problem der Komplexität und der Berechenbarkeit sind weiterhin Bestandteil und müssen gelöst werden. Zudem kommt die Frage der Generalisierung auf. HPAGI stellt zwar eine Reihe von kombinierten Algorithmen dar, um ein stochastischen, approximierten und autonomen Agenten zu erstellen, jedoch bleibt das Problem der Rechenzeit und der geforderten Rechenleistung beibehalten. HPAGI lässt sich also als einen generalisierten approximierten Agenten definieren, der unter der Hauptfunktion von *Monte-Carlo, Bayes'schen Netzen, Hopfield Netzen* und mit lernen mit Verstärkung arbeitet.

5 Diskussion

In der Arbeit wurden diverse Kombinationen von Algorithmen aus dem Bereich des maschinellen Lernens verwendet. Es gibt jedoch noch Aspekte, die eine alternative zu diesen darstellen können. Zum Beispiel wäre es denkbar, dass man den Fokus mehr auf das Deep learning setzt. Gerade die Kategorie der Neuronalen Netze wäre sehr interessant. Jedoch spielen auch dort die klassischen Problem wie der *Fluch der Dimensionen, die Komplexität* und *die Rechenzeit* eine Rolle. In den Experimenten haben wir gezeigt, dass *Hopfield-Netzwerke* eine gute Alternative zu normalen Speichermethoden sind. Sie sind dynamisch und assoziativ. Das verbraucht weniger Speicher und Daten können schnell und einfach klassifiziert und reproduziert werden. Auf der anderen Seite haben wir gezeigt, dass es mit dem theoretisch vorgestellten Ansatz möglich ist, dass man *MCTS-Bäume* mit *Bayes'schen Netzen* verbinden kann und das sogar noch recheneffizient. Zudem wurde gezeigt, dass generalisierte Bäume extrahiert werden können und Trainingsdaten erstellt werden können. Natürlich kann man hier überlegen, ob die Rechenleistung dadurch vermindert wird. In der Programmierung wäre das nicht der Fall. Da es sich um *Neuronale Netzwerke* oder ähnliches handelt, würde die Rechenleistung enorm steigen. Also auch hier ein Punkt,

über dem debattiert werden kann, ob es sinnvoll ist. Zusammenfassend lässt sich sagen, dass die Theorie und Praxis umsetzbar ist, jedoch die Probleme, die vorgestellt worden sind, immer bleiben. Die Probleme könnten gelöst werden, indem der Computer versucht selber und eigenständig Algorithmen zu erstellen. Da ein Computer aber keine Algorithmen erstellen kann, da er nicht weiß wann er determiniert und nicht testen kann, ob dieser für alle Zahlen funktioniert, kann dieser nur eine Folge von Algorithmen erstellen. Trotz dessen kann man so erreichen, das der Computer eine effizientere Methode findet. Durch die Entwicklung des Komprimierungsverfahrens und kann in Zukunft eine Menge Rechenleistung und Rechenzeit gesparrt werden. Bäume können neue Dimensionen und großen erlangen.

Literaturverzeichnis

1. Ertel, Wolfgang (2016): Grundkurs Künstliche Intelligenz, Eine praxisorientierte Einführung, 4. Auf., Wiesbaden: Springer Vieweg Verlag.

2. Salloum, Z. (Feb. 18, 2019): Monte Carlo Tree Search in Reinforcement Learning. Online im Internet: https://towardsdatascience.com/monte-carlo-tree-search-in-reinforcement-learning-b97d3e743d0f (Stand: Feb. 18, 2019)

3. Saha, S. (Dez. 17, 2018): A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way. Online im Internet: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (Stand; Dez. 17, 2018)

4. Deepmind (2017): Mastering the game of Go without human knowledge. Online im Internet: https://www.nature.com/articles/nature24270.epdf?author_access_token=VJXbVjaSHxFoctQQ4p2k4tRgN0jAjWel9jnR3ZoTv0PVW4gB86EEpGqTRDtpIz-2rmo8-KG06gqVobU5NSCFeHILHcVFUeMsbvwS-lxjqQGg98faovwjxeTUgZAUMnRQ (Stand: 2017)

5. Veness, J.; Ng, K. S.; Hutter, M; Uther, W.; Silver, D. (4. Sep. 2009): A Monte Carlo AIXI Approximation. Online im Internet: https://arxiv.org/abs/0909.0801 (Stand: 26. Dez. 2010)

6. Soni, D.; (Jun. 8, 2018): Introduction to Bayesian Networks. Online im Internet: https://towardsdatascience.com/introduction-to-bayesian-networks-81031eeed94e (Stand: Jun. 8, 2018)

7. Brook-Bartlett, J.; (Jan. 5, 2018): Probability concepts explained: Bayesian inference for parameter estimation. Online im Internet: https://towardsdatascience.com/probability-concepts-explained-bayesian-inference-for-parameter-estimation-90e8930e5348 (Stand: Jan. 5, 2018)

Anhang

1 Teil A

1.1 UCT *Selection* Algorithmus

```
46  private MCTSNode select() {
47      MCTSNode best = null;
48      double bestValue = Double.NEGATIVE_INFINITY;
49      for(MCTSNode child : this.children) {
50          assert child.v > 0;
51          double uct = (child.v == 0 ? 0 : child.w / child.v + this.c * Math.sqrt(Math.log(this.v + 1) / child.v));
52          if(uct > bestValue) {
53              bestValue = uct;
54              best = child;
55          }
56      }
57      return best;
58  }
```

1.2 MC-Baum Daten Extrahierung u. Komprimierung

```
102     public LinkedList<double[]> trainingData(IEnvironment environment) {
103         if(this.successWay != null) {
104             LinkedList<double[]> trainingData = new LinkedList<>();
105             for (MCTSNode current : this.successWay) {
106                 double[] data = new double[environment.possibleActions().length];
107                 for (int i = 0; i < data.length; i++) {
108                     if (i == current.s) {
109                         data[i] = 1.0D;
110                     }
111                 }
112                 trainingData.add(data);
113             }
114             return trainingData;
115         }
116         return null;
117     }
```

```
40      private void compress() {
41          this.compressed = true;
42          LinkedList<double[]> data = this.rootNode.trainingData(this.environment);
43          this.environment.reset();
44          LinkedList<Double> processed = new LinkedList<>();
45          for(double[] vector : data) {
46              for (int k = 0; k < vector.length; k++) {
47                  if (vector[k] == 1) {
48                      processed.add((double) k);
49                      break;
50                  }
51              }
52          }
53          int nodes = this.rootNode.getNodes();
54          MCTSNode compressedTree = new MCTSNode();
55          compressedTree.setSuccessWay(this.rootNode.getSuccessWay());
56          compressedTree.expand(processed);
57          if(compressedTree.getNodes() < nodes) {
58              this.rootNode = compressedTree;
59          }
60          this.environment.reset();
61      }
```

1.3 Testergebnisse MC

Man kann gut erkennen, das der Agent mit dem MC Algorithmus es schnell schafft, seinen Reward zu maximieren. X = Iteration, Y = Belohnung.

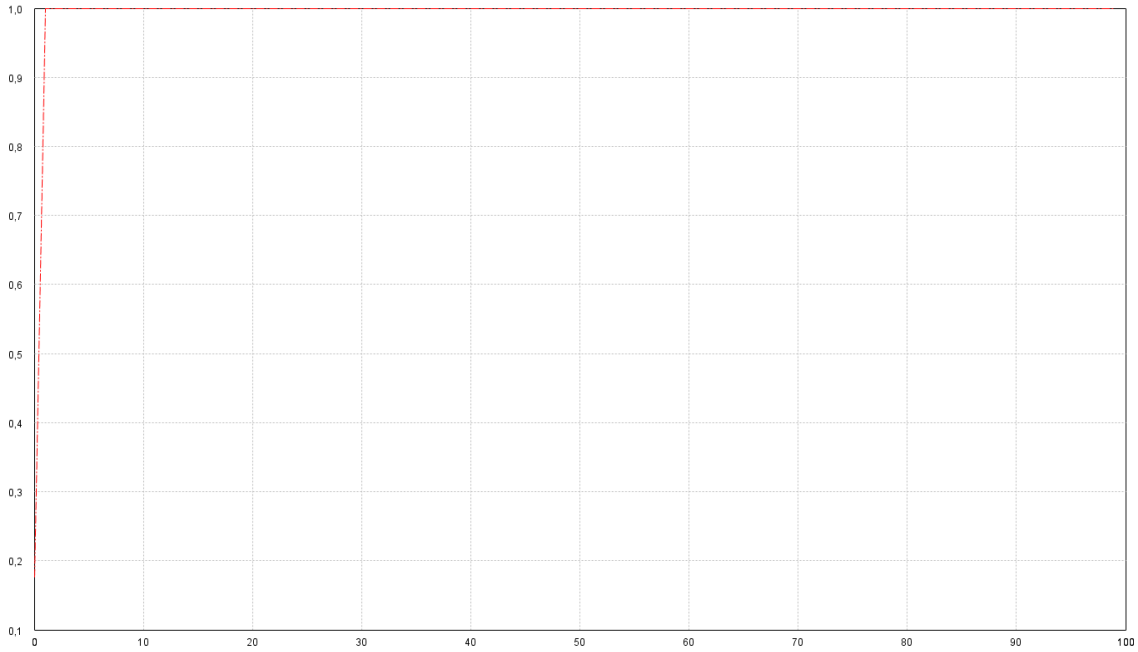

Abb.1

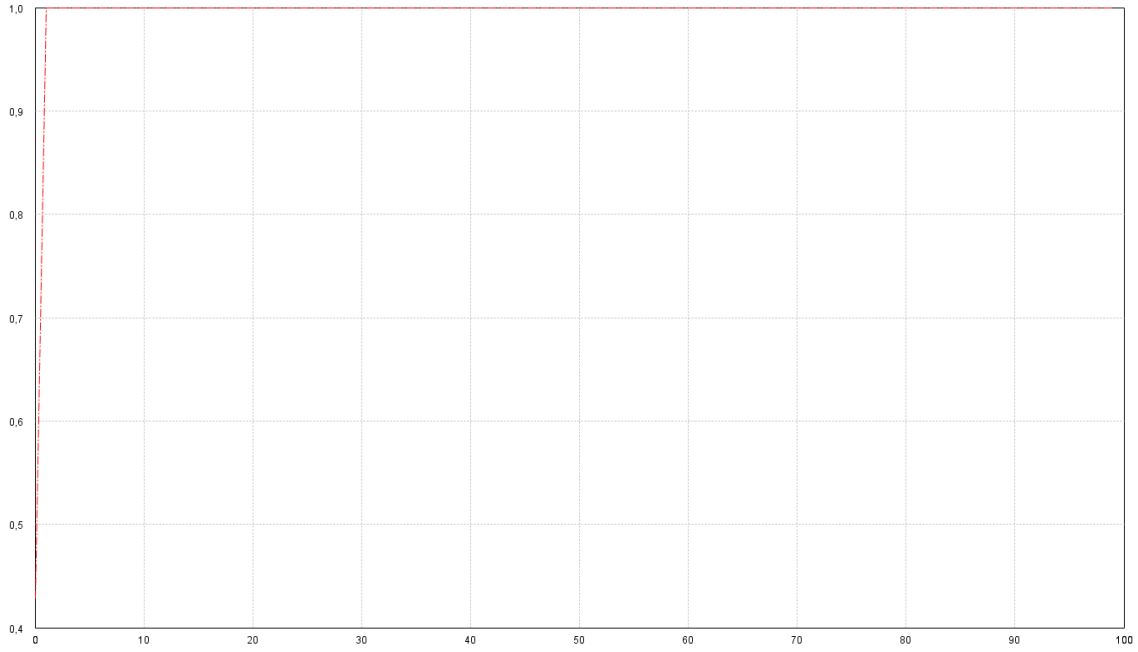

Abb. 2

1.4 Resultate Komprimierung u. Rechenzeit

Computation time wird hier in ms (Millisekunden) ausgegeben.

```
[19:28:28][HPAGI]: **********************************************
[19:28:28][HPAGI]: NPC: [14][2], TargetNPC: [14][14]
[19:28:28][HPAGI]: Testing monte carlo...
[19:28:37][HPAGI]: Vector: [0.0, 1.0, 0.0, 0.0]
[19:28:37][HPAGI]: Vector: [0.0, 1.0, 0.0, 0.0]
[19:28:37][HPAGI]: Vector: [0.0, 1.0, 0.0, 0.0]
[19:28:37][HPAGI]: Vector: [0.0, 1.0, 0.0, 0.0]
[19:28:37][HPAGI]: Vector: [0.0, 1.0, 0.0, 0.0]
[19:28:37][HPAGI]: Vector: [0.0, 1.0, 0.0, 0.0]
[19:28:37][HPAGI]: Vector: [0.0, 1.0, 0.0, 0.0]
[19:28:37][HPAGI]: Vector: [0.0, 1.0, 0.0, 0.0]
[19:28:37][HPAGI]: Vector: [0.0, 1.0, 0.0, 0.0]
[19:28:37][HPAGI]: Vector: [0.0, 1.0, 0.0, 0.0]
[19:28:37][HPAGI]: Vector: [0.0, 1.0, 0.0, 0.0]
[19:28:37][HPAGI]: Vector: [0.0, 1.0, 0.0, 0.0]
[19:28:37][HPAGI]: Nodes-before: 10645500
[19:28:37][HPAGI]: Nodes-after: 12
[19:30:05][HPAGI]: Computation time: 96947
[19:30:05][HPAGI]: Computation time: 0
[19:30:05][HPAGI]: Computation time: 0
[19:30:05][HPAGI]: Computation time: 0
[19:30:05][HPAGI]: Computation time: 0
[19:30:05][HPAGI]: Computation time: 0
[19:30:05][HPAGI]: Computation time: 0
[19:30:05][HPAGI]: Computation time: 0
[19:30:05][HPAGI]: Computation time: 0
[19:30:05][HPAGI]: Computation time: 0
```

1.5 Visualisierung Baum vorher u. Nachher

Vorher (Abb.1, Abb. 2):

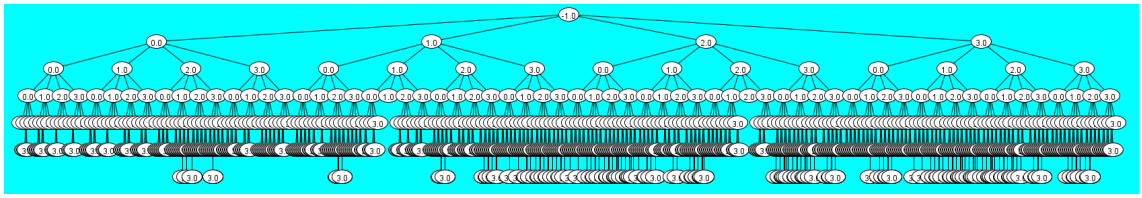

Abb. 1

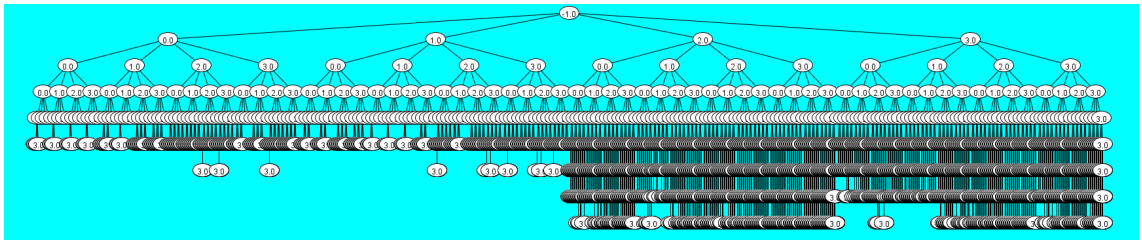

Abb. 2

Nachher (Abb. 3, Abb. 4):

Wie zu erkennen ist, wird der Baum von Abb. 1 deutlich reduziert. Das kann man gut in dem Komprimierten Baum Abb.3 erkennen. Das gleiche gilt für Abb. 2 und Abb. 4.

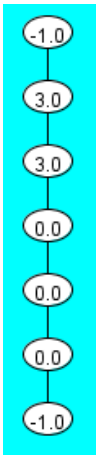

Abb.1

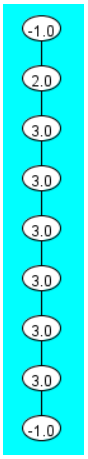

Abb. 2

2 Teil B

2.1 Hopfield

```
42 @    public double[] recreate(double[] input, int maxIter) {
43          double[] output = new double[input.length];
44          for(int i = 0; i < maxIter; i++) {
45              for(int j = 0; j < output.length; j++) {
46                  double value = this.activate(input, j);
47                  if(value > 0.0D) {
48                      output[j] = 1.0D;
49                  } else {
50                      output[j] = -1.0D;
51                  }
52              }
53          }
54          return output;
55      }
56
57      public void train() {
58          for(int i = 1; i < this.neurons; i++) {
59              for(int j = 0; j < i; j++) {
60                  for(double[] data : this.data) {
61                      double value = this.hopfieldValue(data[i]);
62                      double target = this.hopfieldValue(data[j]);
63                      double temp = target * value;
64                      int intValue = (int) (temp + weights[i][j]);
65                      this.weights[j][i] = this.weights[i][j] = intValue;
66                  }
67              }
68          }
69          for(int i = 0; i < this.neurons; i++) {
70              this.storage[i] = 0.0D;
71              for(int j = 0; j < i; j++) {
72                  this.storage[i] += this.weights[i][j];
73              }
74          }
75      }
```

```
96  @    private double activate(double[] X, int idx) {
97           double sum = 0.0D;
98           for(int i = 0; i < this.neurons; i++) {
99               if(i != idx) {
100                  sum += this.weights[idx][i] * X[idx];
101              }
102          }
103          return 2.0D * sum - this.storage[idx];
104      }
105
106 @    private double hopfieldValue(double x) {
107          return x < 0.0D ? -1.0D : 1.0D;
108      }
```

3 Teil C

3.1 Bayes'sche Netzwerk

3.1.1 Daten

```
210 @    private DataTable generateDataTable(LinkedList<double[]> data) {
211          DataTable dataTable = new DataTable();
212          DataColumnCollection dataColumns = dataTable.getColumns();
213          for(String dataDescription : this.dataDescriptions) {
214              dataColumns.add(dataDescription, String.class);
215          }
216          DataRowCollection dataRows = dataTable.getRows();
217          for(double[] vector : data) {
218              String[] processedNodeData = this.processedNodeData(vector);
219              dataRows.add(processedNodeData);
220          }
221          return dataTable;
222      }
```

3.1.2 Struktur

```
177  private void createNetworkStructure(EvidenceReaderCommand evidenceReaderCommand) {
178      /*
179      PCStructuralLearning structuralLearning = new PCStructuralLearning();
180      PCStructuralLearningOptions structuralLearningOptions = new PCStructuralLearningOptions();
181      PCStructuralLearningOutput structuralLearningOutput =
182              (PCStructuralLearningOutput) structuralLearning.learn(
183                      evidenceReaderCommand,
184                      this.network.getNodes(),
185                      structuralLearningOptions);
186      */
187      this.printer.printConsole("Create network structure...");
188      ChowLiuStructuralLearning chowLiuStructuralLearning = new ChowLiuStructuralLearning();
189      ChowLiuStructuralLearningOptions chowLiuStructuralLearningOptions = new ChowLiuStructuralLearningOptions();
190      ChowLiuStructuralLearningOutput chowLiuStructuralLearningOutput =
191              (ChowLiuStructuralLearningOutput) chowLiuStructuralLearning.learn(
192                      evidenceReaderCommand,
193                      this.network.getNodes(),
194                      chowLiuStructuralLearningOptions);
195  }
```

3.1.3 Parameter

```
196  private void createDistributions(EvidenceReaderCommand evidenceReaderCommand) {
197      ParameterLearning parameterLearning = new ParameterLearning(this.network, new RelevanceTreeInferenceFactory());
198      ParameterLearningOptions parameterLearningOptions = new ParameterLearningOptions();
199      parameterLearningOptions.setDecisionPostProcessing(DecisionPostProcessingMethod.PROBABILITIES);
200      parameterLearningOptions.setConvergenceMethod(ConvergenceMethod.PARAMETERS);
201      parameterLearning.learn(evidenceReaderCommand, parameterLearningOptions);
202  }
```

„Ich erkläre hiermit, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.“

Datum: **Unterschrift:**