Vorwort:

Machine-Learning ist eines der am meist wachsenden Felder in der Computer Science und rund um die Welt sehr gefragt. Mit den unzähligen Möglichkeiten wie Machine-Learning der Technischen Entwicklung der Menschheit behilflich sein kann ist für diese Technologie auch kein Ende in Sicht. Schon seit ich meinen ersten Utopie Film mit Künstlicher Intelligenz gesehen habe interessierte mich das Thema. Ende des Jahres 2020 hatte ich das Gefühl, dass meine programmier-Kenntnisse genug fortgeschritten waren, um mich dem Thema Machine-Learning anzunehmen. Im Frühjahr 2021 stand dann die Maturaarbeit an und ich musste mich für ein Thema entscheiden. Und ich hatte die Idee meine andere Passion Gaming mit Machine-Learning zu verbinden und mir das Ziel gesetzt meinem PC beizubringen wie man Computerspiele spielt.

Einleitung:

Was ist Rocket League:

Damit ich einem Computer ein Spiel beibringen kann muss es einige Kriterien erfüllen. Um das Projekt im Rahmen der Maturaarbeit fertigstellen zu können muss das Spiel eine sinnvolle Komplexität haben. Auch muss ich das Spiel möglichst einfach über Computerprogramme ansprechen können und auch Spieldaten daraus auslesen können um daraus Trainingssets für das Neuronale Netzwerk gewinnen zu können. Alle diese Kriterien erfüllt RocketLeague das von Psyonix entwickelt wurde. Das Spielprinzip ähnelt einer vereinfachten Form von Fussball indem zwei Teams versuchen mit Autos einen Ball in das gegnerische Tor zu befördern. Dieses Spiel eignet sich sehr gut für solch ein Projekt weil es auf einem immer gleichen Spielfeld stattfindet und die Grundbedingungen immer gleich bleiben was es für ein Neuronales Netzwerk deutliche einfacher macht bedeutsame Entscheidungen zu treffen. Auch wurde ein Framework namens RLBot für Rocket League entwickelt welches einem erlaubt einigen Programmiersprachen unter anderem Python erlaubt RocketLeague Controllerinputs zu geben und einem auch erlaubt die aktuellsten Spielwerte zu erhalten wie zum Beispiel wo sich der Ball oder die Spieler befinden. Dies erlaubt mir möglichst viel Zeit der praktischen Arbeit in die Entwicklung des Bots stecken zu können und somit muss ich mich nicht noch mit anderen Problemen befassen.

Fragestellung:

In meiner Maturaarbeit wollte ich herausfinden, wie gut solch ein RocketLeague Bot im Rahmen der Maturaarbeit werden kann und ob er schlussendlich sogar einen Menschlichen Spieler schlagen kann.

Auch welche Komponenten und Probleme eines solchen Projektes sich effektiv sinnvoll mit Machine-Learning lösen lassen und welche eine weak-ai also eine künstliche Intelligenz die eine spezifische Aufgabe gut kann beziehungsweise ein simples neuronales Netzwerk nicht sinnvoll lösen kann.

Um eine gute Künstliche Intelligenz zu erzeugen ist natürlich unfassbar wichtig, dass man den richtigen neurale Netzwerke typ wählt und fast noch wichtiger, dass man die richtigen Daten wählt um diese zu trainieren. Dazu gehört das Netzwerk nicht zu overfitten keine unnötigen oder unverständlichen Daten in das Netzwerk zu füttern

Zusätzlich finde ich noch interessant herauszufinden welche spielerischen Vorteile ein vom Computer gesteuerter Bot über den Menschen hat.

Begriffe

Hauptteil:

Librarys:

Erstmals benutze ich die RLbot Python library die mir sehr unkompliziert erlaubt meinen code mit RocketLeague zu verknüpfen. Einerseits erlaubt mir RLbot eine Klasse zu verwenden die dessen get\_output() funktion jeden frame läuft und es mir erlaubt darin meine routinen laufen zu lassen, daten direkt aus dem Spiel zu benutzen und schlussendlich einen controllerinput zurück zu geben. Zusätzlich hat diese Library unglaublich praktische debugging featueres die mir erlauben Gegenstände im Spiel zu teleportieren eigene Spielregeln fest zu legen und polygone und text direkt in das Spiel zu zeichnen. Auch liefert mir RLbot eine vorgefertigte Vektor 3 Klasse die ich aber noch leicht angepasst habe um sie optimal nutzen zu können.

Da das programmieren von guten neuronalen Netzwerken sehr viel Zeit in Anspruch nimmt habe ich mich dafür entschieden eine Library namens Tensorflow dazu zu benutzen. Damit kann ich immer Netzwerke verwenden die optimal funktionieren und unkompliziert Änderungen an diesen vornehmen was essentiell dafür ist herauszufinden wieso sich ein Netzwerk nicht so verhält wie es soll.

Um meine Neuronalen Netzwerke optimal überwachen zu können und um sie zu optimieren benutze ich Matplotlib um graphen zu zeichnen. Dies erlaubt mir auch Phänomene wie overfitting zu erkennen oder das Netzwerk zu optimieren, da ich finde, dass die visuelle darstellung dieser Daten deutlich einfacher zu verstehen und zu interprätieren ist.

Zusätzlich benutze ich für das einfachere arbeiten mit den Daten der Neuronalen Netzwerke Numpy. Für den Umgang mit Neuronalen netzwerken ist Numpy sehr praktisch weil es unzählige Funktionen bietet um den Umgang mit Mehrdimensionalen Arrays die in Neuronalen Netzwerken sehr oft vor kommen zu erleichtern.

Aufbau des Bots:

Es war ausschlaggebend einen sinnvollen Aufbau dieses Projektes zu wählen. Es ist wichtig, dass der Aufbau des Projekts vielfältige Spielweise erlaubt aber auch, dass es nicht übermässig viel Arbeit beansprucht, damit das Projekt auch im Rahmen der Maturaarbeit fertiggestellt werden kann. Mit diesen Kriterien im Kopf habe ich mich dazu entschieden folgenden Aufbau zu wählen:

Der Bot hat vier Komponenten. Alle vier Komponenten werden jeden Frame im Spiel also etwa 30 mal pro Sekunde aufgerufen. Ich werde erst den groben Aufbau erklären, damit man alles in einen Kontext setzten kann um nachher tiefer ins technische detail gehen zu können.

Die erste Komponente Wählt eine Strategie aus die im moment passend ist. Die Zweite sammelt Informationen wie man diese Strategie ausführen kann. Die Dritte wandelt die Gesammelten Informationen in einen vom Auto Fahrbaren Pfad um. Die Letzte Steuert den Bot dann effektiv ans ziel und reguliert die Geschwindigkeit

Komponente 1:

Hier wird zuerst überprüft ob seit dem letzten Frame unvorhersehbare Spielereignisse geschehen sind. Dafür werden der vorhergesagte ball pfad und die momentane Ball position verglichen. Wenn abweichungen erkannt werden spricht das dafür, dass sich der Ball anders verhalten hat als geplant oder, dass der Gegner mit dem Ball kontakt hatte.

Auch wird untersucht ob sich der Pfad mittlerweile so verändert hat, dass er bot das Ziel in der Vorgegebenen Zeit nicht mehr erreichen kann.

Wenn dies der Fall ist entscheidet das Neuronale Netzwerke mithilfe der neusten Spielinformationen welche Strategie nun verfolgt werden soll. Auf den Aufbau des Machine Learning teils werde ich aber im nächsten Kapitel genauer eingehen. Auch wird ein Zeitlicher Rahmen festgelgt indem der Bot das Ziel erreichen können sollte.

Falls keine unvorhersehbare veränderung stattgefunden hat wird die Strategie die bisher verfolgt wurde fortgeführt.

Komponente 2:

Bevor ich fortfahre sollte ich erklären was in diesem kontext mit dem Wort «strategie» gemeint ist. Strategien sind einfache argumente wie zb: «verteidige», «greife an» oder «gehe zur Grundlinie». Für jede Strategie habe ich grundwerte festgelegt. Beim Angreiffen sind das beispielsweise die Koordinaten des gegnerischen goals. Da ich ja am anfang jedes Maneuvers einen Zeitrahmen vorgebe indem der Ball erreichbar sein sollte kann ich jetzt voraussagen wo der Ball am ende des Maneuvers ist. Damit und den Koordinaten des Goals kann ich den Richtungsvektor festlegen wo und in welche richtung der Bot am ende des Maneuvers den Ball richtung Ziel schlagen soll.

Komponente 3:

Jetzt soll mit den zuvor gewonnenen Informationen ein für das Auto abfahrbaren weg konstruiert werden.

Dafür verwende ich eine Methode die unter dem Namen Dubins Path oder auch Arc-Line-Arc bekannt ist.

Konstruiert wird solch ein Pfad wiefolgt:

Erst haben wir einen Start Vektor der an der Position des Autos sitzt und in richtung der Momentanen Fahrtrichtung zeigt. Dazu haben wir auch noch einen Zielvektor der beim Ziel also dem Ball sitzt und in richtung der schiessrichtung zeigt. Um beide diese Vektoren können nun Kreise mit dem kurvenradius eingezeichnet werden. Die grösse dieser Kreise Variiert: Je grösser die Geschwindikeit desto grösser ist der Kurvenradius. Zwischen diesen Kreisen können acht Tangenten konstruiert werden. Wenn man jetzt vom Start Vektor ausgeht, finden sich vier mögliche Pfade zum Zielvektor. Jetzt muss evaluiert werden welcher denn am passendsten ist. Generell wird immer der kürzeste genommen, sofern dieser nicht mit einer Wand kollidiert.

Komponente 4:

Der zuvor erzeugte Pfad kann nun abgefahren werden. Da mit der Arc Line Arc Methode immer der engst mögliche kurven Radius gewählt wird kann man jetzt immer zum nächsten punkt also dem nächsten tangentenpunkt oder zielvektor ansteuern und das Auto bleibt automatisch auf dem Pfad.

Der Zweite Teil der diese Komponente übernimmt ist die Geschwindigkeitskontrolle. Da die vorgegebene Zeit für jedes Manöver nur eine Schätzung ist kann der Bot nicht einfach Vollgas geben. Mit der Pfadlänge, und der restlichen zeit kann berechnet werden, wie schnell der bot sein muss. Jeden Frame versucht diese Komponente also sich diesem geschwindigkeitswert anzunähern.

Der «hardcode» teil

Alles was ich bisher aufgeführt habe (bis auf die Entscheidung der Strategie) sind «hardgecoded». Darunter versteht man, dass diese Teile vom Entwickler gänzlich programmiert wurden und dieser code teil nicht die Fähigkeit hat zu lernen. Dieser «hardcode» teil Baut also nur auf unseren Physik und Mathe Kenntnissen auf und hat nichts mit machine learning zu tun.

Schwierigkeiten

Dieser Teil hat unglaublich viel Zeit beansprucht. Es gab viele Probleme, mit denen ich nicht gerechnet habe. Ich möchte hier einige sehr interessante Probleme und meine Lösungen. Viele dieser Probleme sind mathematischer oder Physischer Natur aber meine bisherigen Physik und Mathematik Kenntnisse reichen schlicht nicht aus um diese Probleme zu lösen und deswegen bin ich viele Probleme mit annäherungslösungen angegangen.

Das weitaus grösste Problem, dass mich mehrere Monate beschäftigt hat hatte mit der voraussage zu tun wann das Bot einen sich bewegenden Ball erreichen würde.

Um den bot so effizient wie möglich zu machen, sollte er immer den nächstmöglichen Zeitpunkt wählen der ihm erlaubt den ball in die richtige Richtung zu spielen. Diesen Zeitpunkt zu bestimmen hat sich aber als unglaublich schwierig herausgestellt, weil es sehr schwierig ist genau bestimmen zu können wie lange das Auto nun effektiv hat, um den weg abzufahren. Denn wärend des gesamten Wegs beschleunigt das Auto und je schneller das Auto fährt desto grösser werden die kurven Radien und somit auch die länge des Gesamtwertes.

Die beste Lösung die ich bisher finden konnte war, dass ich für 60 ballpredictions in den nächsten 6 Sekunden einen möglichen Pfad konstruiere und am ende vergleiche welcher dieser Pfade am kürzesten ist. Das ist nicht besonders genau reicht aber aus.