Wie ein Agent die Gridworld durch Q-Learning meistert

Dieses Projekt befasst sich mit der Entwicklung eines autonomen Agenten, welcher mit dem Reinforced Learning Algorithmus Q-Learning trainiert wird, um sich in einer Gridworld zu bewegen. Dabei ist es das Ziel des Agenten ein Endziel vom einem Startpunkt aus zu erreichen und dabei Belohnungen auf dem Weg einzusammeln und Fallen zu vermeiden. In dieser Dokumentation wird dabei die theoretische Grundlage des Q-Learnings, die Implementierung des Agenten und der Umgebung, die verschiedenen Trainingsparameter sowie die erzielten Ergebnisse und eine Analyse des Agentenverhaltens beschrieben.

Einleitung

Die autonome Navigation und Entscheidungsfindung in komplexen oder sich verändernden Umgebungen ist eine der zentralen Herausforderungen im Bereich der künstlichen Intelligenz und Robotik. Reinforced Learning stellt hierfür einen leistungsstarken Rahmen bereit, welcher es Agenten ermöglicht durch die Interaktion mit der Umgebung und durch die Maximierung einer Belohnungsfunktion optimale Verhaltensweisen zu erlenen und solche Umgebungen zu navigieren. Im Gegensatz zu überwachten Lernen, das auf bereits definierte Datensätze basiert, oder unüberwachten Lernen, welches Muster in Daten identifiziert, lernt ein RL-Agent durch Ausprobieren eine eigene Strategie zu erlenen und passt diese basierend auf dem Feedback der Umgebung an. Dies ermöglicht es dem Agenten auch in Situationen zu agieren, für die keine expliziten Programmierregeln, oder bereits vollständige Daten vorhanden sind.

Im Rahmen dieses Projekts wurde ein Agent entwickelt, welcher in einer simulierten Gridworld operiert. Diese Umgebung ist dazu da um die Kernprinzipien des RL zu demonstrieren, bietet aber dennoch eine dynamische und herausfordernde Lernaufgabe. Die Gridworld ist durch eine feste Startposition, von welcher der Agent beginnt, mindestens einer festen Zielposition, welche vom Agenten erreicht werden soll um einen Durchlauf erfolgreich abzuschließen, eine variable Anzahl an Belohnungsfeldern, bei welchen sich die Position jeden Durchlauf zufällig ändern und einer variablen Anzahl an Fallenfeldern, welche ebenfalls zufällig platziert sind, definiert. Der Agent erhält bei dem erreichen des Ziels sowie dem Einsammeln einer Belohnung eine Belohnung, welche jeweils frei definiert werden kann, und bei dem Betreten einer Falle erhält der Agent eine frei definierbare negative Belohnung. Die Kernaufgabe des Agenten besteht darin, eine optimale Strategie zu erlernen, die es ihm ermöglichen soll, seine Aktionen so zu wählen, dass die Belohnung über die Zeit maximiert werden. Dies beinhaltet das effiziente Einsammeln der von Belohnungen, das Vermeiden von Fallen und das Erreichen des Zielzustandes innerhalb von möglichst wenig Schritten innerhalb einer vorgegebenen maximalen Schrittzahl pro Durchlauf. Die zufällige Platzierung der Belohnungen und Fallen zwingen den Agenten, eine generalisierbare Policy zu lernen, anstatt sich nur eine feste Route einzuprägen.

Das primäre Ziel des Projekts ist die erfolgreiche Implementierung und das Training eines Q-Learning-Agenten, welcher in der Lage ist die beschriebene Gridworld autonom und effizient abzuschließen. Die spezifischen Ziele beinhalten die Entwicklung einer robusten Policy, das Erlenen einer Strategie zum Sammeln von Belohnungen zur Punktgenerierung und der Vermeidung von Fallen, der Priorisierung des Erreichens des Zieles um den Durchlauf erfolgreich abzuschließen und die die Demonstration der erlernten Policy.

Das Projekt basiert dabei auf den fundamentalen Prinzipien des Reinforcement Learning und implementiert den weit verbreiteten, modellfreien Algorithmus Q-Learning.

Reinforcement Learning ist ein Paradigma des maschinellen Lernens, bei dem ein autonomer Agent durch Interaktionen mit einer Umgebung lernt eine optimale Verhaltensweise zu entwickeln. Der Lernprozess ist dabei durch eine kontinuierliche Schleife aus Beobachtung, Aktion und Belohnung gekennzeichnet, und folgt dem Prinzip des Lernens aus Erfahrung. Der Agent ist hierbei die lernende Entität, welche die Entscheidungen trifft und die Aktionen in der Umgebung ausführt. Die Umgebung ist die „Welt“ in welcher der Agent agiert. Sie reagiert auf die Aktionen des Agenten, indem sie ihren Zustand ändert und eine Belohnung (oder Strafe) an den Agenten gibt. Dabei befindet sich der Agent immer in einem bestimmten Zustand, wobei es sich um eine vollständige Beschreibung der aktuellen Situation zu einem bestimmten Zeitpunkt handelt. Der Agent nutzt diesen Zustand um seine nächste Aktion zu bestimmten. Bei diesen Aktionen handelt es sich um die vier Grundlegenden Aktionen in einer Gridworld: Hoch, Runter, Links und Rechts. Bei den Belohnungen handelt es sich um einen skalaren numerischen Wert, welcher als Rückmeldung von der Umgebung an den Agenten dient. Positive Belohnungen signalisieren hier gewünschtes Verhalten, während negative Belohnungen unerwünschtes Verhalten signalisieren. Das übergeordnete Ziel des Agenten ist, die kumulative Summe dieser Belohnung über die Zeit zu maximieren. Dabei nutzt der Agent eine Policy, welche die Wahrscheinlichkeit definiert, mit der der Agent in einem bestimmten Zustand eine bestimmte Aktion ausführt. Eine optimale Police ist dabei das Endziel des Lernprozesses, und schreibt vor wie sich der Agent in jeder Situation verhalten soll um die Belohnung zu maximieren. Die Interaktion zwischen Agent und Umgebung wir auch oft als Markov Decision Process (MDP) formalisiert. Ein MDP ist durch die Eigenschaft gekennzeichnet, dass der nächste Zustand und die Belohnung ausschließluch vom aktuellen Zustand und der ausgeführten Aktion abhängen, nicht aber von der gesamten Historie der vorherigen Zuständen und Aktionen. Dadurch ist die Modellierung komplexerer Probleme deutlich einfacher.

Q-Learning ist dabei ein populärer, modellfreier Reinforcement Learning-Algorithmus, der eine sogenannte Q-Tabelle lernt. Modellfrei bedeutet, dass der Agent kein explizites Modell der Umgebung benötigt, das heißt er muss nicht wissen, wie die Umgebung auf seine Aktionen reagiert oder welche Belohnung er erhalten wird. Er lernt dies vollständig durch Erfahrungen und Interaktionen. Die Q-Tabelle ist dabei das Herzstück des Q-Learnings. Die Q-Tabelle ist eine Matrix, in der jede Zeile einem möglichen Zustand (s) und jede Spalte einer möglichen Aktion (a) entspricht. Jeder Eintrag Q(s,a) speichert dabei den geschätzten Q-Wert für das Ausführen der Aktion a im Zustand s. Dieser repräsentiert die erwartete zukünftige diskontierte Belohnung, die erzielt werden kann wenn der Agent eine Aktion in einem bestimmten Zustand ausführt und danach einer optimalen Policy folgt. Ein höherer Q-Wert für solch ein Paar deutet auf eine vorteilhafte Aktion dabei hin. Der Lernprozess ist dabei iterativ und basiert auf der Bellman-Gleichung. Der Q-Wert für ein Zustand-Aktion-Paar wird nach jeder Interaktion aktualisiert, um die Schätzung zu verbessern. Die Gleichung ist dabei: $Q(S,A) < -Q(S,A) + \alpha * [R + \gamma * \max_{A'} Q(S',A') - Q(S,A)]$. Dabei steht Q(S,A) für den aktuellen Schätzwert des Q-Wertes für das Zustands-Aktion-Paar (S,A), vor der

Aktualisierung, α für die Lernrate, welche ein Wert zwischen 0 und 1 ist, und bestimmt, wie stark neue Informationen die bestehenden Q-Werte beeinflussen. R steht für die unmittelbare Belohnung, welche der Agent erhält, nachdem er eine Aktion A im Zustand S ausgeführt hat und in den nächsten Zustand S´ übergegangen ist. γ für den Diskontierungsfaktor, welcher die Bedeutung zukünftiger Belohnungen im Vergleich zu unmittelbaren Belohnungen bestimmt, $\max_{A´} Q(S´, A´)$ ist der maximale Q-Wert, der vom nächsten Zustand S´ aus erzielt werden kann, indem die beste mögliche Aktion A´ im Zustand S´ gewählt wird. Und $[R + \gamma * \max_{A´} Q(S´, A´) - Q(S, A)]$ wird als Tempora Difference (TD) Error bezeichnet und dieser Ausdruck misst die Differenz zwischen der vom Agenten erwarteten Belohnung und der tatsächlich erhaltenen Belohnung basierend auf der maximalen zukünftigen Belohnung berechnet.

Damit der Agent effektiv Lernen kann muss ein Gleichgewicht zwischen Exploration und Exploitation gefunden werden. Dafür ist die Epsilon-Greddy-Strategie. Mit einer Wahrscheinlichkeit von Epsilon wählt der Agent eine zufällige Aktion. Das dient zur Förderung der Exploration und hilft dem Agenten neue Bereiche zu erkunden und möglicherweise besser Strategien zu finden. Dadurch wird verhindert, dass der Agent zu früh in einem lokalem Optima stecken bleibt. Mit einer Wahrscheinlichkeit von (1-Epsilon) wählt der Agent die Aktion mit dem höchsten Q-Wert für den aktuellen Zustand, wobei es sich dann um die Exploitation-Phase handelt, in der der Agent sein aktuelles Wissen nutzt um die Belohnung zu maximieren und seine Policy zu verfeinern. Zu Beginn des Trainings ist Epsilon typischerweise hoch um eine maximale Exploration zu ermöglichen, da die Umgebung für den Agenten noch Unbekannt ist. Im Laufe des Trainings wird Epsilon schrittweise verringert bis es ein zuvor definiertes min-Epsilon erreicht wird. Dadurch wird sichergestellt, dass der Agent anfangs ausreichend experimentiert und im späteren Verlauf eine optimale Policy konvergiert.

Das Projekt wurde in Python implementiert. Dabei wurden die Bibliotheken NumPy für effiziente numerische Operationen (insbesondere die Verwaltung der Q-Tabelle, welche große Matrizenoperationen erfordert) und Pygame für die grafische Visualisierung der Gridworld und des Agentenverhaltens verwendet. Die Implementierung gliedert sich hierbei in drei Hauptklassen, die jeweils spezifische Verantwortlichkeiten übernehmen und miteinander interagieren, um die RL-Umgebung und den Lernprozess abzubilden. Als erstes ist die Gridworld Klasse, welche das Herzstück der Umgebung ist. Sie modelliert die physische Gridworld und definiert die Regeln, nach welche der Agent mit dieser interagieren kann. Dabei wird die Größe der Gridworld, die Start- und Ziel-Felder, Fallen, Belohnungsfelder und deren Anzahl, sowie deren Verteilung auf der Karte und die Belohnungswerte definiert. Als nächstes ist die QLearningAgent Klasse, welche den Kern des Q-Learning-Algorithmus implementiert und die Lernstrategie des Agenten verwaltet. Dabei werden die übergebenen Lernparameter des Agenten und deren leeren Q-Tabelle initialisiert, die Epsilon-Greddy-Strategie implementiert, die Kernmethode der Q-Tabelle sowie deren Umgang des Agenten mit dieser implementiert und das Speichern und Laden dieser. Die dritte Klasse ist die GridworldVisualizer Klasse, welche für die grafische

Darstellung der Gridworld und der Bewegung des Agenten zuständig ist. Diese nutzt die Pygame-Bibliothek, um den Lernprozess und die finale Policy visuell nachvollziehbar zu machen.

Die Definition des Zustandsraums ist ein kritischer Aspekt im tabellarischen Q-Learning, da sie direkten Einfluss auf die Größe der Q-Tabelle und somit auf den Speicherbedarf und die Trainingszeit hat. Eine zu einfache Zustandsdefinition kann dazu führen, dass der Agent nicht genügend Informationen erhält, um optimale Entscheidungen zu treffen, was zu suboptimalen Verhalten führt. Eine zu komplexe Definition hingegen kann den Zustandsraum unnötig groß machen, selbst für kleine Umgebungen. Für dieses Projekt wurde eine erweiterte Zustandsdefinition gewählt, die einen effektiven Kompromiss zwischen Informationsgehalt und der Komplexität darstellt. Dabei wird ein Zustand nicht nur durch die Position des Agenten, sondern auch durch die benachbarten Felder definiert. Dies ermöglicht dem Agenten eine lokale Wahrnehmung seiner unmittelbaren Umgebung, was für Navigationsaufgaben entscheidend ist. Dabei werden die Nachbarfelder in einer kompakten Form kodiert. Einer Repräsentation für ein leeres Feld (0), eines für ein Belohnungsfeld (1) und eines für ein Fallenfeld (2). Der Gesamt-Zustandsindex wird als eine eindeutige Ganzzahl berechnet, die die Agentenposition und die kodierte Nachbarfelder kombiniert. Die Formel berücksichtigt dabei die Breite und Höhe der Gridworld und die Anzahl der möglichen Nachbarfeldtypen. Jede der vier Nachbarzellen kann dabei eine von drei möglichen Typen haben (0,1,2) was zu 81 möglichen Nachbarkonigurationen führt. Diese Kodierung stellt sicher, dass jeder Zustand eindeutig ist, und die Q-Tabelle korrekt indiziert werden kann. Diese Zustandsrepräsentation ermöglicht es dem Agenten, die unmittelbare Umgebung wahrzunehmen und entsprechend zu reagieren. Dabei ist sie ausreichend detailliert um adaptive Verhaltensweisen zu lernen, ohne das der Zustandsraum exponentiell ansteigt. Der Gesamtumgang für einen Zustandsraum für eine 5*5-Gridworld beträgt nach Definition demnach 2025 Zustände.

In diesem Abschnitt wird auf eine spezifische Konfiguration einer Gridworld-Umgebung für einen gewählten Agenten beschrieben und die Hyperparameter des Q-Learnings für diesen Agenten. Anschließend werden die Beobachtungen während des Trainingsprozesses und die Ergebnisse der finalen Evaluation analysiert und die Effektivität der gewählten Strategie bewertet.

Bei der simulierte Gridworld, in welcher der Agent trainiert und evaluiert wurde, handelt es sich um eine 5*5 Gridworld, bei welcher sich der Start unten links in dieser befindet und es ein Ziel gibt, welches sich oben rechts befindet. Es gab 4 Belohnungsfelder und 2 Fallenfelder in dieser. Der Agent wurde mit 2.000.000 Durchläufen trainiert, um den Agenten ausreichend Erfahrung in den zufällig generierten Umgebungen zu ermöglichen und die Q-Tabelle robust und vollständig zu füllen. Dabei hatte der Agent 500 Schritte pro Durchlauf zur Verfügung, um eine ausreichende Exploration zu ermöglichen. Die Lernrate beträgt 0.1 und der Diskontierungsfaktor 0.99. Zum Start beträgt der Epsilon Wert 1.0 und das minimale Epsilon beträgt 0.02. Die Epsilon-Decay-Rate wird durch die Gleichung: (Epsilon_Start – Min_Epsilon) / Num_Epsiodes berechnet, was eine Rate von 0.00000049 pro Durchgang entspricht. Für das Erreichen des Ziels erhält der Agent 1500 Punkte und für das Einsammeln einer

Belohnung 1000 Punkte. Das Betreten einer Falle kostet 10.000 Punkte und jeder Schritt der der Agent tut (jede Aktion) kostet ihm 5 Punkte.

Am Anfang hat der aufgrund des hohen Epsilon Werts eine hohe Exploration gehabt, was in sehr vielen Durchläufen in sehr stark negativen Punktzahlen akkumulierte. In der mittleren Phase hat der Agent angefangen grundlegende Muster in der Umgebung zu erkennen und die Belohnungen zu erkennen. Die durchschnittlichen Punktzahlen stiegen während dieser Phase an, waren aber immer noch überwiegen negativ. Zudem kam es häufig vor, dass der Agent in lokalen Schleifen steckenblieb, was aufgrund der noch nicht vollständigen Q-Werte war. Gegen Ende des Training wurde die Policy des Agenten zunehmend stabiler und deterministischer. Die durchschnittliche Punktzahl wurde positiv, da der Agent gelernt hat direkte Wege zu Belohnungen und zum Ziel zu finden. Die Q-Tabelle hat sich weitgehend stabilisiert, was darauf hindeutet das der Agent eine konsistente Strategie für die meisten Zustände gelernt hatte. Aufgrund der Schrittkosten kam es allerdings immer noch vor dass der Agent in der Nähe von Fallen anfing auf zwei Feldern hin und her zu laufen, da es dort für ihn keine offensichtliche gute Aktion gab, welche er kannte. Das zeigt das die Schrittkosten mit den hohen Fallenkosten dazu führten das der Agent in deren Nähe vorsichtig wurde, besonders wenn keine weiteren Belohnungen oder das Ziel in der Nähe waren.

Nach Abschluss des Trainings wurde der Agent mehrmals für jeweils 10 Durchläufen evaluiert. Während dieser Evaluierung wurde der Epsilon-Wert auf 0.0 gesetzt, um sicherzugehen dass der Agent ausschließlich seine gelernte, deterministische Policy nutzt. Dabei hat sich ergeben, dass der Agent eine Erfolgsquote von ca. 75% aufwies. Die Durchschnittliche Punktzahl lag bei ungefähr 3500, was darauf hindeutet dass der Agent in der Regel 3 der 4 Belohnungen fand, bevor er zum Ziel ging. In den verbleibenden 25% der Durchläufe kam es vor das der Agent an Fallen hängenblieb und in einem lokalem Optima stecken blieb. Bei diesem verbrauchte er die maximale Anzahl an Schritten und hatte im Durchschnitt weniger Punkte, manchmal auch negative. Hierbei handelte es sich um ein Problem, welches der Agent trotz den Trainings nicht vollständig überwinden konnte.

Hierbei wurde erfolgreich ein Q-Learning-Agent implementiert, der in einer definierten Gridworld-Umgebung gelernt hat diese zu bewältigen. Der Agent kann dabei Belohnungen sammeln, Fallen größtenteils vermeiden und das Ziel in einem Großteil der Fälle erreichen. Durch die Implementierung einer dynamischen Umgebung hat der Agent eine generalisierbare Policy entwickelt, welche sich allerdings nicht auf Gridworld-Umgebungen mit anderen Größeneinstellungen anwenden ließ. Allerdings verzeichnete der Agent bei ähnlich strukturierten Umgebungen Erfolg. Die größte Herausforderung des Projekts war die Feinabstimmung der Belohnungsfunktionen um das gewünschte Verhalten zu erreichen. Besonders die Feinabstimmung der Schrittkosten war entscheidend, da eine zu hohe Bestraffung für Schritte häufig zu einem unerwünschten Verhalten führte. Genauso bei zu geringen Schrittkosten. Das verdeutlicht die starke Sensibilität von RL-Algorithmen gegenüber der Belohnungsgestaltung, wo selbst kleine Änderungen starke Auswirkungen auf das gelernte Verhalten haben. Weite Limitationen umfassen die Lokalität der Wahrnehmung, wo die aktuelle Zustandsdefinition die Wahrnehmung des Agenten auf seine unmittelbare Umgebung beschränkt. Hier könnte diese lokale Wahrnehmung unzureichend, bei

komplexeren Umgebungen, wo eine globale Planung, oder das Erinnern notwendig ist, sein. Auch stößt die Skalierbarkeit des tabellarischen Q-Learnings bei sehr großen oder kontinuierlichen Zustandsräumen an schnell an seine Grenzen, hinsichtlich des Speicherbedarf und der Trainingszeit. Außerdem ist die Leistung des Agenten stark von der korrekten Wahl der Hyperparameter abhängig, deren Optimierung oft durch langwieriges Ausprobieren erfolgt.

Dabei kam es während der Durchführung dieses Projekt zu mehreren Erkenntnissen für die Anwendung des Reinforcement Learning. Zum einem ist eine präzise, ausgewogene und gut durchdachte Belohnungsfunktion entscheidend für das gewünschte Verhaltend des Agenten und das Ziel muss klar und deutlich definiert werden. Zudem ist das Gleichgewicht zwischen Exploration und Exploitation entscheiden um sicherzustellen, dass der Agent ausreichend neue Bereiche erkundet, und sein Wissen effektiv erweitert und nutzt. Auch hat die Art und Weise wie die Umgebung für den Agenten kodiert wurde einen massiven Einfluss auf die Komplexität des Problem, die Größe des Zustandsraums und die Trainingszeit. Und ein Iteratives Design ist unerlässlich, da viele Trainingsdurchläufe nötig sind um eine gute Konfiguration zu finden. Um das Projekt zu verbessern, könnte man adaptive Schrittkosten implementieren, welche die Schrittkosten dynamisch anpasst anstatt diese linear zu halten, oder eine Prgressive Difficulty nutzten, welche die Gridworld im Verlauf eines Trainings immer weiter erschwert. Auch könnten andere RL-Algorithmen genutzt werden, wie zum Beispiel das Deep Reinforcement Learning (DRL).

Anhang:

Pulse · HorZer288/Modellierung-Projekt

Quellen:

Bestärkendes Lernen – Wikipedia
Q-learning - Wikipedia
pygame news
Google Gemini