Machine Learning strukturiert einsetzen

AUF DER SUCHE NACH DEM PASSENDEN ALGORITHMUS

Entwickler stehen regelmäßig vor der Frage, wie sie KI-Prozesse verbessern können und welcher Algorithmus für bestimmte Fragestellungen jeweils der passende ist. Ein Praxisbeispiel aus der Logistik zeigt, wie die Suche nach dem richtigen Algorithmus strukturiert erfolgen kann.

Mustererkennung, Klassifikation und Entscheidungsfindung – Verfahren des Maschinellen Lernens (ML) können auf vielfältige Weise Prozesse vereinfachen, beschleunigen oder weniger fehleranfällig gestalten. Projektverantwortliche und Entwickler stehen dabei häufig vor der Herausforderung, die jeweils beste Methode für eine Fragestellung zu finden. Erschwerend kommt hinzu: Oft sind die Bestandteile einer ML-Lösung nicht klar voneinander abgegrenzt. So wird meist nicht klar zwischen Algorithmen, Lernstilen und Funktionen unterschieden.

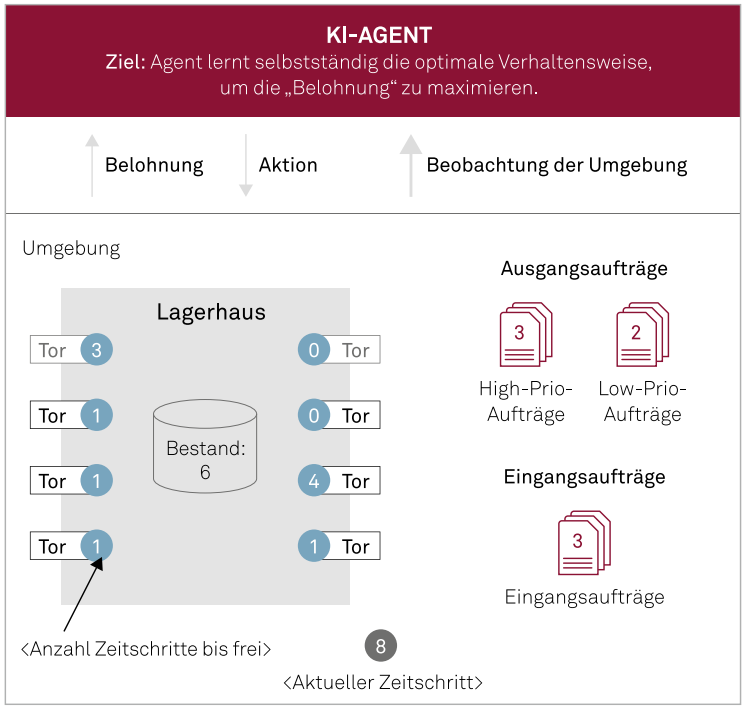

Abb. 1: Ein Lagerhaus-Modell diente als Basis des Machine Learning-Projekts

Außerdem gibt es bislang kaum einen strukturierten Überblick über gängige ML-Methoden und -Algorithmen. Für viele der Algorithmen existieren lediglich mathematische Beschreibungen, die zeigen, wie sich der Algorithmus programmieren ließe. Im Entwicklungsalltag ist dies aber meist zweitrangig, denn in der Regel wird auf bestehende Software-Bibliotheken zurückgegriffen.

Um die Suche nach dem richtigen Algorithmus durch ein strukturiertes Vorgehen zu erleichtern, ist deshalb ein Machine Learning-Katalog hilfreich, der die wichtigsten Begriffe erklärt und definiert, die wesentlichen Algorithmen und Techniken auflistet und sie übersichtlich Funktionsblöcken, Input- und Output-Datentypen sowie Lernstilen zuordnet. Auf diese Weise können Entwickler von jedem dieser Ausgangspunkte schnell geeignete Lösungsansätze identifizieren.

Praxisbeispiel: Warenumschlag in der Logistik

Wie ein solcher Machine Learning-Katalog gewinnbringend in Projekten eingesetzt werden kann, demonstriert ein hier in Folge beschriebenes Praxisbeispiel aus der Logistik, bei dem ein von msg entwickelter ML-Katalog zum Einsatz kam [1]. Ziel des Projekts war die Optimierung des Warenumschlags in einem Lagerhaus. Die dort arbeitenden Disponenten müssen Waren schnell und effizient annehmen, verteilen, lagern, wiederfinden und ausliefern. Zugleich haben sie es mit Betriebsstörungen, Ausfällen von Maschinen und Mitarbeitern sowie mit saisonalen Schwankungen oder Verspätungen

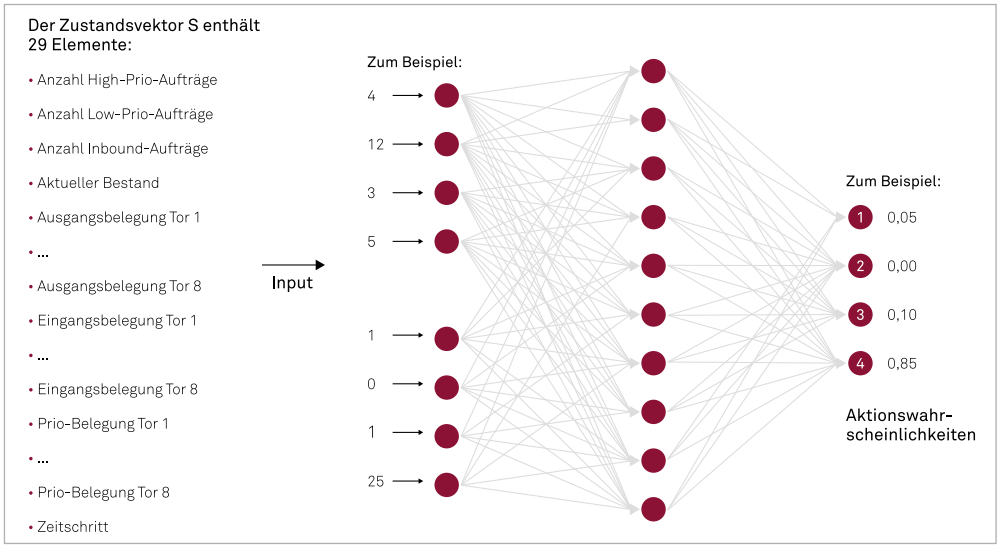

Abb. 2: Neuronales Netz als Entscheidungs-funktion

bei der Anlieferung zu tun. Nur mit jahrelanger Erfahrung ist es möglich, Störungen vorherzusehen und angemessen zu reagieren – ein Wissen, das sich nur schwer auf andere Personen übertragen lässt.

Das Projektteam hat sich deshalb die Frage gestellt, wie sich Teile dieses Know-hows mithilfe Künstlicher Intelligenz abbilden lassen, um Mitarbeiter von Aufgaben zu entlasten und ihnen zu helfen, bessere und schnellere Entscheidungen zu treffen. Am Anfang stand der Entwurf und das mathematische Modell einer auf wenige Funktionen reduzierten Version eines Logistik-Zentrums. Das Modell umfasste Tore für die Warenannahme und den Warenausgang, einen Wareneingang, einen Bestand sowie Ausgangsaufträge mit unterschiedlich hoher Priorität. Die Aufträge benötigten für die Bearbeitung eine definierte Zeit.

Als nächstes setzte das Team den Machine Learning-Katalog ein, um sich einen Überblick über die zur Wahl stehenden Lernstile zu verschaffen. Prinzipiell lassen sich Reinforcement, Supervised und Unsupervised Learning unterscheiden. Zu allen drei Lernstilen bietet der Katalog ausführliche Informationen und eine Auflistung der für diesen Lernstil in Frage kommenden Algorithmen. Ein alternativer Einstiegspunkt in den ML-Katalog wären die Functional Building Blocks. In diesem Projekt sollte Verhalten erlernt werden, sodass der Building Block „Behavioural Modelling“ zum Einsatz kam. Auch hier lieferte der Katalog wieder eine Beschreibung sowie die Verlinkung zu den entsprechenden Algorithmen.

Durch die Einsicht, dass sich das betrachtete fachliche Problem gut als Markow-Entscheidungsprozess (Markov Decision Process, MDP) abbilden ließ, kamen die KI-Spezialisten sehr schnell zur Auswahl des richtigen Lernstils: Reinforcement Learning. Die Umgebung, in der ein Agent agiert, wird im MDP als gerichteter Graph modelliert, in dem jeder Knoten einen Systemzustand darstellt. An jedem Knoten konnte der Agent zwischen verschiedenen Aktionen wählen: nichts tun, Ausgangsaufträge mit hoher Priorität bearbeiten, Ausgangsaufträge mit niedriger Priorität bearbeiten oder Eingangsaufträge bearbeiten. Aktionen, die langfristig zu einem besseren Ergebnis führen, wurden durch eine Belohnungsfunktion verstärkt, Aktionen mit negativen Folgen entsprechend abgeschwächt. Der Agent hatte dabei keine Kenntnisse darüber, wie die Belohnung berechnet wird, sondern lernte allein durch Versuch und Irrtum. Für die Berechnungen kamen ein Python Stack und das ML-Framework Tensorflow zum Einsatz. Die Berechnungen erfolgten auf der Google Cloud Platform unter Verwendung von Grafikprozessoren (GPUs, Graphic Processing Units).

Projektverlauf: Deep Reinforcement Learning als Alternative

Zunächst testete das Team sowohl Monte Carlo Control als auch Temporal Difference Learning als Reinforcement Learning-Methoden. Für das Speichern der erlernten Werte der Q-Funktion verwendeten die Entwickler eine Tabelle zur vollständigen Erfassung aller Zustands-Aktions-Kombinationen. Das sogenannte Q-Learning, eine Form des Temporal Difference Learning, zeigte bei sehr simplen Warenhaus-Modellen eine gute Performance mit einer Laufzeit von zirka zwei Stunden bei einer Million Episoden. Als das Team dann versuchte, die Warenhaus-Simulation durch die Definition weiterer Tore und einer tieferen Auftragswarteschlange komplexer zu machen, passte das kartesische Produkt zwischen allen möglichen Zuständen und allen Aktionen nicht mehr in den Speicher. Somit ließ sich das naive Q-Learning mit tabellarischer Berechnung nicht mehr anwenden. Es war also eine Approximation der Q-Funktion notwendig – zunächst probierten die Entwickler eine gewichtete lineare Funktion aus.

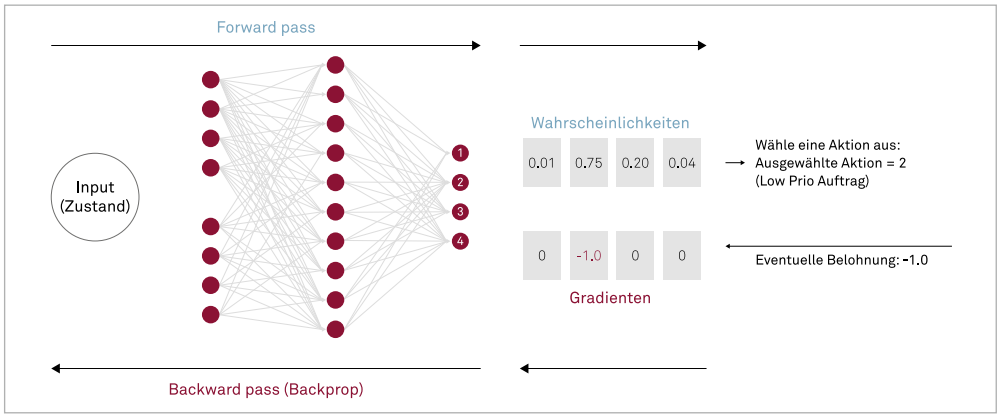

Abb. 3: Training des neuronalen Netzes mittels Policy Gradients und Backpropagation

Es war jedoch äußerst schwierig, eine geeignete Darstellung mittels linearer Funktion (inklusive der Gewichte) zu finden. Außerdem erschien es einleuchtend, dass ein linearer Ansatz schlecht mit der Komplexität der Simulation skaliert.

Daher suchte das Team erneut im Machine Learning-Katalog nach Alternativen für die aufgelisteten Reinforcement Learning-Algorithmen und entschied sich für verschiedene Verfahren des sogenannten Deep Reinforcement Learnings (DRL) – darunter Deep Q-Network (DQN), Dueling Deep Q-Learning (DDQN), Actor-Critic/Advantage Actor Critic (A2C) sowie Policy Gradient Estimation mit einem neuronalen Netz als Funktionsapproximator.

Ergebnisse und Schlussfolgerungen

Wichtigstes Ergebnis des Projekts ist: Die modellierten Logistikprozesse ließen sich mithilfe der im Machine Learning-Katalog aufgeführten Methoden und Algorithmen abbilden. Als Lernstil der Wahl hat sich Reinforcement Learning bewährt, wobei die Policy Gradient Estimation die besten Lernergebnisse erzielte, bei der eine Policy-Funktion die Parameter, welche den aktuellen Zustand beschreiben, auf Aktionen abbildet. Jede Simulation führt dabei zu einem positiv oder negativ bewerteten Ergebnis. Bei einer Belohnung wird die Wahrscheinlichkeit für alle Aktionen in einer Episode erhöht, bei einer Bestrafung reduziert. Das Lernen dauerte mit diesem Algorithmus im Vergleich zu den anderen getesteten DRL-Verfahren zwar deutlich länger. Das Training aber erforderte kaum Hyperparameter-Tuning, und die Ergebnisse waren wesentlich konsistenter.

Weitere wichtige Erkenntnisse des Projekts sind:

Versuch macht klug. Welcher Algorithmus der richtige ist, lässt sich kaum im Vorhinein abschätzen. Das relativ einfach umzusetzende „Policy Gradient"-Verfahren erzielte in diesem Fall das beste Ergebnis. Komplexere Methoden hatten nur deutlich mehr Tuning-Aufwand und Trainingszeit zur Folge, führten aber nicht zu einem erfolgreicheren Agenten. Bei dieser „Trial-and-Error"-Methode erwies sich der ML-Katalog als sehr hilfreich, da er übersichtlich alle passenden Algorithmen und deren Subtypen auflistet und so die Suche nach Alternativen wesentlich beschleunigt.

Die Belohnung macht den Unterschied. Reinforcement Learning kann eine Belohnungsfunktion unter bestimmten Voraussetzungen theoretisch garantiert maximieren. Das lässt sich mathematisch beweisen. In der Praxis können auch Szenarien mit abgeschwächten Voraussetzungen sehr gut funktionieren. Wie sinnvoll das Ergebnis ist, hängt jedoch stark von den Eingangsparametern ab. Bereits minimale Änderungen können erhebliche Auswirkungen auf den Lernerfolg haben.

Kein Fortschritt ohne leistungsfähige Hardware. Bei der Simulation werden Lernzyklen millionenfach durchlaufen. Kleinste Verzögerungen in einzelnen Simulationsläufen multiplizieren sich und können so zu massiven Performance-Einbußen beim Training führen. Es ist daher dringend zu empfehlen, die Simulation selbst sehr effizient umzusetzen sowie ML-optimierte Hardware mit GPUs einzusetzen.

Kein gutes Ergebnis ohne realitätsnahe Simulation. Besonderes Augenmerk ist auf die Definition und Modellierung der Aktionen zu legen. Tätigkeiten, die ein Disponent beispielsweise gleichzeitig ausführt, müssen in eine zusammengesetzte komplexe Aktion überführt oder sequenziell abgearbeitet werden. Darüber hinaus muss die Simulation wie jede Software qualitativ hochwertig, performant und ressourceneffizient programmiert sowie abgesichert und immer wieder angepasst werden. Allgemein gilt: Je realistischer die Simulation, umso realitätsnäher der trainierte Agent, aber auch umso schwieriger und langwieriger das Training.

Literatur & Links

[1] https://machinelearningcatalogue.com/

Autoren

Dragan Sunjka
(dragan.sunjka@msg.group)
ist Senior IT Consultant bei msg. Seit über zehn Jahren entwirft er Lösungen und entwickelt JEE-Individualsoftware. Seine Leidenschaft für KI und ML lebt er auch beruflich aus.

Richard Paul Hudson
(richard.hudson@msg.group)
ist Principal IT Consultant bei msg. Er erforscht neue IT-Trends und berät vor allem zu ML, NLP und Anwendungsarchitektur.