Inhalt

[Vorwort: 1](#_Toc90466615)

[Einleitung: 1](#_Toc90466616)

[Fragestellung: 2](#_Toc90466617)

[Begriffe 2](#_Toc90466618)

[Hauptteil: 2](#_Toc90466619)

[Librarys: 2](#_Toc90466620)

[Aufbau des Bots: 3](#_Toc90466621)

[Der «hardcode» teil 5](#_Toc90466622)

[Schwierigkeiten 5](#_Toc90466623)

[KI Part 6](#_Toc90466624)

[Stärken und schwächen des Bots des bots: 7](#_Toc90466625)

## Vorwort:

Machine-Learning ist eines der am schnellst wachsenden Felder im Bereich 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. 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 Rocket League 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 Rocket League 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 Herausforderungen befassen.

### Fragestellung:

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

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 bezwingen kann aber nicht vielseitig anwendbar ist beziehungsweise ein simples neuronales Netzwerk nicht sinnvoll lösen kann. Dazu gehört auch, dass ich dieses Neuronale Netzwerk mit der mir zur Verfügung stehenden Zeit und der mir zur Verfügung stehenden Hardware trainieren und somit das Projekt vor Abgabe der Maturaarbeit fertigstellen kann.

Um eine gute Künstliche Intelligenz zu erzeugen ist wichtig, dass man den richtigen neurale Netzwerke typ wählt und noch wichtiger, dass man die richtigen Daten wählt um diese zu trainieren. Dazu gehört das Netzwerk einerseits nicht zu overfitten sprich das Netzwerk nicht so gross zu machen, dass es alle Fälle auswendig lernt aber nicht den tiefergreifenden sinn hinter der Problemstellung versteht und andererseits keine unnötigen oder unverständlichen Daten in das Netzwerk zu füttern. Da die Anwendung solcher bedeutungsvoller Daten im bereich des Machine Learning so ausschlaggebend ist wollte ich meine Ergebnisse auch in diesem Bereich festhalten.

Zusätzlich finde ich noch interessant herauszufinden welche spielerischen Vor- und Nachteile ein vom Computer gesteuertes System im Gegensatz zur Menschlichen Spielweise hat und ob sich ein Sinnvolles Verhalten im Spiel für den Menschen und einen Bot unterscheidet.

### Begriffe

…

## Hauptteil:

### Librarys:

Mittlerweile basiert praktisch jede Software die wir im Alltag benutzen auf sogenannten Librarys. Dies sind Vorgefertigte Codebases die jeweils vorgeschriebenen Code enthalten um die Lösung eines Problems zu vereinfachen. Wenn ich also zum Beispiel für die Analyse von Daten einen Graphen anzeigen möchte kann ich das mit einer Library und deren vorgefertigten Funktionen machen, um mir die Arbeit zu vereinfachen.

Die weitaus wichtigste Library die ich benutze ist RLBot für Python. Diese erlaubt es mir meinen code mit Rocket League 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 wie es ein Mensch tun würde. Zusätzlich hat diese Library unglaublich praktische Debugging Features die mir erlauben Gegenstände im Spiel zu teleportieren eigene Spielregeln fest zu legen und Polygone und Text direkt in das Spiel zu zeichnen. Dies erlaubt mir visuell anzeigen zu können was der Bot macht und herauszufinden was unter Umständen nicht wie gewünscht läuft. Auch liefert mir RLBot eine vorgefertigte Vektor 3 Klasse die ich aber noch leicht erweitert 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 dafür zu benutzen. Einerseits kann ich Damit Netzwerke verwenden die optimal funktionieren weil sie auf dem neusten Stand der Technik sind sprich neuste mathematische Techniken benutzen und meine Hardware Leistung optimal ausnutzen und andererseits kann ich damit unkompliziert Änderungen an meinen Netzwerken vornehmen was essenziell 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 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 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 Ballpfad und die momentane Ballposition 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 einer dieser Fälle eintritt, entscheidet das Neuronale Netzwerke mithilfe der neusten Spielinformationen welche Strategie nun verfolgt werden soll. Auf den Aufbau des Neuronalen Netzwerkes werde ich aber im nächsten Kapitel genauer eingehen. Auch wird ein Zeitlicher Rahmen festgelegt 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 zum Beispiel: «verteidige», «greife an» oder «gehe zur Grundlinie». Für jede Strategie habe ich Grundwerte festgelegt. Beim Angreifen sind das beispielsweise die Koordinaten des gegnerischen Goals. Da ich ja am Anfang jedes Manövers einen Zeitrahmen vorgebe, indem der Ball erreichbar sein sollte kann ich jetzt voraussagen wo der Ball am Ende des Manövers ist. Damit und den Koordinaten des Goals kann ich den Richtungsvektor festlegen wo und in welche Richtung der Bot am Ende des Manövers den Ball Richtung Ziel schiessen soll.

#### Komponente 3:

Jetzt soll mit den zuvor gewonnenen Informationen ein für das Auto abfahrbaren weg konstruiert werden. Mit abfahrbar ist gemeint, dass der Bot mit seiner Momentanen Geschwindigkeit jede Kurve abfahren kann ohne von dem Pfad abzuweichen und, dass der Pfad im Spielfeld ist.

Dafür verwende ich eine Methode, die unter dem Namen Dubins Path oder auch Arc-Line-Arc bekannt ist. Dieser erlaubt mir zuverlässig mit immer gleichbleibender Konstruktion einen Weg zu erschaffen, der fahrbar ist. Dieser wird wie folgt konstruiert:

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 sich beim Ziel also meist dem Ball befindet und in Richtung der Schussrichtung zeigt. Um beide diese Vektoren können nun Kreise mit dem Kurven-Radius eingezeichnet werden. Die Grösse dieser Kreise Variiert: Je grösser die Geschwindigkeit 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 den 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 viel Zeit beansprucht. Es gab viele Probleme, mit denen ich nicht gerechnet habe. Ich möchte hier einige sehr interessante Probleme und meine Lösungen aufzeigen. Viele dieser Probleme sind mathematischer oder Physikalischer Natur aber meine bisherigen Physik und Mathematik Kenntnisse reichen schlicht nicht aus um diese Probleme zu lösen und deswegen bin ich viele Probleme mit annäherungs-Lösungen 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 sehr schwierig herausgestellt, weil es 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 Gesamtweges.

Die beste Lösung die ich 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. Die Genauigkeit davon ist nicht perfekt reicht aber ohne Probleme aus.

## KI Part

In diesem und den meisten Projekten in denen Machine-Learning zum einsatz kommt ist es dafür zuständig eine komplexe Aufgabe zu erfüllen, dessen logic der Entwickler unmöglich programmieren könnte weil es zu viele Komplexe zusammenhänge gibt. Das vorzeige Beispiel dafür ist die Unterscheidung zwischen Katze und Hund. Menschen können dies ohne Probleme aber überlege dir mal, wieso wir das können. Mancher wird jetzt beispielsweise denken, dass Katzen spitze Ohren oder ich sage jetzt bewusst dreieckige Ohren haben. Aber was ist denn überhaupt eine Linie aus denen sich offensichtlicher Weise ein Dreieck zusammensetzt. Und so geht es immer weiter. Es ist unglaublich schwer solch einen Sachverhalt mathematisch zu beschreiben und Deswegen wird Machine Learning für solche Probleme verwendet. Genauso ist es mit einer Strategie in Rocket League. Mathematisch zu bestimmen, ob es in einem gegebenen Moment besser ist anzugreifen oder zu verteidigen ist sehr schwierig wenn nicht unmöglich. Die KI entscheidet also anhand den Momentanen Spieldaten welche Strategie der Bot nun verfolgen soll.

Bevor ich aber hier fortfahre, möchte ich einen kurzen Einschub zu KI halten denn um zu verstehen wie solch ein System funktioniert muss man einige Missverständnisse ausbügeln die dem heutigen nutzen mit dem Wort KI zu schulden sind:

Für viele ist KI eine unglaublich mächtige blackbox bei der man nicht genau weiss was sie macht. Durch Film und Kunst driftet das Wort KI im Volksverständnis schon länger von der Realität ab. Aus diesem Grund distanziert man sich in der Software Entwicklung und Forschung schon länger vom Wort KI zum Wort Machine Learning. Denn intelligent ist mein Bot sicherlich nicht wenn ich ihn mit einem Hund vergleiche der heutzutage als einigermassen intelligent gilt. Mein Bot kann eine Tätigkeit gut nämlich aus den Daten die ich ihm gebe eine Strategie zu entwickeln. Man nennt dies auch weak AI. Die trainierte KI die in meinem Bot verwendet wird ist plump ausgedrückt nichts anderes als eine sehr komplizierte Funktion die tausende von Variablen hat die wärend dem Trainingsprozess verändert werden um eine möglichst gute Performance zu erreichen. Wichtig zu verstehen ist auch noch, dass meine KI keine Ahnung hat, was sie macht. Sie probiert einfach Dinge aus und wenn sie funktionieren wird dies gespeichert und in ähnlichen Situationen Wiederholt. Im Gegensatz dazu gibt es auch noch strong ai und darunter versteht man eine AI die sich vielen Umgebungen anpassen kann. Strong Ai’s könnte man intelligent nennen aber die Forschung ist in diesem Gebiet trotz sehr grossem Aufwand noch eher am Anfang eines sehr langen Weges.

Für die Problemstellung ein Spiel mit einer KI zu spielen verwende ich Reinforcement Learning genauer gesagt deep-q-learning. Ein klassisches Neuronales Netzwerk kann sehr gut ein Problem lösen, dass eine eindeutige lösung hat. Also beispielsweise Bilder mit Hunden und Katzen klassifizieren. Aber solch eine Klassische lösung ist sehr schlecht im vorausschauen. Sie kann nicht bewerten ob die Aktion die sie jetzt greade ausführt in der Zukunft gut ist. Sie versucht einfach aus den jetztigen gegebenheiten das beste zu machen was für ein Spiel schlicht nicht funktionniert. Bei deep-q-learning werden ganze episoden also mehrere spielzüge hintereinander bewertet und somit kann man mit sehr viel training einen guten Bot erschaffen.

Aber was versteht man denn unter Training? In meinem Fall versucht mein Bot am anfang einfach zufällig dinge aus bis er zum Beispiel durch Zufall ein goal schiesst. Daraus kann er schliessen, dass in der Situation zuvor die Abfolge der Aktionen die er gemacht hat zu einem erfolg geführt hat. Wenn er also in eine ähnliche Situation gerät wird er das geliche tun. Das ist natürlich stark vereinfacht veranschaulicht aber wieso das Training von solch einer KI so unglaublich lange dauert. Der bot muss tausende von male ausprobieren wie er an eine Situation heran gehen soll bis er die Optimale gefunden hat.

## Stärken und schwächen des Bots des bots:

Ein Bot spielt rocket league sehr anders als ein Mensch. Ein mensch kann deutlich vorasschauender spielen. Einem Bot beizubringen komplexe zusammenhänge beizubringen wie: «wenn der gegner näher am ball ist und er versucht auf mein Tor zu schiessen sollte ich zu meinem Tor um zu verteidigen» ist sehr komplex. Im gegensatz dazu hat der Mensch den Nachteil, dass er Mechanisch nicht so gut spielen kann. Ein Bot kann Theoretisch in diesem Bereich absoloute perfektion erreichen und jeden Ball in das gegnerische Tor treffen, da schlussendlich jeder schuss auf den immer gleichbleibenden game physics basieren.

Die Probleme die sich beim längeren spielen gegen den von mir Programmierten Bot herausstellen sind, dass er nicht gut mit vielen Ball kontakten des gegners klar kommt. Bei jeder Ballberührung des Gegners berechnet der Bot einen neuen Pfad und verliert somit immer sekundenbruchteile. Auch die Wände verwirren ihn und er versucht den Ball immer möglichst von den Wänden fern zu halten.

## Arbeitsprozess

Schon ende 2020 hatte ich mich für das Thema machine learning interessiert und desswegen das Buch Neural Networks From Scratch gelesen, dass einen Einblick in den theoretischeren Teil von Machine learning gibt. Bis März habe ich mich mit dem gesammelten wissen an kleinere Projekte wie bilderkennung oder Aktienkurs-Vorhersage verfestigt und sichergestellt, dass die gelernten dinge auch wirklich sinn machen. Ich habe schnell gemerkt, dass einige Dinge die im Buch sehr logisch geklungen haben auf einmal unverständlich waren. Um diese Konzepte wirklich greifbar zu machen war es sehr hilfreich erst ein bischen mit der Materie umzugehen bevor ich mich an das Maturaarbeites-Projekt begeben habe.

Im April habe ich mich dann angefangen in RLbot einzuarbeiten und habe damit begonnen einen Bot zu programmieren der immer zum nächsten boost pad fährt. Damit habe ich ein Gefühl für das gamepackage also das Objekt, das game informationen enthält, und für eine gute Arbeitsweise mit der Library bekommen.

Im Mai kam der Lernaufwendigste teil nämlich die Programmierung von Reinforcement Netzwerken. Dieses Konzept war für mich doch erst sehr anspruchsvoll aber für einen Bot der in echtzeit Entscheidungen, die in der Zukunft auswirkungen haben, treffen soll führt kein weg an Reinforcement Learning vorbei. Dafür habe ich hauptsächlich die recourcen von pythonprogramming.net benutzt. Diese Recourcen sind vom gleichen Author geschrieben der auch Neural Networks From Scratch geschrieben hat.

Da ich zu diesem Zeitpunkt noch nicht sehr viel darüber wusste welche Herausforderungen ein Neuronales Netzwerk mit meiner Hardware in einem sinvollen Zeitramen bewältigen konnte habe ich die Fähigkeiten des Netzwerkes, das ich für den Bot programmiert habe stark überschätzt. Der erste Ansatz war nämlich die KI versuchen zu lassen die ganze Steuerung zu übernehmen. Ich versuchte dem bot beizubringen sich selbst an eine Bestimmte koordinate zu bewegen und diese im Optimalfall auch in der Richtigen richtung zu überfahren. Aber nach 2 Tagen Trainingszeit musste ich feststellen, dass der Bot keine ahnung hatte was er tat und einfach zufällige aktionen ausführte ohne das gewünschte Ziel zu erreichen.

Daraufhin musste ich mir eine andere Methode für die Navigation des Bots überlegen und der Machine-Learning Teil wurde darauf reduziert, Spiel-Entscheidungen zu treffen und nicht den gesamten Bot zu Steuern. Im Juni begann ich dann mit der Navigationsmethode Arc Line Arc zu der im Kapitel «Aufbau des Bots» weitere Informationen zu finden sind. Dies war für mich eine Komplexe Herausforderung für die ich meine Vektorgeometrie Kenntinse auffrischen musste. Bis im August widmete ich meine Zeit vor allem diesem System und auch den Steuerungssystemen die sehr ausschlaggebend für den Erfolg des Bots sind. Denn wenn diese gut funktionieren wird das Trainieren des Bots deutlich einfacher, da dann weniger falsche trainingsdaten durch Ungenauigkeiten des Bots entstehen. Ende August machte der Bot noch einen riesigen Performance Sprung, denn ich habe eine Ungenauigkeit in den Angaben der Kurvenradien entdeckt, der dazu führte, dass die Pfadlänge ungenau berechnet wurde und somit der Bot somit bei nahen Bällen immer zu früh und bei Bällen die sich weit weg befanden immer etwas zu spät ankam.

Im Oktober und November habe ich nochmals einige Zeit in den Strategie Teil und vor allem darin investiert, dass die Trainingsdaten aussagekräftig sind. Auch passierte mir der bizarre Fehler, dass ich ein Minus bei den Belohnungen vergessen habe und das dazu führte, dass der Bot lernte möglichst schlecht zu werden.

Ende November begann ich dann hauptsächlich kleinere Fehler zu beheben und den Bot abzuschliessen. Damit ich Anfangs Dezember beginnen konnte den Bot trainieren zu lassen.

## Resultate