Überwachtes Lernen vs. Bestärkendes Lernen

Seminararbeit
Vertiefungen und Anwendungen für KI

Julius Schmidt
Inf103009

Inhalt

1 Einleitung

Im Jahre 2018 nutzten mehr als 4 Milliarden Menschen das Internet[1], zwei Drittel der Menschheit besaßen ein Smartphone[2]. Die Digitalisierung schreitet voran und immer mehr Daten werden generiert. Wo die Künstliche Intelligenz (KI) in den 90er Jahren aufgrund eines Mangels an Forschungsdaten und leistungsfähigen Computern[3] schnell an ihre Grenzen stieß, kann sie sich nun kaum vor Daten retten. Und auch Computer wurden mit der Zeit immer schneller. So besaßen CPUs (Prozessoren) 2018 mehr als 120 Tausend mal so viele Transistoren wie noch 1978[4]. Das führt dazu, dass momentan so viel in die „Künstliche Intelligenz"-Forschung investiert wird wie noch nie. Alleine die USA investierte 2018 ca. 1,3 Milliarden Euro[5] für den nicht geheimen Teil der Forschung.

Immer mehr Menschen fangen an sich für das Thema Künstliche Intelligenz zu interessieren. Dies liegt vor allem an Aufsehen erregenden Ereignissen wie dem Sieg der künstlichen Intelligenz AlphaGo über den weltbesten Go Spieler[6]. AlphaGo wurde dafür mit Hilfe von Deep Learning und künstlichen neuronalen Netzen trainiert und stellt ein Paradebespiel für Machine Learning dar.
Doch wie funktioniert eigentlich dieses Machine Learning, bei dem Künstliche Intelligenzen „selber" lernen? Das Gebiet des Machine Learnings lässt sich in drei Teilgebiete unterteilen. Es wird unterschieden zwischen unüberwachtem, überwachtem und bestärkendem Lernen. Im nachfolgenden wird sich mit den zwei letzteren Teilgebieten genauer beschäftigt. Dabei werden die grundlegenden Konzepte erklärt, Schwierigkeiten erläutert und aktuelle Einsatzgebiete aufgezeigt.

2 Überwachtes Lernen

Überwachtes Lernen (*engl. supervised learning*) stellt eine Form des Maschinellen Lernens dar, in der die KI mit Hilfe von Expertenwissen trainiert wird, um Regelmäßigkeiten und Zusammenhänge zu erkennen. Das Expertenwissen liegt in Form von Trainingsbeispielen vor. Diese Trainingsbeispiele sind immer ein Paar (Tupel) von Werten, bestehend aus einem Input, der meistens als Vektor dargestellt wird, und einem Output, auch Überwachungssignal genannt.

Um eine KI mit überwachtem Lernen zu trainieren muss eine große Menge von Trainingsbeispielen vorliegen. Dabei muss darauf geachtet werden, dass die Varianz der Beispiele groß genug ist und genügend Beispiele für jeden möglichen Input vorhanden sind. Sonst kann es passieren, dass Inputs, für die es nicht genügen Trainingsbeispiele gab oder die stark von den Trainingsbeispielen abweichen, nicht korrekt erkannt und somit einem falschen Output zugeordnet werden.
Auf Grund der großen Datenmengen die benötigt werden, stellt das Auszeichnen/Erstellen von Trainingsbeispielen einen großen manuellen Aufwand dar. Daher gibt es mittlerweile Websites die sich darauf spezialisiert haben Daten zu sammeln und auszuzeichnen (zum Beispiel https://www.clickworker.de/). Dazu werden je nach nötigem Expertenwissen auch normale Internetnutzer einbezogen, die dafür in einer Internet Währung oder auch mit echtem Geld bezahlt werden.

[1] https://wearesocial.com/de/blog/2018/01/global-digital-report-2018
[2] https://wearesocial.com/de/blog/2018/01/global-digital-report-2018
[3] https://www.zeit.de/digital/internet/2016-08/kuenstliche-intelligenz-geschichte-neuronale-netze-deep-learning/seite-3
[4] https://de.wikibooks.org/wiki/Computerhardware:_Prozessor
[5] https://www.kas.de/documents/252038/3346186/Vergleich+nationaler+Strategien+zur+F%C3%B6rderung+von+K%C3%BCnstlicher+Intelligenz.pdf/46c08ac2-8a19-9029-6e6e-c5a43e751556?version=1.0&t=1542129691776
[6] https://www.spiegel.de/netzwelt/gadgets/alphago-besiegt-weltbesten-go-spieler-ke-jie-in-go-turnier-a-1148889.html

Bevor die KI genutzt werden kann, muss sie erst einmal trainiert werden. Dazu geht die KI nacheinander die Trainingsbeispiele durch und probiert mit Hilfe eines Lern-Algorithmus den Input des Beispiels auf einen Output abzubilden. Dieser Output wird nun mit dem erwarteten Output des Beispiels verglichen und die KI bekommt ein Feedback auf Grundlage dieses Vergleichs. Mit Hilfe dieses Feedbacks passt die KI ihren Lern-Algorithmus an.
Um zu testen, ob die KI hinreichend trainiert ist, werden weitere bereits klassifizierte Beispiele verwendet. Anhand der hierbei entstehenden Ergebnisse wird entschieden, ob der Algorithmus der KI genau genug ist, oder ob mehr Trainingsbeispiele genutzt werden müssen um den Algorithmus weiter zu trainieren.

Überwachtes Lernen wird meistens für Klassifikationsaufgaben, sowie Funktionsapproximations- oder Regressionsaufgaben genutzt.
Klassifikationsaufgaben bezeichnen dabei Aufgaben, bei denen der Input in bestimme Kategorien oder Klassen eingeordnet wird. So sind zum Beispiel die Bestimmung, ob eine E-Mail Spam ist (Einordnung in die Kategorien „Spam“ und „Nicht Spam“) und die Interpretation von Handschriftlichen Zeichen (Einordnung in die Klassen der verschiedenen Buchstaben und Ziffern) Klassifikationsaufgaben.
Funktionsapproximations- oder Regressionsaufgaben bezeichnen Aufgaben, bei denen das Ergebnis ein beliebiger Wert innerhalb eines definierten Bereichs ist. Diese Aufgaben werden häufig auch als Schätzungen oder Prognosen bezeichnet. Dazu zählen unter anderem Prognosen zur Preisentwicklung bestimmter Regionen oder Häusertypen, Stauprognosen, sowie Schätzungen zum Alter einer Person.

Innerhalb der jeweiligen Aufgabenbereiche gibt es verschiede Algorithmen die als Lern-Algorithmus genutzt werden können.

2.1 Allgemeine Schritte zum Lösen eines Problems mit Hilfe von Überwachtem Lernen

Zu Beginn sollte sich der Nutzer überlegen, was für einen Typ von Daten er zum Trainieren der KI nutzen möchte. Bei der Handschriftenanalyse zum Beispiel müsste der Nutzer überlegen, ob er die KI anhand eines einzelnen Buchstabens, eines ganzen Wortes oder sogar einer ganzen Zeile trainieren will. Danach müssen Trainingsdaten des vorher bestimmten Typen gesammelt oder erstellt werden. Dabei muss darauf geachtet werden, dass die Daten ihr Verhalten in der realen Welt widerspiegeln. Die Ausgaben der Trainingsdaten müssen von Experten oder mit Hilfe von Messungen bestimmt werden. Wenn genügend Trainingsdaten vorhanden sind, muss die Eingabe des Algorithmus bestimmt werden. Die Eingabe wird meistens als Vektor dargestellt und jeder einzelne Werte des Vektors, auch Feature genannt, repräsentiert ein relevantes Merkmal der Daten. Hierbei gilt es genügend Features anzugeben, damit eine präzise Vorhersage für den Output getroffen werden kann. Falls zu viele Features angegeben werden, kann es zum „Fluch der Dimensionalität“ kommen (siehe 2.2.3 Dimension des Inputs). Nachdem der Eingabe-Vektor definiert wurde, ist es an der Zeit sich für einen Algorithmus zu entscheiden. Dabei gibt es einige Aspekte, die bei der Auswahl des Algorithmus beachtet werden sollten. Diese Aspekte werden im nächsten Abschnitt erläutert. Nachdem ein Algorithmus ausgesucht wurde, kann dieser mit den gesammelten Trainingsdaten trainiert werden. Je nach Algorithmus muss der Nutzer noch extra Parameter festlegen, mit deren Hilfe er den Algorithmus optimieren kann. Zuletzt muss überprüft werden, wie akkurat der Algorithmus ist. Dazu wird ein Test-Set genutzt, welches der Algorithmus nicht kennt, bei dem die Ausgabe-Werte bereits von Experten oder Messungen bestimmt wurden.

2.2 Auswahl des Algorithmus

Alle bisher bekannten und genutzten Algorithmen haben Vor- und Nachteile, so dass es keinen „besten“ Algorithmus, der zur Lösung aller Probleme genutzt werden kann, gibt (auch als „No Free Lunch Theorem“ [7] bekannt). Daher muss der Programmierer an Hand seines Nutzungsfalls entscheiden, welcher Algorithmus der Beste ist. Die nachfolgenden Aspekte zeigen, wie sich die Algorithmen unterscheiden und welche Auswirkungen sie auf die Algorithmen haben.

2.2.1 Verzerrungs-Varianz-Dilemma

Dieses Dilemma beschreibt das Problem der gleichzeitigen Minimierung von zwei Fehlerquellen, der Verzerrung und der Varianz.
Ein Verzerrungsfehler beschreibt eine falsche Annahme des Lern Algorithmus und tritt auf, wenn der Algorithmus nicht die richtige Beziehung zwischen den Eingaben und den erwarteten Ausgaben erkennt. Wenn ein Lernalgorithmus, welcher Bilder von Vögeln erkennen und in der Kategorie „Vogel“ einordnen soll, diese nicht in der Kategorie „Vogel“ einordnet, weil dort ein Vogel zu sehen ist, sondern weil Wolken auf dem Bild sind, liegt ein Verzerrungsfehler vor. Dies wird auch als Unteranpassung bezeichnet (siehe Abbildung 1 - Underfitting).
Die Varianz beschreibt die Empfindlichkeit auf Schwankungen in den Trainingsdaten. Falls die Varianz zu hoch ist kann es passieren, dass der Algorithmus zu „kleinlich“ wird und anstatt, alle Vögel als „Vogel“ zu kategorisieren, nur noch Amseln als „Vogel“ einordnet. Dieses Phänomen wird als Überanpassung bezeichnet (siehe Abbildung 1 - Overfitting).

Da ein niedriger Verzerrungsgrad zu einer hohen Varianz führt, sowie eine niedrige Varianz zu einem hohen Verzerrungsgrad, muss ein Kompromiss zwischen beiden Werten gefunden werden.
Das Verändern der beiden Werte kann, je nach genutztem Algorithmus, automatisch stattfinden oder vom Nutzer über Parameter eingestellt werden.

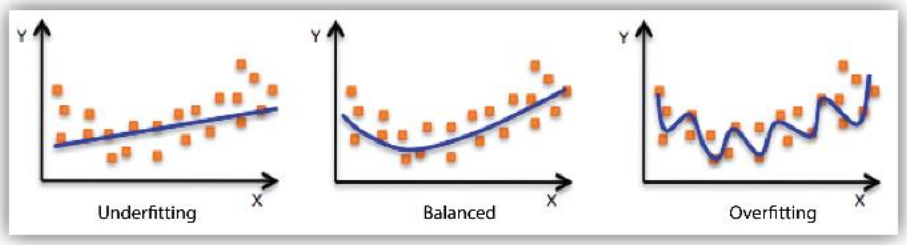

Abbildung 1 Auswirkungen von Über-/Unteranpassung
https://docs.aws.amazon.com/de_de/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

2.2.2 Menge der Daten und Komplexität der abzubildenden Funktion

Die benötigte Menge von Trainingsdaten, sowie der benötigte Verzerrungs-/Varianzgrad sind von der Komplexität der zu modellierenden Funktion abhängig. Versucht man eine einfache, nicht komplexe, Funktion darzustellen, so kann ein „unflexibler“ Algorithmus mit hoher Verzerrung und geringer Varianz, diese aus einer kleinen Menge von Trainingsdaten lernen. Versucht man allerdings eine sehr komplexe Funktion darzustellen, so wird dies nur gelingen, wenn man einem „flexiblen“ Lernalgorithmus mit niedriger Verzerrung und hoher Varianz eine sehr große Menge von Trainingsdaten zum Lernen gibt.

[7] https://en.wikipedia.org/wiki/No_free_lunch_theorem

2.2.3 Dimension des Inputs

Die Dimension des Inputs beschreibt, aus wie vielen Eingabeparametern der Input besteht. Je höher die Dimension des Inputs wird, desto mehr Daten werden benötigt, um den Input mit gleichbleibender Genauigkeit beschreiben zu können. Dieses Problem wird auch als „Fluch der Dimensionalität“[8] beschrieben. Trotz dieser Problematik ist es allerdings wichtig, alle für die Lösung des Problems relevanten Eingabeparameter im Input-Vektor zu definieren, da sonst der Algorithmus nicht die gewollte Funktion erlernen kann.

2.2.4 Ausnahmeerscheinungen in den Ausgabewerten

Auf Grund von menschlichem Versagen kann es dazu kommen, dass falsche Output-Werte innerhalb der Trainingsdaten zu finden sind. Diese können dazu führen, dass der Algorithmus, in dem Versuch die Daten anzupassen, in einer Überanpassung endet. Dies kann nicht nur durch Fehler in den Trainingsdaten passieren, sondern auch, wenn die Funktion die dargestellt werden soll zu komplex für den gewählten Algorithmus ist.
Um dies zu verhindern, ist es sinnvoll die Trainingsdaten vor dem Training manuell oder mit Hilfe von Algorithmen, die auf das Finden von falschen Werten in Trainingsdaten spezialisiert sind, zu prüfen.

2.2.5 Weitere wichtige Faktoren

Wenn darauf geachtet wird, dass die Eingabe aus heterogenen Daten besteht (diskrete (geordnete) Werte, fortlaufende Werte usw.), lassen sich einige Algorithmen einfacher anwenden. Manche Algorithmen, darunter auch die meistbenutzten Algorithmen „Lineare Regression“ und „Logistische Regression“, können nur genutzt werden, wenn die Daten als Zahlen vorliegen und innerhalb des gleichen Intervalls liegen. Außerdem sollte darauf geachtet werden, redundante Daten zu vermeiden. Je nach Algorithmus können redundante Daten zu schlechterer Performance führen.
Die Beziehungen zwischen den einzelnen Werten der Eingabe haben Einfluss darauf, wie gut die Algorithmen funktionieren. Wenn jeder Wert einen unabhängigen Einfluss auf den Output hat, sind lineare Algorithmen normalerweise sehr performant. Falls zwischen den einzelnen Werten komplexe Beziehungen bestehen, können lineare Algorithmen zwar auch genutzt werden, dies erfordert allerdings eine genaue Konfiguration durch den Nutzer. Dort bieten sich eher Algorithmen wie „Entscheidungsbäume“ oder auch „Neuronale Netze“ an, da sie extra zum Aufspüren und Verarbeiten solcher Beziehungen entworfen wurden.

Bei Erschaffen einer neuen KI kann der Nutzer verschiedene Algorithmen ausprobieren und über Tests, zum Beispiel mit Hilfe von Cross-Validation, den bestmöglichen Algorithmus bestimmen. Da das Fine-Tuning eines Algorithmus sehr zeitaufwendig sein kann, ist es meistens sinnvoller stattdessen mehr Trainingsdaten zu sammeln und weitere relevante Werte für die Eingabe zu finden, um bessere Ergebnisse zu erhalten.

[8] https://en.wikipedia.org/wiki/Curse_of_dimensionality

2.3 Algorithmen

2.3.1 Lineare Regression

Die Lineare Regression ist einer der meist genutzten Algorithmen zur Lösung von Funktionsapproximations- und Regressionsaufgaben. Sie nutzt lineare Modelle um die Beziehung zwischen einer „abhängigen“ Variablen, auch Output genannt, und einer oder mehrerer „unabhängiger“ Variablen, auch Input genannt, darzustellen. Bei diesem Verfahren wird eine Gerade zwischen die gegebenen Datenpunkte gelegt und damit versucht, zu unbekannten Inputs die richtigen Outputs zu bestimmen. Die zugrundeliegende mathematische Formel ist:

$$y = ß_0 + ß_1 * x$$ [9]

Dabei steht $ß_0$ für die Regressionskonstante, die angibt welchen Wert der Output haben soll, wenn der Input 0 entspricht. Der Regressionskoeffizient $ß_1$ stellt die Steigung der Regressionsgeraden dar und zeigt, wie stark sich der Output aufgrund des Inputs verändert.

2.3.2 Logistische Regression

Die Logistische Regression baut auf der Linearen Regression auf und ist einer der meist genutzten Algorithmen zur Lösung von Klassifikationsproblemen. Bei diesem Verfahren wird der Zusammenhang zwischen Input und Output mit Hilfe einer Link-Funktion in den Bereich zwischen 0 und 1 skaliert. Das Ergebnis stellt eine Wahrscheinlichkeit dar, die genutzt wird um das Ergebnis zu klassifizieren. So könnte man zum Beispiel bei der Frage, ob sich jemand gesund ernährt, alle Ergebnisse größer 0,5 zu „Ernährt sich gesund“ zuordnen und alle Ergebnisse kleiner oder gleich 0,5 zu „Ernährt sich ungesund“.
Die logistische Regressionsgleichung für den Zwei-Variablen-Fall lautet:

$$p_i = \frac{1}{1+e^{-(ß_0+ß_1*x)}}$$ [10]

Es ist aber auch möglich, in mehr als nur zwei Kategorien einzuordnen. Dazu muss nur eine andere Regressionsgleichung genutzt und die Zuordnung der Wahrscheinlichkeiten auf die Anzahl der Kategorien angepasst werden.

2.3.3 Entscheidungsbäume

Entscheidungsbäume sind ein weiteres Verfahren zur Lösung von Klassifikationsproblemen, aber auch zur Lösung von Regressionsproblemen. Die Entscheidungsbaumalgorithmen unterscheiden dabei zwischen Klassifikations- und Regressionsbäumen. Wie der Name bereits vermuten lässt, liegt diesem Verfahren ein Baum als Struktur zu Grunde. Dieser wird meistens mit der Wurzel oben, also umgekehrt, gezeichnet. Die Wurzel sowie alle Knoten enthalten Fragen, mit Hilfe derer entschieden wird, in welcher der Teilbäume abgestiegen wird. Bei Klassifikationsbäumen stellen die Blätter die Kategorien dar, in die der jeweilige Input eingeordnet wird. Im Zusammenhang mit maschinellem Lernen wird der Entscheidungsbaum nicht vom Nutzer per Hand erstellt, sondern mit Hilfe eines Algorithmus selbstständig generiert. Per Hand erstellte Entscheidungsbäume sind häufig im Feld der nicht selbstlernenden Künstlichen Intelligenzen anzutreffen.

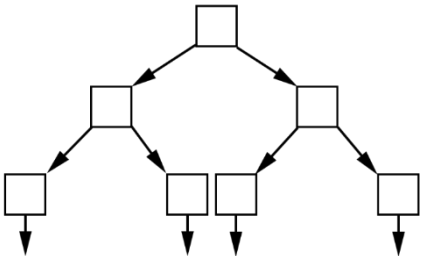

Abbildung 2 Entscheidungsbaum

[9] https://de.wikipedia.org/wiki/Lineare_Regression
[10] https://en.wikipedia.org/wiki/Logistic_regression

2.3.4 Künstliche Neuronale Netze

Eine weitere Möglichkeit zum Lösen von „Überwachtes Lernen“-Problemen stellt das Nutzen Neuronaler Netze dar.
Neuronale Netze bestehen aus einer Input-Schicht, einer Output-Schicht und einer oder mehreren „Versteckten“ Schichten. Die über die Input-Schicht angegebenen Parameter werden innerhalb der versteckten Schichten verarbeitet, so dass an der Output-Schicht ein Ergebnis abzulesen ist.
Was genau dabei innerhalb der versteckten Schichten passiert, lässt sich nur mit erheblichem Aufwand nachvollziehen. Dies ist auch die größte Gefahr bei der Nutzung von Neuronalen Netzen. Durch die versteckten Schichten kann es dazu kommen, dass eine Künstliche Intelligenz, deren Aufgabe es ist Bilder der richtigen Kategorie zuzuordnen, sich an den völlig falschen Merkmalen orientiert. Ein Beispiel dafür ist ein Neurales Netz das darauf trainiert war Pferdebilder zu sortieren, sich nicht an den Pferden, sondern an den Copyright-Angaben am Rand des Bildes orientierte[11].

2.4 Herausforderungen

Künstliche Intelligenzen nehmen jede Veränderung ihrer Umgebung war, auch solche die nicht einmal eine Sekunde dauern und somit von Menschen nicht wahrgenommen werden. Dies kann je nach Bereich ein entscheidender Vorteil sein, aber auch einen Nachteil darstellen, da somit Künstliche Intelligenzen einfach getäuscht werden können.
Eine von Google vorgestellte Künstliche Intelligenz, die in der Lage sein sollte Objekte in Videos zu erkennen wurde getäuscht, indem in das Video alle zwei Sekunden ein Objekt eingebaut wurde. Für Menschen wäre dieses Bild nicht wahrnehmbar, die Künstliche Intelligenz erkennt aber praktisch nur noch das eingefügte Objekt. So klassifizierte diese Künstliche Intelligenz ein Video über Gorillas, in dem ein Teller Spaghetti eingebaut wurde, als ein Video über Pasta[12] [13].

2.5 Einsatzgebiete

Sehr häufig wird Überwachtes Lernen zur Bilderkennung genutzt. Dies geschieht in ganz unterschiedlichen Gebieten. Zum Beispiel in der Medizin, wo Überwachtes Lernen genutzt wird um EKGs automatisch zu interpretieren, Lungenknoten zu erkennen oder das Risiko einer koronaren Herzerkrankung abzuschätzen[14]. Auch Airbus nutzt Bilderkennung in Kombination mit Drohnen, um Mängel an Flugzeugen festzustellen und die Ingenieure darauf hinzuweisen[15]. Außerdem kann Überwachtes Lernen mit Hilfe von alten Daten Prognosen zu Daten der Zukunft erstellen.

[11] https://www.sueddeutsche.de/wissen/kuenstliche-intelligenz-auf-der-falschen-spur-1.3932390
[12] https://arxiv.org/pdf/1703.09793.pdf
[13] https://www.zeit.de/digital/internet/2017-04/kuenstliche-intelligenz-hacker-neuronale-netze-pertubation
[14] https://www.ias.informatik.tu-darmstadt.de/uploads/Publications/Kober_IJRR_2013.pdf
[15] https://www.airbus.com/newsroom/press-releases/en/2018/04/airbus-launches-advanced-indoor-inspection-drone-to-reduce-aircr.html

3 Bestärkendes Lernen

Bestärkendes Lernen (*engl. Reinforcement learning*) ist ein Teil des Maschinellen Lernens, der sich damit beschäftigt, wie sich „Agenten“ in ihrer Umgebung verhalten müssen um eine möglichst hohe Belohnung zu erhalten. Beim Bestärkenden Lernen gibt es nicht zu jedem Input genau einen richtigen Output, es kann mehrere gleich gute Outputs geben. Auch ein nicht perfekt gewählter Output kann zu dem richtigen Ergebnis führen. Es geht vielmehr darum, eine Balance zwischen Erkundung unbekannter Gebiete und Ausnutzung bekannter Gebiete zu finden (siehe 3.4 Erkundung vs. Ausbeutung). Die Umgebung der Künstlichen Intelligenz wird meisten als Markov-Entscheidung-Prozess beschrieben, da dieses Modell besonders einfach von Künstlichen Intelligenzen verarbeitet werden kann. Um etwas mit Hilfe eines Markov-Entscheidungs-Prozesses zu beschreiben muss allerdings die Markov-Eigenschaft erfüllt sein.

3.1 Markov-Eigenschaft

Die Markov-Eigenschaft besagt, dass der nächste Zustand eines Objekts nur vom aktuellen Zustand und der ausgeführten Aktion abhängig ist, nicht aber von den vorherigen Zuständen. Zum Beispiel wird aus einer Urne, in der zwei rote und zwei grüne Kugeln, liegen an 3 aufeinanderfolgenden Tagen jeweils eine Kugel gezogen und daraufhin zurückgelegt. Nachdem wir am zweiten Tag eine rote Kugel gezogen haben, wissen wir, dass unabhängig davon welche Farbe die Kugel hatte, die am ersten Tag gezogen wurde, die Wahrscheinlichkeit eine rote Kugel am dritten Tag zu ziehen bei 50% liegt (siehe Abbildung 3). Die Vergangenheit ist somit irrelevant.
Macht man das gleiche Experiment, nur legt man die Kugel nicht zurück, so würde man am zweiten Tag, nachdem man eine rote Kugel gezogen hat, davon ausgehen, dass die Wahrscheinlichkeit am nächsten Tag eine rote Kugel zu ziehen wieder bei 50% liegt. Weiß man allerdings, dass am Tag zuvor bereits eine rote Kugel gezogen wurde, so liegt die Wahrscheinlichkeit eine rote Kugel ziehen bei 0% (siehe Abbildung 4). Bei diesem Beispiel ist die Vergangenheit relevant und somit ist die Markov-Eigenschaft nicht erfüllt.

Abbildung 3: Beispiel für erfüllte Markov Eigenschaft

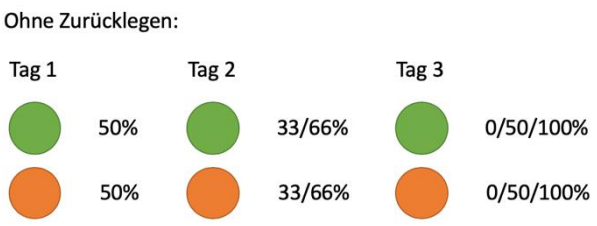

Abbildung 4: Beispiel für nicht erfüllte Markov Eigenschaft

3.2 Markov-Entscheidungs-Prozess

Ein Markov-Entscheidungsprozess ist ein 5-Tupel, bestehend aus einer Menge von Zuständen der Umgebung und des Agenten, einer Menge von Aktionen die der Agent ausführen kann, einer Zustandswechsel-Matrix, einer Belohnungsfunktion, sowie einem Discout-Faktor γ. Die Zustandswechsel-Matrix gibt an, wie hoch die Wahrscheinlichkeit eines Zustandswechsels von Zustand s zu s‘ bei Aktion a ist. Die Belohnungsfunktion definiert die direkte Belohnung für den Wechsel vom Zustand s in s‘ mit Hilfe der Aktion a. Der Discout-Faktor liegt zwischen 0 und 1 und dafür sorgt, dass positive Aktionen nicht ins Unendliche verschoben werden.

Gängige Algorithmen setzen voraus, dass die Mengen der Zustände und der Aktionen endlich sind, auch wenn dies vom Theorem nicht gefordert ist. Die Hauptaufgabe eines Markov-Entscheidungsprozesses ist es, eine Strategie für die Entscheidungen, in welchem Zustand welche Aktion angewandt werden soll, zu entwickeln, so dass die Summe der erhaltenen Belohnungen maximiert wird. Meistens handelt es sich bei der Funktion zum Berechnen der Summe der erhaltenen Belohnungen um eine „discounted sum over a potentially infinite horizon“, welche wie folgt aussieht:

$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1})$$ [16]

Es gibt viele Methoden um die beste Strategie für einen Markov-Entscheidungsprozess zu finden, darunter einige, die die Zustandswechsel-Matrix und die Belohnungsfunktion R kennen müssen, sowie einige, die nur durch Experimentieren die beste Strategie finden.

3.3 Algorithmen

3.3.1 Brute-Force

Das „Brute-Force“-Verfahren ist ein Suchalgorithmus, der durch Ausprobieren das beste Ergebnis findet. Das Verfahren besteht aus zwei Schritten. Im ersten Schritt wird jede mögliche Strategie auf den Markov-Entscheidungsprozess angewandt und das Ergebnis der Strategie gespeichert. Danach wird die Strategie ausgewählt, bei der das beste Ergebnis, also die höchste Belohnung, vorliegt. Da dieses Verfahren jede mögliche Strategie ausprobiert, wird es bei steigender Anzahl an Strategien immer langsamer und ineffizienter. Es kann sogar vorkommen, dass unendlich viele Strategien möglich sind, so dass dieses Verfahren nie ein Ergebnis liefert.
Es gibt Abwandlungen dieses Verfahrens, bei denen mit Hilfe einer Abbruchbedingung das Gebiet der auszuprobierenden Möglichkeiten eingeschränkt wird. Dabei gibt es allerdings meistens keinen perfekten Mittelweg zwischen Geschwindigkeit und Korrektheit des Ergebnisses.

3.3.2 Value-Function

Value-Funktionen versuchen eine Strategie mit maximaler Ausbeute zu finden, in dem sie eine Menge mit den geschätzten Ausbeuten einer Strategie, meistens der aktuellen oder der bisher stärksten, speichern. Diese geschätzten Werte werden bei der Entscheidung genutzt, um die Möglichkeit mit der besten erwarteten Ausbeute zu wählen. Nachdem die Entscheidung getroffen wurde und der Algorithmus die Belohnung für den Zug erhält, wird die geschätzte Ausbeute mit Hilfe der wirklich erhaltenen Belohnung aktualisiert und die Prognosen-Funktion angepasst, so dass Prognosen für die nächsten Züge genauer werden.

3.4 Erkundung vs. Ausbeutung

Beim Lösen von „Bestärkendes Lernen“-Problemen, stellt der „Erkundung vs. Ausbeutung“-Kompromiss eine Herausforderung dar. Dieses Problem ist größtenteils als „Multi-Armed Bandit“-Problem[17] bekannt und beschreibt die Schwierigkeit, die maximale Ausbeute aus seinen Aktionen zu erhalten, ohne zu wissen, welche die besten Aktionen sind. Ausbeutung bezeichnet dabei das wiederholte Anwenden einer bereits bekannten Aktion. Damit bekommt man sicher eine bekannte Ausbeute. Nun kann es allerdings sein, dass der Algorithmus bisher nur die schlechteste Aktion kennt und somit eine viel geringere Ausbeute einsammelt als möglich wäre. Hier kann die Erkundung genutzt werden. Entscheidet sich der Algorithmus dazu eine neue Aktion zu erkunden, wird eine zufällige Aktion gewählt. Diese kann mehr Ausbeute liefern als alle bisher gekannten Aktionen und somit wichtig sein um die maximale Ausbeute zu erzielen. Die Aktion kann allerdings auch weniger Ausbeute liefern als die Aktion die genutzt worden wäre, wenn der Algorithmus nicht erkundet sondern ausgenutzt hätte. Da es kaum Algorithmen gibt, die bei einer größeren Anzahl an Zuständen effizient sind, werden für die Entscheidung, ob der Algorithmus erkundet oder ausnutzt, größtenteils sehr simple Methoden genutzt.

[16] https://en.wikipedia.org/wiki/Markov_decision_process
[17] https://en.wikipedia.org/wiki/Multi-armed_bandit

Eine solche einfache Methode nennt sich „ε-greedy“. Bei dieser Methode kontrolliert eine Variable ε, die zwischen 0 und 1 liegt, die Wahrscheinlichkeit, dass der Algorithmus eine neue Aktion erkundet. Die Wahrscheinlichkeit der Erkundung liegt dabei für jeden Zug bei ε und die der Ausnutzung bei 1-ε.

3.5 Einsatzgebiete

Bestärkendes Lernen ist überall dort zu finden, wo ein Agent mit seiner Umwelt interagiert und auf immer wieder neue Situationen reagieren muss. Eines der bekanntesten Einsatzgebiete dürften Spiele sein. Ob Brettspiele wie Go, bei dem die Künstliche Intelligenz AlphaGo mit Hilfe von bestärkendem Lernen[18] den weltbesten Spieler schlug[19] oder auch Videospiele, bei denen es Google Entwickler schafften, eine Künstliche Intelligenz nur mit Screenshots als Input, Atari Spiele spielen zu lassen[20]. Auch Chatbots, die mittlerweile in großer Zahl existieren, setzen auf Bestärkendes Lernen[21]. Doch auch in der echten Welt findet das Bestärkende Lernen seinen Einzug. Eine Gruppe der Stanford University schaffte es mit Hilfe von Bestärkendem Lernen einen autonomen Helikopter komplizierte Flugmanöver fliegen zu lassen[22]. Sehr bekannt dürfte das Gebiet der selbstfahrenden Autos sein. Auch hier wird Bestärkendes Lernen eingesetzt[23]. Ebenfalls in der Robotik[24], so wie vielen weiteren Gebieten der Informatik ist Bestärkendes Lernen vorhanden.

3.6 Ultimative Künstliche Intelligenz

Alle bisher erfolgreichen Künstlichen Intelligenzen zeichnen sich dadurch aus, dass sie genau für einen Zweck entwickelt wurden. Dadurch schaffen sie es zum Teil in diesem Gebiet besser als Menschen zu sein, in allen anderen Gebieten sind sie allerdings nicht zu gebrauchen. Seit einiger Zeit stellen sich Forscher die Frage was wäre, wenn es eine Künstliche Intelligenz geben würde, die auf allen Gebieten besser als der Mensch wäre? Oder die, wenn sie schon selbst nicht in jedem Gebiet den Menschen übertreffen kann, in der Lage ist sich selbst zu verbessern, bis sie dazu in der Lage ist? Eine solche Künstliche Intelligenz wird als „Ultimative Künstliche Intelligenz“ bezeichnet, als „Superintelligenz“ oder als „technologische Singularität“. Gemeint ist immer das gleiche, eine Künstliche Intelligenz, deren Intelligenz die Menschliche übertrifft. Dies wäre laut Forschern wohl die letzte Erfindung der Menschheit, da danach alle weiteren Erfindungen von einer Künstlichen Intelligenz entwickelt werden würden. Das solch eine Künstliche Intelligenz eine Gefahr für die Menschheit darstellen könnte, wird nicht nur in zahlreichen wissenschaftlichen Artikeln und Büchern, sondern auch in Romanen, wie „Die Tyrannei des Schmetterlings (Frank Schätzing)“ [25] und Videospielen, wie „Horizon – Zero Dawn (Guerilla Games)“[26] aufgezeigt. Doch Forscher sind sich einig, dass wir von einer ultimativen Künstlichen Intelligenz noch sehr weit entfernt sind.

[18] https://hci.iwr.uni-heidelberg.de/system/files/private/downloads/643877180/fallenbuechel_anwendungen-reinforcement-learning-report.pdf
[19] https://www.spiegel.de/netzwelt/gadgets/alphago-besiegt-weltbesten-go-spieler-ke-jie-in-go-turnier-a-1148889.html
[20] https://hci.iwr.uni-heidelberg.de/system/files/private/downloads/643877180/fallenbuechel_anwendungen-reinforcement-learning-report.pdf
[21] https://arxiv.org/abs/1709.02349
[22] https://hci.iwr.uni-heidelberg.de/system/files/private/downloads/643877180/fallenbuechel_anwendungen-reinforcement-learning-report.pdf
[23] https://web.stanford.edu/~anayebi/projects/CS_239_Final_Project_Writeup.pdf
[24] https://www.ias.informatik.tu-darmstadt.de/uploads/Publications/Kober_IJRR_2013.pdf
[25] https://de.wikipedia.org/wiki/Die_Tyrannei_des_Schmetterlings
[26] https://de.wikipedia.org/wiki/Horizon_Zero_Dawn

4 Überwachtes Lernen vs. Bestärkendes Lernen

Da es sich bei Überwachtem Lernen und Bestärkendem Lernen um zwei unterschiedliche Teilgebiete des Maschinellen Lernens mit unterschiedlichen Stärken und Schwächen handelt, kann man beim Vergleich der beiden Gebiete nicht sagen, welches der beiden besser ist, sondern viel mehr, welches für das aktuelle Problem besser genutzt werden kann. Überwachtes Lernen ist am Stärksten, wenn es zum Lösen von Regressions- oder Kategorisierungsproblemen genutzt wird. Dabei werden je nach Komplexität allerdings sehr viele Daten zum Trainieren der Künstlichen Intelligenz benötigt.
Bestärkendes Lernen wird überall genutzt, wo ein Agent (die Künstliche Intelligenz) mit seiner Umwelt interagieren muss. Es werden keine Trainingsdaten benötigt, dafür ist es sehr wichtig eine gute Belohnungsfunktion zu finden.
Oftmals wird auch eine Kombination der verschiedenen Lernarten genutzt um ein Problem zu lösen. Die Gruppe „Obvious Art“ nutzte eine Kombination von Bestärkendem Lernen und Überwachtem Lernen um ein Bild zu erstellen, das so nah an die Originale kam, dass die Künstliche Intelligenz es nicht mehr von einem Original unterscheiden konnte[27].

5 Bibliotheken

Wer sich selber an der Implementation von Maschinellem Lernen versuchen will, kann dafür eine der vielen bereits bestehenden Bibliotheken nutzen. Diese vereinfachen dem Nutzer die Implementation erheblich, da wichtige Teile des Künstliche Intelligenz Gerüsts schon implementiert sind und der Nutzer meisten nur noch Input, Output, sowie kleinere Einstellungen definieren muss. Die großen Frameworks sind dabei auf alle Teilgebiete des Maschinellen Lernens ausgelegt, also können sie sowohl zur Implementation von Überwachtem Lernen als auch zur Implementation von Bestärkendem Lernen genutzt werden.
Das wohl größte Framework für Maschinelles Lernen ist „Tensorflow“[28]. Es wurde von Google entwickelt, ist mittlerweile aber Open Source. Tensorflow ist dabei eine Bibliothek für Python, kann aber auch als Tensorflow.js Variante mit Javascript genutzt werden. Tensorflow wird oft aufgrund der Komplexität als Expertensystem beschrieben. Um es auch Personen mit geringer Erfahrung auf dem Gebiet der Programmierung von Künstlichen Intelligenzen zu ermöglichen z.B. Tensorflow zu nutzten, wurde 2015 „Keras“ geschrieben. Keras ist eine Open Source Deep Learning Bibliothek, die eine einheitliche Schnittstelle für verschiedene Backends, darunter auch Tensorflow, anbietet[29]. Keras vereinfacht die Implementation von Maschinellem Lernen massiv, da viele Schritte die in z.B. Tensorflow manuell vom Programmierer durchgeführt werden müssen bereits automatisch erfolgen.
Eine weitere Bibliothek für Python die nur bestärkendes Lernen unterstützt ist „OpenAI Gym“[30]. OpenAI Gym ist auch Open Source und wird vor allem von der Firma OpenAI genutzt um verschiedene Künstliche Intelligenzen mit bestärkendem Lernen zu testen und zu vergleichen.

[27] https://www.zdf.de/nachrichten/heute/belamy-portraets-pariser-kuenstlergruppe-obvious-hat-erfolg-mit-ki-kunst-100.html
[28] https://www.tensorflow.org/
[29] https://keras.io/
[30] https://gym.openai.com/

Auch wenn Python definitiv die meistgenutzte Sprache für das Entwickeln von Maschinellem Lernen ist, gibt es auch Bibliotheken für andere Programmiersprachen. Eine davon ist DeepLearning4J oder ausgesprochen „Deep Learning for Java“ [31] . Diese Open Source Bibliothek ermöglicht es Maschinelles Lernen in Java zu programmieren. Dabei ist die Bibliothek auch mit Sprachen wie Scala oder Kotlin nutzbar. Weitere häufig genutzte Bibliotheken sind „Caffe“[32], welche mit C, C++, Python und MATLAB nutzbar ist, die von Microsoft bereitgestellte Bibliothek „The Microsoft Cognitive“[33], welche man mit C++ oder Python nutzen kann, sowie „Torch“/„Pytorch“[34], welche mit Lua und Python nutzbar sind und „MxNet“[35], welches man mit Python und C++ aber auch mit R und Julia nutzen kann. Welche der Bibliotheken man für die Implementierung von Maschinellem Lernen nutzen möchte hängt ganz vom Programmierer ab. Relevante Faktoren sind aber sicherlich die bevorzugte Programmiersprache, sowie die bereits gesammelte Erfahrung auf dem Gebiet der Programmierung von Maschinellem Lernen.

[31] https://deeplearning4j.org/
[32] https://caffe.berkeleyvision.org/
[33] https://www.microsoft.com/en-us/research/product/cognitive-toolkit/?lang=fr_ca
[34] https://pytorch.org/
[35] https://mxnet.apache.org/

Literatur-Verzeichnis

Bücher:

Bild der Wissenschaft Spezial (Sommer 2019): „Künstliche Intelligenz: Die Revolution der Roboter“

Kai Walldorf: „Künstliche Intelligenz: Die Kriegsführung, neue Technologien und der Fortschritt der letzten 50 Jahren dieser Macht und Einblick in Robotik“ (1. Auflage - 2019)

Manuela Lenzen: „Künstliche Intelligenz: Was sie kann & was uns erwartet“: C.H.Beck (2. Auflage - 2018)

Websites:

AI United: Entscheidungsbäume für Maschinelles Lernen: https://www.ai-united.de/entscheidungsbaeume-fuer-maschinelles-lernen/ [25.11.2019]

Alexander Thamm: Supervised Machine Learning: https://www.alexanderthamm.com/de/artikel/supervised-machine-learning/ [25.11.2019]

Babeș-Bolyai-Universität Cluj: Maschinelles Lernen und Data Mining https://math.ubbcluj.ro/~csacarea/wordpress/wp-content/uploads/Vorlesung-8.pdf [25.11.2019]

Data Science Blog: Entscheidungsbäume: https://data-science-blog.com/blog/2017/02/13/entscheidungsbaumverfahren-artikelserie/ [25.11.2019]

Deep Learning for Java: https://deeplearning4j.org/ [25.11.2019]

Demis Hassabis: Dieser Mann will die Ultimative Künstliche Intelligenz entwickeln, in Süddeutsche: https://www.sueddeutsche.de/digital/demis-hassabis-dieser-mann-will-die-ultimative-kuenstliche-intelligenz-entwickeln-1.3335109 [25.11.2019]

Eva Wolfangel: Künstliche Intelligenz auf der falschen Spur, in Süddeutsche: https://www.sueddeutsche.de/wissen/kuenstliche-intelligenz-auf-der-falschen-spur-1.3932390 [25.11.2019]

Fraunhofer Institut: Maschinelles Lernen – Kompetenzen, Anwendungen und Forschungsbedarf: https://www.bigdata.fraunhofer.de/content/dam/bigdata/de/documents/Publikationen/BMBF_Fraunhofer_ML-Ergebnisbericht_Gesamt.pdf [25.11.2019]

Free Code Camp: Introduction to reinforcement Learning: https://www.freecodecamp.org/news/an-introduction-to-reinforcement-learning-4339519de419/ [25.11.2019]

Geeks for Geeks: Dynamische Programmierung: https://www.geeksforgeeks.org/dynamic-programming/ [25.11.2019]

Jan-Uriel Lorbeer: Entwicklung von Reinforcement-Learning-Agenten für Strategie-Gesellschaftsspiele: http://edoc.sub.uni-hamburg.de/haw/volltexte/2013/1950/pdf/Bachelorarbeit_EvRLfSG.PDF [25.11.2019]

Keras: https://keras.io/ [25.11.2019]

KIT: Überwachtes Lernen: Klassifikation und Regression: https://dbis.ipd.kit.edu/download/veranstaltung2-20090427.pdf [25.11.2019]

Marutitech: Top 8 Deep Learning Frameworks: https://marutitech.com/top-8-deep-learning-frameworks/ [25.11.2019]

Open AI Gym: https://gym.openai.com/ [25.11.2019]

Research Gate: Logistische Regression: https://www.researchgate.net/publication/237522887_Eine_kurze_Einfuhrung_in_die_Logistische_Regression_und_binare_Logit-Analyse [25.11.2019]

Stack Exchange: Role of Discount Factor in Reinforcement Learning: https://stats.stackexchange.com/questions/221402/understanding-the-role-of-the-discount-factor-in-reinforcement-learning [25.11.2019]

Statistik Nachhilfe: Lineare Regression: https://www.statistik-nachhilfe.de/ratgeber/statistik/induktive-statistik/statistische-modellbildung-und-weitere-methoden/regressionsmodelle/lineare-regression [25.11.2019]

Statworx: Deep Learning: https://www.statworx.com/de/blog/deep-learning-teil-2-programmierung/ [25.11.2019]

TechTarget: Überwachtes Lernen/Supervised Learning: https://whatis.techtarget.com/de/definition/Ueberwachtes-Lernen-Supervised-Learning [25.11.2019]

Tensorflow: https://www.tensorflow.org/ [25.11.2019]

TowardsDataScience: Markov Decision Process: https://towardsdatascience.com/getting-started-with-markov-decision-processes-reinforcement-learning-ada7b4572ffb [25.11.2019]

Trendreport: Überwachtes Lernen: https://www.trendreport.de/wiki/ueberwachtes-lernen/ [25.11.2019]

Uni Potsdam: Reinforcement Learning: https://www.cs.uni-potsdam.de/ml/teaching/ss12/ml2/Reinforcement%20Learning.pdf [25.11.2019]

Very Possible: Machine Learning vs. Neural Networks: https://www.verypossible.com/blog/machine-learning-vs.-neural-networks [25.11.2019]

Wikipedia: Bestärkendes Lernen: https://de.wikipedia.org/wiki/Best%C3%A4rkendes_Lernen [25.11.2019]

Wikipedia: Bias-Variance-Tradeoff: https://en.wikipedia.org/wiki/Bias–variance_tradeoff [25.11.2019]

Wikipedia: Markov Decision Process: https://en.wikipedia.org/wiki/Markov_decision_process https://de.wikipedia.org/wiki/Markow-Entscheidungsproblem [25.11.2019]

Wikipedia: Markov Property: https://en.wikipedia.org/wiki/Markov_property [25.11.2019]

Wikipedia: Reinforcement Learning: https://en.wikipedia.org/wiki/Reinforcement_learning [25.11.2019]

Wikipedia: Superintelligenz: https://de.wikipedia.org/wiki/Superintelligenz [25.11.2019]

Wikipedia: Supervised Learning: https://en.wikipedia.org/wiki/Supervised_learning [25.11.2019]

Wikipedia: Technologische Singularität: https://de.wikipedia.org/wiki/Technologische_Singularit%C3%A4t [25.11.2019]

Wikipedia: Überwachtes Lernen: https://de.wikipedia.org/wiki/Überwachtes_Lernen [25.11.2019]

Abbildungs-Verzeichnis

Abbildung 1: https://docs.aws.amazon.com/de_de/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

Abbildung 2-4: Selber erstellt von Julius Schmidt