F A C H H O C H S C H U L E W E D E L

- University of Applied Science -

BACHELOR-THESIS

in der Fachrichtung

Medieninformatik

Thema:

AI Acceleration in Raytracing

Eingereicht von:	Jonas Sorgenfrei Minf101767 Op de Wisch 1 25436 Moorrege Tel. (+49) 04122 / 83420 E-Mail: minf101767@fh-wedel.de
Erarbeitet im:	7. Semester
Abgegeben am:	
Referent (FH Wedel):	Prof. Dr. Christian-A. Bohn Fachhochschule Wedel Feldstraße 143 22880 Wedel Tel. (+49) 04103 / 804840 E-Mail: bo@fh-wedel.de

AI Acceleration in Raytracing

Forschungsarbeit zu Beschleunigungen des Raytrace-Prozesses durch Maschinelles-Lernen

2019, Bachelor-Thesis von Jonas Sorgenfrei, Fachhochschule Wedel

Zusammenfassung:

In dieser Arbeit wird die Anwendbarkeit von maschinellem Lernen auf den geometrischen Abschnitt des Raytracing/Pathtracing untersucht.
Es wird erforscht welchen Einfluss ein maschinell trainiertes Modell auf den Raytrace-Algorithmus nehmen kann und welche Genauigkeit dieses zurückliefert. Im Anschluss wird diskutiert ob sich anhand von Zeit/Genauigkeit eine praktische Anwendbarkeit eines solchen Modells im Raytrace-Prozess realisieren lassen würde und welche Vor- und Nachteile daraus entstehen würden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listing-Verzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
ML	Maschinelles Lernen (engl. Machine Learning)
KI (engl. AI)	Künstliche Intelligenz (engl. Artificial Inteligence)
PBR	Physically Based Rendering
PBRT	Physical Based Ray-Tracer; verweis auf den in [1] beschriebenen Raytracer
RMSE	Root Mean square error (dt. Wurzel der mittleren quadratische Abweichung)
FPS	Bilder pro Sekunde (engl. Frames per Second)
GI	Globale Beleuchtung (engl. Global Illumination)

1.

Einleitung

Im Rahmen dieser Bachelor-Thesis soll erforscht werden in wie weit sich maschinelles Lernen in den Ray-Trace Prozess integrieren lässt. Insbesondere soll die Geschwindigkeit des Raytracing-Vorgangs in Verbindung mit der Qualität des resultierenden synthetischen Bildes analysiert werden.

Maschinelles Lernen findet heut zu Tage Einzug in viele Bereich [2], z.B. Diagnoseverfahren, Bild- und Videoanalysen sowie Sprach- & Texterkennung um einige Beispiele zu nennen. Insbesondere für die automatische Verarbeitung und Analyse von großen Datenmengen wird Maschinelles Lernen als „Schlüsseltechnologie“ [2] gesehen.

Im Gegensatz zu bisherigen Ansätzen der Integration von maschinellem Learning in den Ray-Trace Prozess, soll in dieser Arbeit nicht die Anwendung des maschinellen Lernens auf ein bereits bestehendes Ausgabe-Bild untersucht werden [3], sondern dies direkt in dem Geometrie-Teil (Schnitt-Test) des Raytracing-Prozesses Anwendung finden.

Insbesondere soll untersucht werden ob sich ein sogenanntes (Maschine Learning) Modell auf eine Szene trainieren lässt um eine Aussage, darüber zu treffen ob ein beliebiger Strahl die Szenen Geometrie trifft. Dadurch sollen zum Beispiel Schattenstrahlen schneller bestimmt werden können. Es sollen in dieser Arbeit verschiedene Methoden zur Generierung solcher Modell untersucht werden um eine Aussage treffen zu können, ob sich diese Modelle für den Prozess eigenen und wie sich die Qualität unter diesen unterscheidet.

Als Grundlegende Struktur für den Prozess der Raytracing-Implementation soll der im Buch Phyiscal Based Rendering (Phar et. All) [1] beschreiben Ray-Tracer dienen. Bei diesem handelt es sich um eine generelle Struktur für den Raytracing Prozess, welche sehr viele Thematiken dieses behandelt & implementiert. Die Struktur dieses wurde auch als Basis für den von Pixar entwickelten Renderer RenderMan genutzt und der zu einem der in der Industrie meist verwendeten Renderern gehört.

1.1. Motivation

In der Produktions-Pipeline bei der Erzeugung von computergenerierten Bilder und Animation ist der letzte Schritt (vor der Nachbearbeitung) die Erstellung eines Bildes aus den 3D-Daten/-Kompositionen.

Insbesondere werden in diesem Schritt alle Elemente der Computergrafik kombiniert Modellierte-Meshes, Animationen/Simulationen, Beleuchtung, atmosphärische Effekte etc. Dadurch, dass in diesem Schritt alle Komplexitäten der einzelne Bereich kombiniert werden, wird eine große Rechenpower genutzt und generell liegt hier ein Bottle-Neck.

Abbildung 1 zeigt ein paar Beispiele computergenerierter Bilder, welche mit dem in Maxon Cinema-4D integrierten Renderer gerendert wurde. Bei dieser Szene handelt es sich um eine 1-minütige Animation einer Kamerafahrt gefolgt von einer Auto-Simulation. Es können mehrere Lichtquellen, Schatten und Reflektionen erkannt werden, im Hintergrund kann zusätzlich Nebel erkannt werden. Die Szene wurde mit einer Auflösung von 5760x1080 Pixeln gerendert und einer Framerate von 25 Bildern pro Sekunde. Die Dauer der Erstellung eines Einzelbildes Betrug zwischen 4 und 10 Minuten. Bei einer Frame-Rate von 25 FPS und einer Minute Animation Betrug die gesamte Renderzeit für die 1500 Bilder knapp 140 Stunden.

Abbildung 1 Render-Ergebnisse 3D-Komposition gerendert mit dem Cinema 4D Renderer

Abbildung 2 gibt ein Überblick über die Szenengeometrie der Szene, es kann eine sehr komplexe Flora Geometrie auf beiden Seiten der Straße erkannt werden.

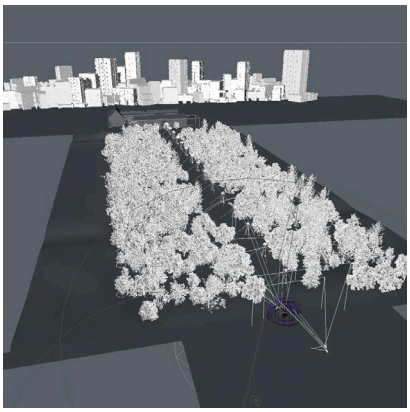

Abbildung 2 Komposition einer 3D Szene im Cinema 4D Viewport

Der Mantra Renderer, integriert in der Houdini Software von Sidefx hat für die Erstellung der Wolkensimulation in Abbildung 3 ca. 2 Stunde gebraucht. Das exportierte Bild hat eine Auflösung von 1280x720.

Abbildung 3 Wolken Simulation in Houdini gerendert mit Mantra

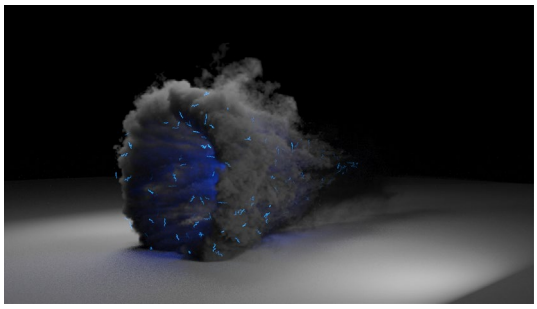

Abbildung 4 Rauch Simulation gerendert mit V-Ray Next

Abbildung 4 zeigt eine weitere Szene, die V-Ray gerendert wurde. Bei diesem Render handelt es sich um einen der meist genutzten Renderer in der Hollywood-Filmindustrie. Für diese HD Szene (1920x1080) benötigte V-Ray Next etwas mehr als eine Minute für den Render-Vorgang, es ist jedoch zu beachten, dass das Ergebnis ohne Multi-Sampling erstellt wurde und ein starkes Rauschen auf dem Boden zu sehen ist, des Weiteren wurde nicht wie in Abbildung 1 eine komplexe Szenen Geometrie verwendet. Wie in der Einleitung beschrieben, erhöht sich die Renderdauer je nach Szenen Geometrie und Qualitätseinstellung des Renderers.

Abbildung 5 Arnold GPU (Beta) Ray-Tracer (Echtzeit-Vorschau)

Abbildung 6 Renderergebnis Pixar Renderman

Abbildung 5 zeigt eine Szene die mit der Beta Version des Arnold GPU Renderers von Solid Angle erstellt wurde und standardmäßig in Maya integriert ist. Diese Szene kann mit dem interaktiven Renderer mit einer Framerate von ca. 1 FPS bei einer Auflösung von 960x540 gerendert werden, es ist zu beachten, dass globale Beleuchtungsphänomene dabei Beachtet werden. Es ist jedoch das starke Rauschen in der Vorschau zu beachten, aber für eine schnelle Echtzeitvorschau für Beleuchtungseinstellung, Shadereinstellungen und Texturierung ist dies ein gutes Werkzeug.

Abbildung 6 zeigt eine ähnliche Szene wie in Abbildung 5, diese Szene in 2 Minuten Renderzeit erstellt. An den Schatten, Reflektionen und Refraktionen kann auch hier der Einsatz von GI erkannt werden.

Sämtliche Szenen wurden selber erstellt und im Rahmen dieser Arbeit gerendert insbesondere um die zeitlichen Unterschiede der unterschiedlichen Bildgenerierungen auf zu zeigen. Das Rendering sämtlicher Szenen wurde mit dem in Abschnitt 3 beschrieben System durchgeführt.

Allgemein kann aus diesen Beispielen erkannt werden, wie hoch die Renderdauer bei der Erstellung dieser Bilder ist.

Zwar haben große Firmen ganze Renderfarmen und besser Hard- und Software für diesen Prozess, jedoch sind auch generell die Szenen entsprechen komplexer.

Um das Licht möglichst realistisch zu simulieren, müssen extrem viele Geometrische Tests durchgeführt werden, einfach Gesagt wird dabei die Sichtbarkeit unterschiedlicher Punkte untereinander getestet. Eine detaillierte Beschreibung befindet sich in Absatz 2.1. Viele dieser Tests sind dabei sehr ähnlich und unter Umständen sogar redundant.

Im Rahmen dieser Thesis wird die Hypothese aufgestellt, dass ein Modell aus dem Bereich des maschinellen Lernens auf eine gegebene Geometrie trainiert werden

kann um diesen Schritt zu beschleunigen.
Ein Beispiel für aktuelle Forschungen bezüglich der Integration von maschinellem Lernen in den Render-Prozess ist z.B. die Filterung von starkem Rausch in partiellen Render-Ergebnissen mittels eines erlernten Netzwerk, das die Fehlenden Pixel berechnet bzw. durch Filterung korrigiert. [3] Z.B. kann mit diesem Ansatz ein Bild wie das der Abbildung 5 verbessert werden.
In dieser Arbeit soll konkret ein eher Geometrischer Ansatz untersucht werden, ob ein Modell die Schnittests mit der Szenen-Geometrie in einer überschaubaren Zeit erlernen kann um im Folgenden Strahlen zu Klassifizieren.
Das Ziel dieser Arbeit ist es, anhand verschiedener trainierter Modelle zu untersuchen, ob ein Modell gefunden werden kann, das ein gutes Ergebnis für die Schnittteste liefert und dessen Einsatzfähigkeit in dem Ray-Trace Prozess zu evaluieren.
Dabei sollen zeitliche Komponenten wie das Erzeugen von Trainingsdaten und das Trainieren des Netzwerkes einbezogen werden sowie die Genauigkeitsabweichung die dieses im Vergleich zu den bisherigen geometrischen Verfahren liefert.
Im Rahmen dieser Überlegung wird vor Beginn der Arbeiten folgende Thesis aufgestellt, die es zu untersuchen gilt.
Ein Modell aus dem Bereich des Maschinellen Lernens kann eine schnelles Ergebnis bei einer Reihe von Bildern einer Szene (z.B. Kamerafahrt) liefern, dessen Ergebnis nicht zu stark von dem geometrischen Ansatz abweicht.

1.2. Struktur

Diese Arbeit ist in 8 Abschnitte unterteilt. Der erste Abschnitt dient dazu einen Überblick über das Thema zu bieten und die Problemstellung sowie die Motivation für diese zu erläutern.
Der 2. Abschnitt soll eine spezifischere Einführung in die Thematik bieten, insbesondere wird in diesem der Raytracing-Prozess erklärt insbesondere wird auf den Teil des Prozess eingegangen, der durch diese Forschungsarbeit betroffen ist.
Des Weiteren wird in dem Abschnitt auf maschinelles Lernen eingegangen und es werden die in diesem Kontext verwendeten Begriffe definiert. In Abschnitt 2.5 wird auf das Framework (Tensorflow) eingegangen, welches verwendet worden ist um verschieden Varianten von Modellen (des maschinellen Lernens) zu erstellen, trainieren und auszuwerten, dies wurden im Rahmen eines schnellen Prototypings

und dementsprechend schnelleren Ergebnissen in Python durchgeführt.
In Abschnitt 3 wird auf die verwendete Hardware eingegangen, die bei der Forschung verwendet wurde und wie sich die Leistung und Auslastung über die Testreihen verhält.
Abschnitt 4 beschreibt die verwendete Szene für die ein Datensatz exportiert wurde für das Training und die anschließende Evaluierung der Modelle. In 4.1 wird auf die generelle Geometrie der Szene eingegangen, in 4.2 wird dargestellt wie die Daten aussehen, die aus dieser Szene exportiert wurden und der Abschnitt 4.3 beschreibt wie diese Daten Unterteilt wurden und analysiert die Verteilung innerhalb dieser Daten.
In Abschnitt 5 werden die Strukturen der Modelle und Parameter der einzelnen Testdurchläufe vorgestellt, für jeden Durchlauf wird der Lernfortschritt über die Trainingsdauer gezeigt und das Training sowie das Finale Resultat jedes Modells analysiert. In den Unterabschnitten 5.1-5.3 sind die Modelle ja nach Modell-Kategorie geordnet beschrieben.
Abschnitt 6 beschreibt das Fazit, welches aus den Testreihen des vorherigen Abschnittes gezogen wurde, insbesondere wird auf die Genauigkeit und die damit verbundene Dauer des Trainings eingegangen.
Abschnitt 7 bewertet die praktische Anwendbarkeit der Ergebnisse in bestehenden Raytracern und Abschnitt 8 gibt ein Ausblick und beschreibt einige Ideen und Hypothesen, die aus der Forschungsarbeit entstanden sind. Dieser als Denoising bezeichnete Ansatz [3] von Chakravarty et al. ist insbesondre bei Path Tracing wie der Monte Carlo Integration von Vorteil, bei der ein sehr große Menge an Berechnungen durchgeführt werden muss (siehe Absatz 2.1).

2.

Basics

Dieser Abschnitt führt in die Thematik der synthetischen Bilderstellung mithilfe von Raytracing ein und gibt ein Überblick über das Thema des maschinellen Lernens. Es werden Begrifflichkeiten definiert, die in den folgenden Hauptteilen der Arbeit benutzt werden.

2.1. Raytracing

Raytracing ist der grundlegende Algorithmus, der in fast allen professionellen Render-Softwares genutzt wird um 3D-Szenen zu rendern die in Softwares wie Houdini, Maya Cinema-4D oder 3ds Max erstellt wurden und dadurch ein Pixelbild zu erstellen.
Zu den bekanntesten Render-Software gehören derzeit Arnold, V-Ray, Render-Man, Octane sowie Redshift [4]. Abbildung 1 bis Abbildung 6 in Abschnitt 1.1 zeigen einige Render-Ergebnisse dieser Softwares, die diverse Techniken verwenden jedoch letztendlich auf den Raytrace-Algorithmus zurückzuführen sind.

Insbesondere wird die Reflexion, Refraktion und Absorption von Lichtstrahlen simuliert um ein möglichst realistisches Bild zu generieren.
Je nach gewünschter Genauigkeit (mit anderen Worten, je physikalisch korrekter die Annäherung sein soll) kann sich die Berechnungsdauer gravierend erhöhen, sodass an dieser Stelle ein Bottle-Neck der synthetischen Bildgenerierung vorliegt.

Dieser Abschnitt soll einen Überblick über den Raytrace-Algorithmus liefern und die damit verbundene synthetische Bilderstellung. Für eine tiefere Beschreibung des Algorithmus, der Implementation und Anwendung ist das Buch „Physical Based Rendering: From Theory to implementation“ von Matt Pharr und Wenzel Jakob [1] sehr zu empfehlen.

Der Raytrace-Algorithmus ist generell die Verfolgung von Strahlen in der Szene. Dabei wird der Strahl z.T. von Objekten reflektiert oder beeinflusst diese, bzw. wird

von diesen beeinflusst.
In vielen Systemen wird von einem Augpunkt ausgegangen, von dem aus die Szene betrachtet wird. Dieser wird oft als Kameraobjekt angesehen. Wie Abbildung 7 zeigt kann das resultierende Bild als eine Bildebene betrachtet werden, die sich vor dem Augpunkt befindet und die Informationen aus der dahinterliegenden Szene beinhaltet, die das Sichtvolumen einschließt. Das Sichtvolumen ist durch die blauen Linien definiert, die vom Augpunkt durch die 4 Ecken der Bildebene gehen. Von der Kamera (Augpunkt) werden Sichtstrahlen in die Szene verfolgt.
Ein Strahl ist geometrisch definiert als:

$$r(\lambda) = O + \lambda\vec{D}$$

Formel 2.1 Geometrische Definition Strahl/Halbgerade; Parametrische Form

In der Formel 2.1 beschreibt O den Ursprungspunkt des Strahls und $\vec{D}$ den Richtungsvektor. Der skalare Parameter λ besitzt einen Wertebereich von [0, ∞] und beschreibt die Skalierung des Richtungsvektors. (siehe Kapitel 2.5 in [1])
Im einfachsten Fall wird beim Raytracing der Augpunkt als Ursprungspunkt genutzt und der Richtungsvektor ist der Vektor, der vom Augpunkt durch den jeweiligen Bildpunkt auf der Bildebene geht. Um ein Bild zu erstellen, muss der Algorithmus über das gesamte Bild iterieren. In Abbildung 7 ist einer dieser Strahlen in Rot dargestellt.

An dieser Stelle beginnt der wichtigste Teil des Algorithmus, denn es muss festgestellt werden, welche Objekte der Strahl in der Szene schneidet. Wenn ein Objekt der Szene durch eine implizite Funktion beschrieben werden kann, kann dieser Schnitttest per Substitution der Strahl Gleichung gelöst werden.

$$\left(O_x + \lambda\vec{D}_x\right)^2 + \left(O_y + \lambda\vec{D}_y\right)^2 + \left(O_z + \lambda\vec{D}_z\right)^2 - r^2 = 0$$

Formel 2.2 Substitution der Strahlgleichung mithilfe des impliziten Kugelgleichung

Diese quadratische Gleichung wird dann für den Parameter Lambda gelöst, z.B. mithilfe eines Gleichungssystems. Sollte das Gleichungssystem nicht mit reellen Zahlen lösbar sein, so schneidet der Strahl die Kugel nicht. Typischerweise wird der Schnittpunkt mit dem geringsten Abstand zur Kamera betrachtet, der mithilfe des Parameters Lambda ermittelt werden kann.
Für jeden Schnittpunkt müssen dabei noch weitere Parameter wie z.B. die

Oberflächennormale und das Material des Objektes bestimmt werden, die für die folgenden Berechnungen benötigt werden.

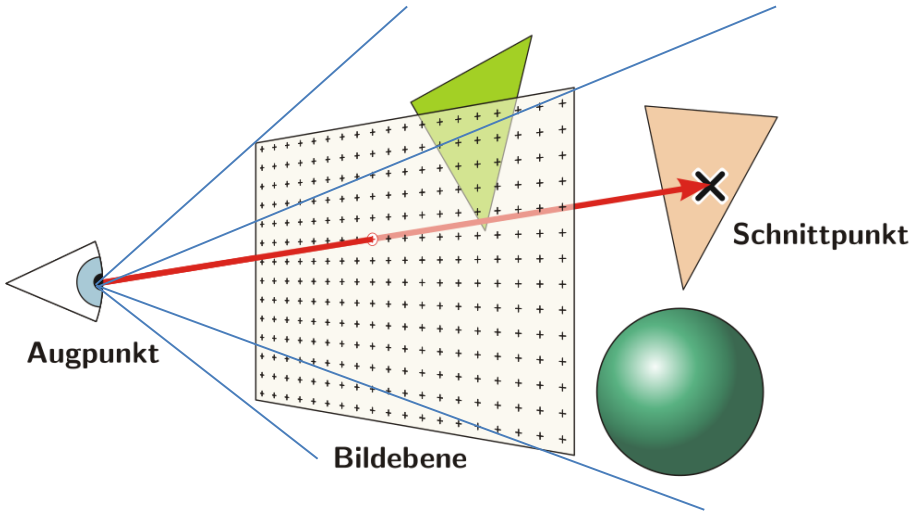

Abbildung 7 Raytracing-Prinzip. Quelle: Wikipedia [5]

Wie durch die Objekte genau iteriert wird, um den Schnittpunkt mit dem kleinsten Abstand zu finden wird genauer in Abschnitt 2.3 erklärt, da dies der entscheidende Punkt für diese Arbeit ist.

Damit Objekte in der Szene sichtbar sind muss ein Ray-tracer Lichtquellen simulieren und dessen Einfluss auf die Objekte berechnen. Dies ist deshalb - entscheidend, da die Menge an Licht bestimmt werden soll, die von dem berechneten Punkt, Richtung Augpunkt (Abbildung 9; ω_0) geht und entsprechend des verwendeten Farbmodells in Farben umgerechnet werden kann. (z.B. RGB-Farbe)
Da es sich bei Licht um eine Form von elektromagnetischer Strahlung handelt und einen Energie-Transport darstellt können Beobachtungen und Gesetze aus der Radiometrie in diesem Schritt Anwendung finden. Um zu bestimmen, ob und wie viel Einfluss eine Lichtquelle auf den Punkt hat, muss die Sichtbarkeit untereinander geprüft werden. Für diese Prüfung kann wieder ein Strahl (Schattenstrahl) zu den Lichtquellen verfolgt werden (Abbildung 8) und es kann auf Verdeckung von Objekten zwischen der Lichtquelle und dem Objekt geprüft werden[1].

[1] In der realen Welt gibt es keine Punktlichtquellen, sondern lediglich geometrische Objekte die Licht emittieren. Der einfachheitshalber wird hier an dieser Stelle eine Approximation durch eine Punktlichtquelle angenommen, welche Licht gleichmäßig in alle Richtungen ausstrahlt.

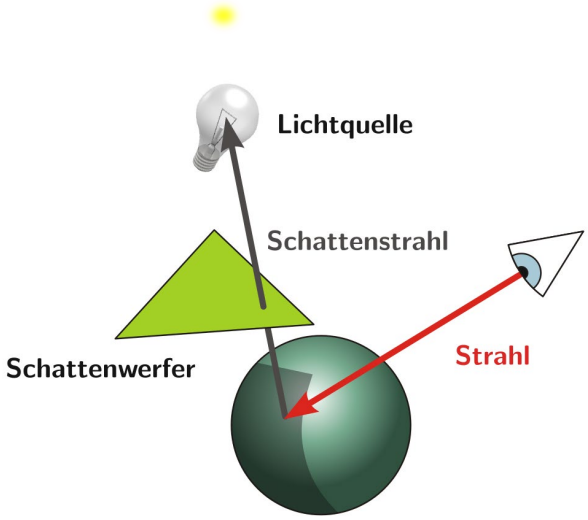

Abbildung 8 Sichtbarkeit und Schatten beim Raytracing. Wikipedia [5]

Als weiteren Punkt muss beachtet werden, dass die Energie über die Länge eines Strahls abnimmt, da es sich in den meisten Szenen nicht um ein Vakuum handelt, in dem sich der Strahl befindet. Dadurch können atmosphärische Effekte wie Nebel und Rauch generiert werden. Die Energie die von einem Licht an einem Punkt P ankommt, nimmt mit der quadratischen Distanz (r in Abbildung 9) ab.

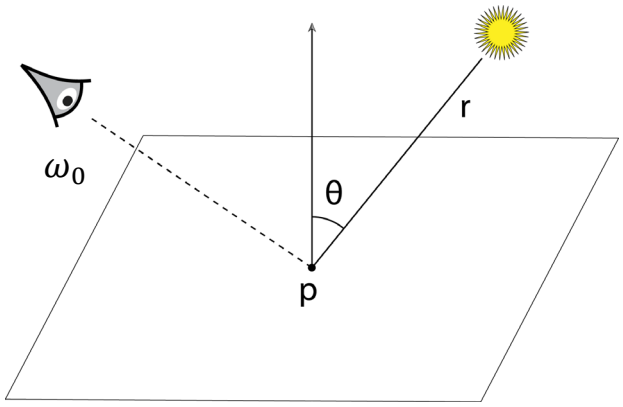

Abbildung 9 Geometrische Herleitung des Lichtanteils, der von der Lichtquelle zum Punkt P ausgesendet wird. Quelle: Physical Based Rendering [1]

Des Weiteren kann gezeigt werden, dass das Licht, welches denk Punkt P (bzw. den kleinen Oberflächenausschnitt um P) erreicht proportional zu *cos(θ)* ist.
θ beschreibt den Winkel zwischen der Oberflächennormalen an dem Punkt *P* und dem Vektor der vom Punkt *P* zu der Lichtquelle zeigt.

Wenn eine Sichtbarkeit zwischen dem Objekt und der jeweiligen Lichtquelle besteht, kann mithilfe der Oberflächeneigenschaften (z.B. Material, Medium, Emissions Charakteristik) bestimmt werden, wie sich das Licht auf dieses auswirkt. Dadurch kann bestimmt werden, welcher Lichtanteil in Richtung der Kamera geht (Abbildung 9; ω_0).
Zur Berechnung dieses Anteils der ausgehenden Energie von einer eingehenden Richtung (Lichtquelle zu Punkt P) kann die Bidirectional Reflectance Distribution Function (BRDF) verwendet werden. Bei der BRDF handelt es sich um eine 4-dimensionale Funktion. Eine allgemeinere Form stellt die die Bidirectional Scattering Distribution Function (BSDF)[2] dar.

Bei mehreren Lichtquellen kann der jeweilige Anteil aufaddiert werden um den Gesamtanteil an Licht für den betrachteten Ausschnitt (Punkt P) zu bestimmen.

Neben dem beschriebenen direkten Lichttransport bietet ein Ray-tracer den Vorteil die globale Beleuchtung zu beachten, daher auch den indirekten Lichttransport, der durch indirekte spiegelnde Reflexionen und Transmissionen stattfindet.

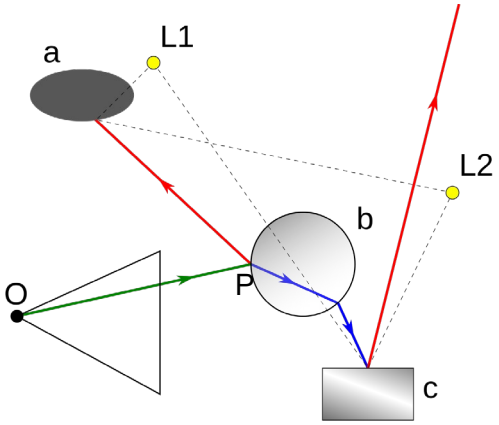

Abbildung 10 Rekursives Raytracing. Quelle: Wikpedia [5]

Abbildung 10 zeigt den rekursiven Strahlenverlauf. Objekt A stellt ein opakes, nicht reflektierendes Objekt dar.
Objekt b stellt dabei ein ideal semi-transparentes und spiegelndes Objekt in der Szene dar, welches einen Brechungsfaktor ungleich 1 besitzt.
Objekt C stelle ein spiegelndes Objekt dar. L1 und L2 sind jeweils Punktlichtquellen.

[2] Von der BSDF gibt es eine Reihe von varianten, z.B. wird für die Berechnung der Lichtstreuung bei Volumen die Bidirectional Scattering Surface Reflactance Distribution Function (BSSRDF) verwendet.

Ein Strahl der von Augpunkt O verfolgt wird, trifft in diesem Beispiel auf das Objekt *b*, bei dem zum einen der Reflexions- und der Transmissions-Strahl[3], da sich der gesamte Lichtanteil aus der Reflexion und der Transmission bestimmt wird, wird die direkte Beleuchtung an dieser Stelle nicht bestimmt werden.
Der reflektierte Strahl ist in Rot dargestellt, der transmittierte Strahl in Blau. Jeder dieser Strahlen wird im folgenden Schritt rekursiv weiterverfolgt. Für Objekt *a* und *c* wird zusätzlich der direkte Lichtanteil für jede Lichtquelle bestimmt (die gestrichelten Linien).
Der an dem initialen Schnittpunkt vorhandene Lichtanteil ergibt sich somit aus den indirekten Anteil der weiteren Szenen Geometrien. An diesem Beispiel wird für die Bestimmung des Lichtanteils an der Position P der initiale Sichtstrahl (grün), 2 reflektierte Strahlen (rot), 2 transmittierte Strahlen (blau) sowie 4 Schattenstrahlen (grau gestrichelt) benötigt.
Abbildung 5 und Abbildung 6 zeigen jeweils Beispiel von spiegelnden und semi-transparenten Objekten.
Diese Beobachtung führt zu der Rendering Equation in Formel 2.3:

$$L_0(P, \omega_0) = L_0(P, \omega_0) + \int_{S^2} f(P, \omega_0, \omega_i) L_i(P, \omega_i) |cos\theta_i| d\omega_i$$

Formel 2.3 Rendering Equation

Der ausgehende Lichtanteil von Punkt P $L_0(P, \omega_0)$ in die Richtung ω_0 ergibt sich aus der Summe des emittierenden Lichtes der Oberfläche $L_0(P, \omega_0)$ und der einkommenden Lichtanteile $L_i(P, \omega_i)$ aus allen Richtungen der Kugel S^2 um den Punkt P skaliert durch die BSDF $f(P, \omega_0, \omega_i)$ und den Kosinus Term.

An Formel 2.3 kann erkannt werden, dass aufgrund des Integrals über die Kugel, das Problem in der Regel. nicht analytisch gelöst werden kann, sodass dies durch verschiedene Ansätze approximiert werden muss[4].
Trotz dieser Approximationen verbleibt in der Regel eine große Anzahl an Schnitttests die im Rahmen des Raytracing Algorithmus durchgeführt werden.
Generell kann erkannt werden, dass viele der Strahlen sehr ähnlich sind bzw. es sogar in einigen Fällen identische Strahlen geben könnte.

[3] Um den reflektierten bzw. transmittierte Strahl zu bestimmen wird das Snelliussche Brechungsgesetze verwendet, mithilfe der Reflexions-/Brechungindizes und der Oberflächennormalen an Punkt P.

[4] Eine tiefere Analyse dieser Methoden würde den Rahmen dieses Überblicks sprengen und wurde deshalb hier vernachlässigt.

Ausdiesem Grund ist es elementar, dass die Implementation der Schnitttests mit der Geometrie effizient ist. Neben verschiedenen Datenstrukturen zur Beschleunigung (siehe Abschnitt 2.3), wird der Render-Prozess durch Ausnutzung der Hardware wie besondere CPU-Befehlsinstruktionen, Multi-Threading oder GPU-Berechnungen beschleunigt.

Wie in der Einleitung beschrieben, soll diese Arbeit einen Ansatz untersuchen, um genau diesen Schritt des Algorithmus zu beschleunigen.
Dies soll durch das Training eines Modells mit vorhanden Strahlen und Schnitttest Ergebnissen geschehen. Im Rahmen dieser Arbeit soll das Modell lediglich unterscheiden, ob ein gegebener Strahl die Szenengeometrie trifft oder nicht. Hauptsächlich soll dieser Ansatz helfen Schattenstrahlen zu klassifizieren.
Eine detailliertere (spezifischere) Ausgabe des Modells, wie Geometrie (-ID) oder Schnittpunkt-Parameter wären auch denkbar, aber für diese wird angenommen, dass es zu komplex für ein Modell werden würde und sich somit nicht eignen würde.

2.2. PBRT Rendering System

Physical based Rendering ist eine Technik, bei der Algorithmen verwendet werden die versuchen das Render-Ergebnis möglichst nahe zu der realen physikalischen Welt zu erzeugen. Insbesondere ist damit eine sehr komplexe Lichtberechnung verbunden die sich sehr stark an der realen Radiometrie orientiert.

Das pbrt System ist eine kompletter Ray-tracer dessen Implementierung in dem Buch Physical Based Rendering: From Theory to Implementation, von Matt Pharr und Wenzel Jakob [1] geschrieben wurden. Das System implementiert alle modernen Render Techniken die auch in den meisten kommerziellen Ray-tracern zu finden sind. Es handelt sich dabei um einen Objekt-Orientierten Ansatz mit C++ als Implementationssprache. Das System ist darauf ausgelegt eine einfache Schnittstelle für neue Implementationen zu bieten und sich dadurch einfach erweitern lässt.
In den Basis Klassen können die in Absatz 2.1 beschriebenen Teile eines Standard Raytracer gefunden werden. (z.B. Camera, Shape, Material etc.)
Im Rahmen dieser Arbeit wurde lediglich eine kleine Komponente beim pbrt ergänzt, ein Modul zum Export von Schnittests aus eine Szene in eine CSV-Datei. Insbesondere musste dafür beachtet werden, dass das pbrt System hoch parallel arbeitet (arbeiten kann) und dies beim Schreiben in die CSV-Datei als atomarer

Vorgang beachtet werden muss.

2.3. Acceleration Alogrithms

In einem naiven Ansatz, wird für jeden Strahl der getestet werden muss die gesamte Szenengeometrie durchgegangen werden. Wenn jedoch Szenen vorliegen wie aktuelle Produktionsstandards von Firmen wie Disney kann schnell erkannt werden, dass dieser Ansatz wenig effizient ist. Abbildung 11 zeigt ein Beispiel aus dem 2016 erschienenen Disney Film Vaiana (engl. Moana).

Abbildung 11 Disney Moana Island Szene
Quelle: Walt Disney Animation Studios (WDAS) [6]

Die Szene repräsentiert viele Herausforderungen bei aktuellen Produktionen. Die Szene beinhaltet viele Instanz Objekte (Bäume, Blatter etc.) in einer sehr großen Szene und komplexen Lichttransport (z.B. beim Wasser und den Wolken).
Eine genauere Beschreibung der Szene kann in dem 2018 vom Walt Disney Animation Studio veröffentlichen Paper [6] zu der Szene gefunden werden.

Durch die in Abschnitt 2.1 beschriebe Rekursion, die beim Raytracing stattfindet kann dies in solchen Szene extrem zeitaufwändig werden wenn für jeden Strahl die komplette Szenen Geometrie getestet werden muss. Es ist fest zu stellen, das jeder Strahl in der Regel nur wenige Objekte in der Szene schneidet und die anderen weit verfehlt. Um Szenen in solchen Dimensionen rendern zu können wird eine Verbesserung dieser Schnittests beim Raytracing benötigt.

Sogenannte Beschleunigungsstrukturen (engl. Acceleration structures) soll den Ray-Trace Prozess beschleunigen, indem eine effizientere Traversierung der Szene bei

den Schnitttesten Gewählt wird. Anstelle jeweils alle Objekte der Szene testen zu müssen, kann mithilfe dieser Beschleunigungsstrukturen, die Anzahl der der Schnitteste auf eine logarithmische Komplexität reduziert werden. Diese Komponente ist eine der Komponenten im „Herzen des Raytracers“ (Kapitel 4.2 [1]) und ist essentiell entscheidend für die finale Laufzeit dieses. Dabei ist das Ziel schnell und gleichzeitig Gruppen von Primitiven auszuschließen. Beide Ansätze erstellen eine neue Szenenhierarchie für die Traversierung, die jeweils darauf ausgelegt ist, die „Kosten“ der gesamten Schnitttests zu minimieren.

In den folgenden beiden Unterabschnitten werden zwei der meist genutzten Beschleunigungsstrukturen kurz vorgestellt.
Die Bounding Volume Hierarchie und Kd-Trees.
Beide Strukturen und deren Implementierung sind im Detail in Kapitel 4.3 und 4.4 in dem Buch „Physical Based Rendering: From Theory to Implementation“ beschrieben [1].

2.3.1. Bounding Volume Hierarchies

Bounding Volume Hierarchies (BVHs) sind dem Bereich der Objekt Unterteilungsalgorithmen zuzuordnen. Dabei werden Objekte absteigend in kleinere disjunkte Teilmengen aufgeteilt.

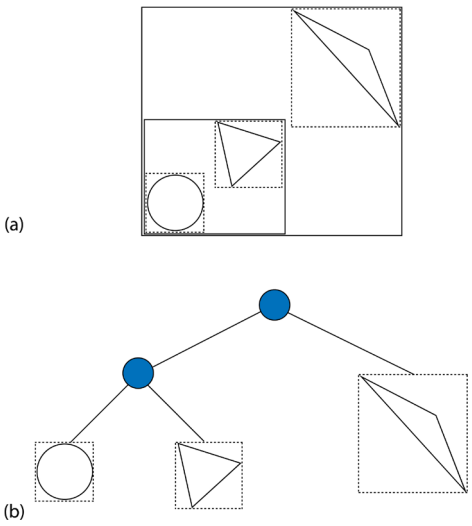

Abbildung 12 Bounding Volume Hierarchie Quelle: Physical Based Rendering [1]

Abbildung 12 (b) zeigt ein Beispiel einer Bounding Volume Hierarchie, von der Wurzel geht ein Blatt und ein weitere Knoten ab, von dem mittleren Knoten gehen

weitere zwei Blätter ab.
Dabei werden die Primitive in den Blättern gespeichert und jeder Knoten speichert eine Bounding Box[5] gestrichelte Boxen Abbildung 12 (a) seiner Kind-Knoten. Dadurch kann bei einer Traversierung ein Teilbaum übersprungen werden, wenn die Bounding Box dieses nicht geschnitten wird. Die Bounding Box des Wurzelknotens enthält demnach die komplette Szene.

Die genaue Konstruktion der Struktur wird diesem Rahmen nicht diskutiert. Zusammengefasst besteht der Prozess aus 3 Schritten. Zuerst wird die Hüllengeometrie für jedes Objekt der Szene bestimmt. Mithilfe dieser Information kann ein binärer Baum mithilfe verschiedener Teilungsalgorithmen konstruiert werden, der Knoten mit den Bounding Boxen und Blätter mit den Geometrien enthält, die Verbindung basiert auf Pointern. In einem folge Schritt wird diese Struktur in eine effizienter Struktur (depth-first array layout [1]) umgewandelt, die sich besser für das intern Speicherlayout des Computer eignet.
Eine detaillierte Beschreibung kann aber in Kapitel 4 Abschnitt 3 in dem Buch „Physical Based Rendering“ [1] gefunden werden.

Bei einer Traversierung wird beim Wurzelknoten begonnen und dann die Kinderknoten. Um die Traversierung effizienter machen, kann dieser Abhängig von der Richtung des Strahls den Kindknoten wählen, der je nach angewendetem Teilungsalgorithmus-Algorithmus eher getroffen wird, basierend auf den geometrischen Informationen.

Am Ende folgenden Abschnittes werden die Vor- und Nachteil dieser Struktur im Vergleich zu der Kd-Tree Struktur diskutiert.

2.3.2. Kd-Trees

Kd-Trees sind dem Bereich der räumlichen Unterteilungsalgorithmen (vgl. Binary Space Partitioning (BSP) Trees) zuzuordnen, die den 3D Raum in disjunkte Regionen unterteilen und dabei speichert, welchen Regionen Primitive schneiden. Bei der eigentlichen Traversierung müssen nur Primitive Testen, die die Regionen schneiden,

[5] Ein Hüllkörper (engl. Bounding Volume) ist ein einfacher geometrischer Körper der ein oder (in diesem Fall) mehrere komplexere Objekte umschließt. Bounding Boxen sind eine Teilmenge der Bounding Volume. Eine gute Übersicht über diesen Bereich bietet das Buch „Computational Geometry – Algorithms and Applications“ [13], insbesondere die konvexe Hülle in Kapitel 11 ist oft im Rahmen der Computer Grafik anzutreffen.

die gefunden wurden. Die Aufteilung des Raumes wird mithilfe von Schnitt-Ebenen durchgeführt.

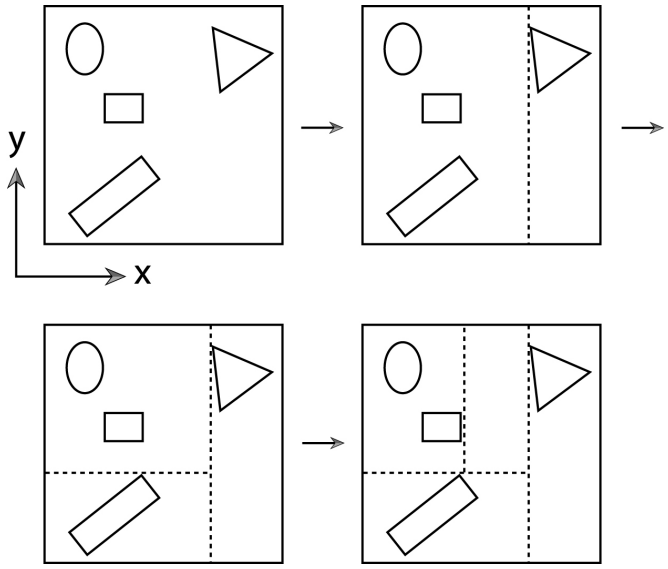

Abbildung 13 Kd-Tree Sturktur Quelle: Physical Based Rendering [1]

Auch dieser Algorithmus beginnt mit einer Bounding Box, die die gesamte Szene umschließt (Abbildung 13; oben links).
Es wird ein Schwellwert festgelegt, der angibt wie viele Primitive sich maximal in ein einem Unterschabschnitt befinden dürfen. Überschreitet die aktuelle Anzahl an Primitiven in einer Region diesen Schwellwert, wird die Region (3D Raum) mittels einer Ebene (2D) geschnitten und die Primitive werden auf die jeweiligen Unterregionen je nach Zugehörigkeit aufgeteilt (Abbildung 13; oben rechts & unten links). Es kann dabei vorkommen, dass die Teilungsebene so gewählt wird, dass Geometrien in beiden Teilregionen liegen; in diesem Fall werden diese mit beiden Teilregionen assoziiert und können demnach mehrfach in der Struktur vorkommen.
Dieser Prozess wird rekursiv fortgeführt bis entweder jedes Blatt die entsprechend definierte maximal Anzahl an Primitiven beinhaltet oder eine maximale Rekursionstiefe erreicht ist. Die Schnittebene ist immer rechtwinklig zu einer der drei Koordinaten Achsen[6].

Auch zu diesem Algorithmus kann eine detaillierte Beschreibung in dem Buch „Physical Based Rendering“ [1] in Kapitel 4 Abschnitt 4 gefunden werden.

[6] Eine weitere bekannte Variation des BSP Baums ist der Octree, hierbei wird die aktuelle Box durch 3 Ebenen die zu den Achsen rechtwinklig sind geteilt in 8 Unterregionen bei jedem Schritt.

Beim Traversieren, wird jede Region getestet und bei positiven Schnittest mi den Knoten, wird ermittelt welcher Kindknoten zuerst traversiert werden soll und dieser Prozess nach der Reihe durchgeführt. Auch dieser Algorithmus kann so konstruiert werden, dass beim Traversieren, die Kinderknoten zuerst getestet werden, die geometrisch wahrscheinlicher getroffen werden (depth-first, front-back Traversierung).

Eine wichtige positive Eigenschaft dieser der BVHs gegenüber der Kd-Trees ist, dass jedes Primitiv lediglich einmal in der Hierarchie vorkommt.
Ein Nachteil dieser Struktur ist, dass ein Primitiv mehrere räumliche Unterteilungen schneidet und daher unter Umstände mehrfach getestet werden müsste. Ein weiterer Nachteil bei den BVHs ist die benötigte Speichergröße der Struktur, insbesondere bei Szenen, wie der die in der Einleitung von Abschnitt 2.3 vorgestellt wurde kann dieses ein bestimmte Relevanz haben, die beachtet werden muss.
Die Kd-Tree Struktur liefert in der Regel schneller Schnitt-Test-Ergebnisse zurück ist aber komplizierter und weniger effizient zu konstruieren als die BVHs und benötigt daher eine längere Konstruktionsdauer. Des Weiteren sind Kd-Trees stärker anfällig bezüglich numerischer Robustheit auf Grund von Rundungsfehler.

2.4. Maschinelles Lernen

Dieser Abschnitt beschreibt was maschinelles Lernen ist und definierte bestimmte Begriffe die im Laufe der weiteren Arbeit wiederverwendet werden. Es ist zu beachten, das zum Großteil auf die Englischen Begriffe zurückgegriffen wurde, da diese sich in der Praxis durchgesetzt haben.

Das Prinzip des maschinellen Lernens basiert auf der Generierung von Wissen aus Erfahrung. Ein solches System beruht auf der Entwicklung von Lernalgorithmen aus Beispielen, wodurch ein komplexes Modell entwickelt werden kann.
Mithilfe dieses erlernten Wissens kann das System auf neue und potenziell unbekannte Daten angewendet werden. Bei Problemen die zu komplex sind um sie analytisch zu beschreiben, es jedoch viele Beispieldaten gibt bietet sich Maschinelles Lernen als Mittel an.
Einen guten Überblick über das Thema bietet das Buch „Deep Learning“ von Goodfellow et al. [7] sowie eine praxisorientierte Analyse kann in der 2018 vom Frauenhofer-Institut veröffentlich Studie „Maschinelles Lernen Eine Analyse zu Kompetenzen, Forschung und Anwendung“ [2] gefunden werden .

Labels & Features

Bei Features handelt es sich um die Input Parameter. Diese werden beim maschinellen Lernen als Eingangsparameter genutzt, um damit zu arbeiten. Als Label wird die Ausgabe (auch mehrere Ausgaben sind möglich) bezeichnet die geliefert wird, ausgehend von den (Input)-Features.

Dies kann mit einer Funktion $f(x_i,x_{ii}) = y$ verglichen werden, bei der x_i und x_{ii} die Features sind und y das Label. Features und Labels sind die Datenstruktur die verwendet wird um ein Netzwerk zu trainieren und nach abgeschlossenem Training stellen die Features die (unbekannten) Parameter dar, mit denen das Netzwerk arbeiten kann.

Generell gesagt steigt je nach Komplexität des Problems auch die Anzahl an Features.

Modell

Ein Modell beschreibt die Beziehung zwischen Features und Labels. Ein Model wird im Rahmen des maschinellen Lernens mit Features trainiert die ein Label besitzen und lernt dabei diese Beziehung. Anhand der erlernten Beziehungen kann das Modell Schlussfolgerungen bei unbekannten/neuen Features ziehen, die kein Label besitzen. Das Erlernen der Beziehung geschieht durch die Minimierung des Verlustes (s.u.) und wird als empirische Risikominimierung bezeichnet. [8]

Es gibt Verschiedene Arten von Modellen, wovon im Rahmen dieser Arbeit hauptsächlich zwei Typen von Modellen unterschieden werden:

Regression/Klassifikation

Ein lineares Regressionsmodell gibt kontinuierliche Werte zurück während ein Klassifikationsmodell diskrete Werte (z.B. Kategorien) zurück gibt. Ein Klassifikation Modell basiert in der Regel auf logistischer Regression, dabei wird eine Sigmoid Funktion ($y = \frac{1}{1+e^{-\sigma}}$) verwendet um die Prognose (σ) eines linearen Modelles in den Wertebereich [0,1] zu bringen, was als Wahrscheinlichkeitswert interpretiert werden kann.

Außerdem wird in dieser Arbeit zwischen Regressions-/Klassifikationsmodell und neuronalem Netzwerk unterschieden.

Bei einem linearen Modell werden die einzelnen Parameter mithilfe von angelernten Skalaren verbunden und die Beziehung definiert. Hierbei muss die

Zusammengehörigkeit von Werten vordefiniert sein (z.B. Feature Kreuzungen (Multiplikationen), Funktionen wie Sinus/Kosinus, Potenzfunktionen).
Dies ist in der Regel die schnellste Möglichkeit des maschinellen Lernens jedoch wird dies bei einer großen Anzahl an Features sehr schnell zu komplexen, nicht überschaubaren Zusammenhängen die schwer manuell modellierbar sind.
An dieser Stelle können Neuronale Netze ein Vorteil bieten. Neuronale Netze besitzen eine weitere Struktur zwischen Inputs und dem Output. Hierbei handelt es sich um sogenannten. Neuronen die sich in verschieden Layern befinden und die Inputs verbinden. Jedes Neuron besitzt zusätzlich zu den verschiedenen Gewichten der Verbindungen Aktivierungsfunktionen die ebenfalls über das Training veränderte Werte bekommen. Diese Aktivierungsfunktionen führen zu dem Verlust der allgemeinen Linearität des Modells.

Trainings-Set/Validation-Set/Test-Set

Um sicherzustellen, dass ein Modell generalisierbar bleibt werden die Testdaten in drei Teilmengen unterteilt. Dabei wird das Training-Set direkt zum Trainieren des Modells genutzt. Das Validation-Set wird genutzt um ein Trainiertes Modell zu evaluieren und damit dem Overfitting (Spezialisierung des Models bezüglich der Testdaten) entgegenzuwirken. Das Test-Set wird nach abgeschlossenem Training zum manuellen evaluieren des Models in einem zweiten Schritt genutzt. Der Hauptunterschied zwischen dem Validation- und Test-Set ist, dass das Validation Set bereits während der einzelnen Trainingsschritte genutzt wird. Hierbei beinhaltet die Schnittmenge aus Training- und Validation-Set generell 80-90% der Testdaten während das Test-Set den kleineren Anteil, also die verbleibenden 10-20% darstellt.

Loss

Als Verlus (engl. Loss) wird der Verlust durch schlechte Prognosen des Modells bezeichnet. Dieser kann beim Training mithilfe der etikettierten Trainings- und Validations-Daten bestimmt werden. Das Ziel des Trainings ist es, einen geringen Verlust über den Durchschnitt aller Werte zu erhalten.

Squared Loss (L2 loss) / Mean Square Error

Beim Squared Loss handelt es sich um eine Verlust Funktion, die die Differenz zwischen dem Label und der Prognose quadriert.
Die mittlere quadratische Abweichung (engl. Mean Square Error; MSE) bezeichnet den durchschnittlichen quadratischen Verlust pro Stichprobe über den gesamten

Datensatz. Die mittlere quadratische Abweichung ist folgendermaßen definiert (Kapitel 8, [9]):

$$MSE = \frac{1}{N} \sum_{(x,y)\in D} (y - prognose(x))^2$$

Formel 2.4 Mean Square Error

D beschreibt den Datensatz mit N Stichproben $\{(x_0,y_0),\ldots,(x_N,y_N)\}$ wobei x die Menge an Features und y das Label darstellt.
Squared Loss wird als Loss Funktion für lineare Regressionsmodelle genutzt.

Log Loss

Der Logarithmische Verlust (engl. abk. Log Loss) wird als Verlust Funktion für logarithmische Regression wie bei Klassifiziermodellen verwendet.
Die Log Loss Funktion ist folgendermaßen definiert:

$$LogLoss = \sum_{(x,y)\epsilon D} -y * log(prognose(x)) - (1-y) * \log(1 - prognose(x))$$

Formel 2.5 Log Loss

D beschreibt den Datensatz mit N Stichproben $\{(x_0,y_0),\ldots,(x_N,y_N)\}$ wobei x die Menge an Features und y das Label darstellt. Hierbei ist zu beachten, dass das Label y entweder 1 oder 0 ist. Die Prognose (*prognose(x)*) entspricht einem Wert zwischen 0 und 1.

Konvergenzkurven

Die Konvergenzkurve beschreibt die Veränderung des Velustes des Modells über die Perioden des Trainings. In der Regel wird diese für das Training-Set sowie das Validation-Set ermittelt.

Optimizer

Die Aufgabe des Optimierungsalgorithmus ist es den Verlust des Modells zu verringern und dadurch bessere Prognosen zu garantieren. Dieser wird beim Training des Modells angewendet. Optimierungsalgorithmen sind in der Regel spezifische Implementation des Gradient Descent Optimizers.

Gradient Descent Optimizer

Der Gradient Descent Optimizer ist ein Optimierer für generell konvexe Problems.

Als Gradient wird hier ein Vektor von partiellen Ableitungen der (unabhängigen) Gewichte des Modells bezeichnet. Mithilfe dieses Gradienten kann die Richtung der größten bzw. kleinsten Steigung der Funktion ermittelt werden. Der Gradient wird als ∇f notiert ist für eine gegebene Funktion *f(x, y)* der Vektor der partiellen Ableitungen:

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}(x,y), \frac{\partial f}{\partial y}(x,y)\right)$$

Formel 2.6 Gradient, partielle Ableitungen der Gewichte

Der Gradient Descent Optimizer nutzt die negative Richtung des Vektors (Richtung der geringsten Steigung) sowie dessen Betrag multipliziert mit der Learning Rate (s.u.) um schnellstmöglich den Verlust zu minimieren.

Learning Rate

Bei der Learning Rate handelt es sich um die Schrittgröße mit der ein Optimierer die Verlust-Funktion minimiert. Beim Gradient Descent Optimizer wird z.B. die Learning Rate auf den Betrag des Gradienten multipliziert. Das skalare Ergebnis dient als Faktor für die Optimierung in Richtung des Gradienten.
Generell gilt, dass eine sehr kleine Learning Rate eine längere Laufzeit beim Training veranlasst, während eine zu hohe Learning Rate dazu führen kann, dass das Optimum nicht erreicht wird. Jedes Problem hat eine eigene optimale Learning Rate (für eindimensionale Probleme ist dies die Inverse 2. Ableitung von f(x); $\frac{1}{f(x)''}$).

Batch/Batch Size

Ein Batch beschreibt die Menge an Stichproben die in einer Iteration des Modelltrainings genutzt werden soll. Bei der Batch Size handelt es sich um die Anzahl der Stichproben in der oben beschriebenen Menge. Generell gilt, dass es effizienter ist, den Verlust mithilfe eines kleinen Batches zu berechnen. Ein Batch besteht in der Regel aus 10 bis 1000 Stichproben.

Regularization

Generell ist ein größeres/komplexeres Modell stärker auf die Trainingsdaten spezialisiert, wodurch es schnell zu Overfitting kommen kann. Um dieser Spezialisierung entgegenzuwirken, wird die Regularisierung (engl. Regularization) genutzt um je nach Komplexität des Modells das Modelltraining anzupassen ähnlich

wie es die Verlust-Funktion tut. Dafür gibt es verschiedene Arten der Regularisierung:

L1 Regularization bestimmt die Komplexität des Modells anhand der Summe der absoluten Gewichte. Dadurch werden Features, die irrelevant oder kaum relevant sind aus dem Modell entfernt.

L2 Regularization bestimmt die Komplexität des Modells anhand der Summe der quadrierten Gewichte. Insbesondere bewirkt es bei Außreisern, dass diese nicht so stark auf das Modell wirken.

Dropout Regularization bewirkt, dass z.B. eine bestimmte zufällige Auswahl an Layern eines neuronalen Netzwerkes in einem einzelnen Gradientenschritt entfernt werden.

Early Stopping ist eine Methode, bei der das Training beendet wird, bevor der Verlust beim Validation-Datensatzes wieder zu steigen beginnt.

Mithilfe der Regularisierungsrate (λ) kann die Stärke der Regularisierungsfunktion auf das Modelltraining beeinflusst werden, sodass sich die Verlust Gleichung folgendermaßen ändert. Dies wird als „Strukturierte Risikominimierung“ bezeichnet.

$$minimize(loss\ function + \lambda(regularization\ function))$$

Formel 2.7 Strukturierte Risiko Minimierung

Threshold/Confusion Matrix

Klassifikationsmodelle geben als Prognose einen Prozentwert zurück. Um diesen zu einer Klassifikation zwischen zwei Kategorien zu konvertieren, wird ein Schwellwert (engl. Threshold Value) genutzt, mithilfe dessen die Prognose einer der beiden Kategorien zugeordnet werden kann. Durch die Zuordnung mithilfe des Schwellwertes kann es zu folgenden Zuständen kommen, die in der Folgenden Confusion Matrix gezeigt werden.

Tabelle 1 Confusion Matrix

True Positive (TP): Das Modell prognostiziert korrekt ein positives Ergebnis.	**False Positive (FP):** Das Modell prognostiziert ein positives Ergebnis, welches jedoch nicht ein negatives Ergebnis ist.
False Negative (FN): Das Modell prognostiziert ein negatives Ergebnis, welches jedoch ein positives Ergebnis ist.	**True Negative (TN):** Das Modell prognostiziert korrekt ein negatives Ergebnis.

Evaluation

Zur Evaluation des trainierten Modells können verschiedene Metriken verwendet werden. Im Folgendem werden die in dieser Arbeit verwendeten Metriken vorgestellt und kurz erläutert.

Accuracy/Precision/Recall/ROC Curve/AUC

Die Genauigkeit (engl. Accuracy) ist definiert als der Prozentsatz, den das Modell richtig prognostiziert.

$$Accuracy = \frac{Anzahl\ der\ korrekten\ Prognosen}{Gesamtanzahl\ der\ Prognosen} = \frac{TP + TN}{TP + FP + TN + FN}$$

Formel 2.8 Accuracy

Bei der Nutzung der Genauigkeit ist zu beachten, dass diese Metrik nicht stabil gegenüber nicht ausbalancierten Datensätzen ist, beispielsweise wenn Klasse 1 zu 2% vertreten ist, während Klasse 2 zu 98% vertreten ist.

Die Präzision (engl. Precision) beschreibt den Anteil der positiv prognostizierten Stichproben die korrekt sind (Precision = TP / (TP+FP)). O

Die Treffer-Quote (engl. Recall) beschreibt den Anteil der positiven Stichproben die korrekt prognostiziert wurden. (Recall = TP / (TP+FN)).

Eine ROC-Kurve (Grenzwertoptimierungskurve; engl. Receiver Operating Characteristic Curve) kann genutzt werden um die Performanz eines Klassifikationsmodells bei allen Klassifikationsschwellwerten zu plotten. Die Ordinate entspricht der Sensitivität (korrekt positiv Klassifizierungen/Recall) und die Abszisse

der 1-Spezifität (korrekt negativ Klassifizierung; engl. False Positive Rate; FPR = FP / (FP + TN)) des Modelles. Die Fläche unter der geplotteten Kurve wird als AUC (engl. Area under the Roc Curve) bezeichnet. Dieser Wert kann als weitere Metrik genutzt werden um die Performanz des Modells zu bestimmen. Bei einem Modell, dessen Prognosen immer richtig sind, ist die der AUC-Wert 1 (bzw. 0 bei einem Inversen Ergebnis). Bei einem Modell, dessen Prognosen immer richtig sind, ist der AUC-Wert 1, in allen anderen Fällen liegt der Wert zwischen diesen beiden Werten [0,1], wobei 0.5 der schlechteste Fall ist (zufälliges Klassifizierungsmodell).

2.5. Tensorflow

Im Rahmen der Forschungsarbeit dieser Thesis wird das Tensorflow Framework genutzt.
Hierbei handelt es sich um eine Bibliothek zur datenstromorientierten Programmierung, welche einer der meist verwendeten Bibliotheken ist, die im Rahmen des Maschinellen Lernens verwendet wird. Die Bibliothek bietet gleichzeitig Low-Level und High-Level Funktionen zum Definieren, Trainieren und Ausführen von Modellen, außerdem werden Möglichkeiten des Exportes und Importes geboten.
In der folgenden Grafik von der Tensorflow Website wird die Hierarchie des Tensorflow Toolkits gezeigt.

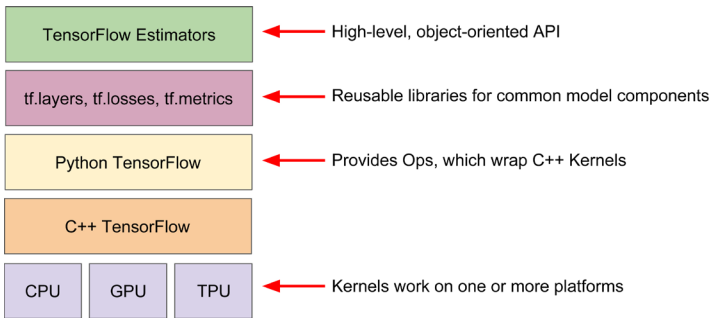

Abbildung 14 Tensorflow Framework Hierarchie

Im Rahmen dieser Arbeit wird hauptsächlich auf der Ebene der TensorFlow Estimators gearbeitet und es werden die vordefinierten Modelle (linear regressor, linear classifier & neural networks) genutzt.
Der folgende Code Ausschnitt zeigt wie Tensorflow in Python verwendet wird in dem

ein Linearer Regressor erstellt und Trainiert sowie ausgewertet wird.

```
1.  import tensorflow as tf
2.
3.  # erstellen eines linearen Regressors
4.  estimator = tf.estimator.LinearRegressor(feature_columns)
5.
6.  # trainieren des Models mit Beispieldaten
7.  estimator.train(input_fn=train_input_fn, steps=2000)
8.
9.  # mithilfe des trainierten Models Prognosen erstellen
10. Predictions = estimator.predict(input_fn=predict_input_fn)
```

Listing 2.1 Tensorflow Erstellen, Trainieren und nutzen eines Linearen Regressors

In sämtlichen Testläufen wurde die Tensorflow Version 1.14.0 in der GPU variante genutzt.

3.

Hardware

Sämtliche Training und Testläufe wurden auf einem Computer durchgeführt, welcher folgende Spezifikationen aufweist.

Tabelle 2 Verwendete Computer Hardware

CPU	AMD Ryzen Threadripper 2950X 16-Core Processor 3,50Ghz
GPU	NVIDIA GeForce RTX 2080 Ti Dedizierter GPU-Speicher: 11GB
Arbeitsspeicher	32 GB
Sekundär Speicher	Samsung SSD 970 Pro 512GB Samsung SSD 860 EVO 1TB
Betriebssystem	Windows 10 – Pro 64 Bit

Wie im 2. Abschnitt beschrieben wurde die GPU Variante von dem Tensorflow Framework verwendet. Während sämtlichen konnte folgende Beobachtungen gemacht werden.

Bei jedem Training wurden lediglich 12% der GPU Kapazität genutzt, die meiste Zeit befand sich die GPU im Copy-Modul, bei genaueren Analyse kann erkannt werden, dass der dedizierte GPU-Speicher eine maximale Auslastung hat und sich hier scheinbar das Bootleneck befindet. Die folgende Abbildung zeigt die einzelnen Auslastungen der Grafikkarten Komponenten, hier ist klar die starke Auslastung des dedizierten GPU Speichers zu erkennen. Im direkten Vergleich konnte jedoch festgestellt werden, dass das Training auf der GPU in der og. Systemkonfiguration trotz dieses Bottlenecks schneller ist als auf der CPU.

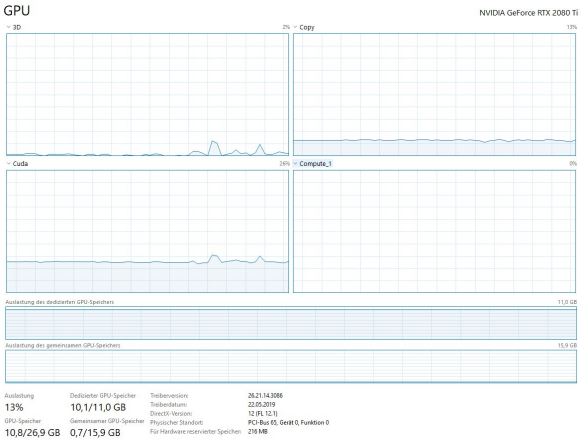

Abbildung 15 GPU Auslastung beim Modell-Training

Als Umgebung wurde Python gewählt, dies stellte sich jedoch an einigen Stellen als problematisch dar, da Python Script zur Laufzeit ausgeführt werden und nicht kompiliert werden mit den entsprechenden Compiler-Checks, sodass an einigen Stellen semantische Fehler oder fehlende Bibliotheks-Importe dazu führten, dass ein Training nach sehr langer Berechnungszeit nicht komplett evaluiert werden konnte. Insbesondere aus diesem Grund wurde direkt nach dem Schritt des Modelltrainings dieses Exportiert, sodass dieses in einem weiteren Schritt manuell Evaluiert werden kann und das trainiert Modell nicht verloren geht.

4.

Dataset

Dieser Abschnitt beschreibt die verwendete Szene für die Testdaten und die darin enthaltene Geometrie sowie die relevanten Szene Parameter.
Im weiteren Verlauf wird beschrieben wie die Testdaten in Features & Label konvertiert werden und der Datensatz in Trainings-, Validation- und Testdatensatz aufgeteilt wird. Außerdem wird die Verteilung innerhalb dieser Teilmengen analysiert.

4.1. Testszene & Geometrie

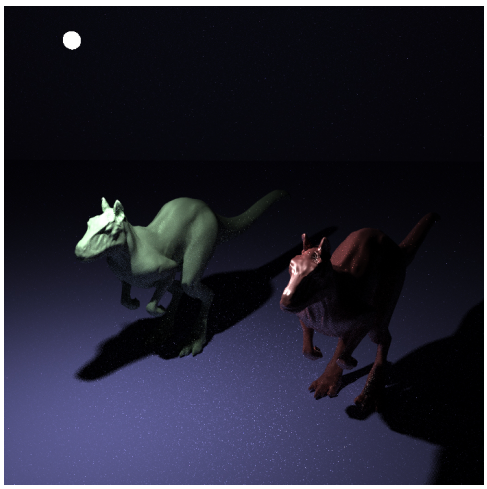

Abbildung 16 Killeroo-Simple Scene

Bei der verwendeten Testszene, aus der der Datensatz für diese Arbeit exportiert wurde handelt es sich um die “Killeroo”-Simple Scene des PBRT-Renderers. Es handelt sich um eine einfache Szene mit 2 Mesh-Modellen, einer Grundfläche Sowie eine Hintergrund Fläche und einer Kugel. Außerdem wurden die Render Parameter so angepasst, dass die Anzahl der zu exportierenden Strahlen nicht zu groß wird, dies kann insbesondere in Abbildung 16 an den Artefakten auf dem Boden erkennen. Mit dem in Abschnitt 2.2 beschriebenen PBRT-System kann diese Szene auf

obengenannter Hardware innerhalb von unter einer Minuten gerendert werden, unter Ausnutzung sämtlicher Kerne.

Tabelle 3 Übersicht Szenenparameter Testszene

Kamera	Look-At-Matrix	$\begin{matrix} 400 & 20 & 30 \\ 0 & 63 & -110 \\ 0 & 0 & 1 \end{matrix}$	
	Rotation	-5° um die Z-Achse	
	Field-Of-View	39	
Objekte		Position	Skalierung
	Killerroo Tier #1 (Mesh)	100 200 -140	[<TODO>]
	Killerroo Tier #2 (Mesh)	-100 200 -140	[<TODO>]
	Grundläche (Triangle Mesh)	0 0 -140	2000x2000 horizontal
	Hintergrundfläche (Triangle Mesh)	-400 0 -140	2000x2000 vertikal
	Sphere	150 120 20	Radius 3

Tabelle 3 zeigt die Szenenparameter dieser Szene.
Eine Kamera ist an der Position (400 20 30) und hauptsachlich in negative X-Richtung ausgerichtet. Der normalisierte (gerundete) Richtungsvektor der Kamera kann aus der Look-At-Matrix abgeleitet werden und lautet (-0.94, 0.1, -0.33), es ist dabei zu beachten, dass das PBRT System ein linkshändiges Koordinatensystem nutzt wie in Kapitel 2.1 beschrieben. Die Kamera ist um -5 Grad um die Z-Achse gedreht. Es ist zu beachten, dass in dieser Szene die Z-Achse den Up-Vektor darstellt. Bei der Kamera handelt es sich um eine perspektivische Kamera mit einem Sichtfeld von 39°. Das gerenderte Bild der Szene (Abbildung 16) hat eine quadratische Auflösung von 700x700 Pixeln, eine Dateigröße von 1,92 MB und besitzt das von Industrial Light & Magic entwickelte high dynamic-range Bildformat.
<TODO> Es wird als Sampler der Halton und ein Pathracing Algorithmus

angewendet. Es wird ein Pixelsample von 128 verwendet.
Die Kugel befindet sich an der Position (150 120 20) und besitzt den Radius 3, sie stellt eine Flächenlichtquelle dar, die Licht emittiert.
Die Grundfläche liegt in der XY-Ebene und auf beiden Achsen jeweils im Wertebereich von [-1000,1000].
Die beiden Mesh-Objekte sind gegenüber den Definitionen um den Faktor 0.5 skaliert und jeweils um -60° um die Z-Achse rotiert. Somit haben das Mesh-Modell nach dieser skalierung eine Ausmaße von [<TODO>].
Das erste Modell befindet sich an der Position (100, 200, -140).
Und das 2. Modell an der Position (-100, 200, -140).
Da die Material- und Lichteigenschaften keine Relevanz für die Schnitttests haben, wird auf eine genauere Analyse dieser Parameter im Rahmen dieser Arbeit verzichtet.

Aus den Statistiken die beim Rendern erstellt wurde, kann erkannt werden, dass für die Szene 16,87 Mio. Schnitttests durchführt bei denen es sich bei 6,16 Mio. Schnittests um Schattenstrahlen handelt. Von der Kamera selber sind von den Schnittests 3.9 Mio. Strahlen ausgegangen.

4.2. Feature Engineering

In diesem Abschnitt wird beschrieben wie die exportieren Raw Daten aus dem PBRT Raytracer aussehen und wie diese in Feature Vektoren konvertiert werden.

Raw Data

```
0 : {
        origin: [(float) origin.x, (float) origin.y, (float) origin.z]
        direction: [(float) direction.x, (float) direction.y, (float) direction.z]
        hit: (boolean) hit
}
```

In der oberen Beispielstichprobe wird dargestellt, welche Parameter beim Rendern der Szene exportiert wurden. Es handelt sich hier um den Strahlentest bzw. deren Ergebnis. Wie in Abschnitt 2.1. beschrieben ist, basiert das Raytracing auf der Strahlenverfolgung und ein Strahl kann durch den Ursprung (origin) und die Richtung (direction) definiert werden. Bei diesen beiden Eigenschaften handelt es sich jeweils um Vektoren mit 3 Komponenten. Des Weiteren befindet sich in den Raw Daten noch das Feld hit, welches angibt ob der Strahl eine der Szenen-Geometrien schneidet

oder nicht.
Wenn der Wert des Feldes „hit“ 0 ist, wird die Szenen-Geometrie nicht geschnitten, wenn der Wert 1 ist, trifft der Strahl die Szenen Geometrie.
Wie man an der Beispielspiel-Stichprobe aus den Raw Daten erkennen kann sind sämtliche Parameter, die als Features verwendet werden sollen bereits gleitkomma-Werte sodass für diese kein Mapping genutzt werden muss.

Bei der Feature Konstruktion und Optimierung (engl. Feature Engineering) werden beim maschinellen Lernen oft Gleitkommawerte in bestimmten Wertebereichen zusammengefasst und somit diskretisiert, dies kann den Lernaufwand des Modells minimieren und den Einfluss von Ausreisern minimieren. Dies wird auch als Binning bezeichnet. (Kapitel 8 Representation; Cleaning Data [8])
Z.B. könnte man Geografische Koordinaten zu Gebieten anhand des Langen und Breitengrades zusammen fassen.
Im Rahmen des Raytracing ist jedoch die geometrische Genauigkeit insbesondere bei Schnittest von ausschlaggebender Bedeutung. Eine kleine Abweichung bezüglich der Floating Point Wert kann entscheidende Auswirkungen haben. Insbesondere aus diesem Grund nutzen viele Ray-tracer double Präzision anstelle von floating point Präzision[7] und nehmen dadurch einen höheren Speicherverbrauch und eine längere Renderdauer in Kauf.
Auch weitere Methoden, wie das Skalieren der Felder des Feature Vektors oder das Clamping (Abschneiden) wurden aus obengenannten Bedenken vermieden. (Kapitel 8 Representation; Cleaning Data [8])
Beim Skalieren von Features wird in der Regel der Bereich kleiner gemacht, dies würde zu einer geringeren Präzision führen. Im Gegensatz würde bei m Skalieren mit einem Faktor > 1 zwar keine Präzision verloren gehen, solange die Werte im Werte Bereich des float Wertebereichs bleiben, jedoch können die Modelle des maschinellen Lernens besser mit kleineren Wertebereichen (>0) arbeiten. (vgl. [8]) Dadurch würde diese Veränderung der Daten voraussichtlich auch kontraproduktiv wirken. Beim Clamping geht ebenfalls, wie der Begriff bereits erahnen lässt, die Präzision der Daten verloren.
Insbesondere werden die vorgestellten Methoden des Feature Engineerings genutzt um Daten die eine größere Streuung besitzen oder Abweichungen von den

[7] Nach dem IEEE Standard [14] umfasst ein float Wert 32 Bit und ein double 64 Bit, somit die doppelte Größe und daher eine höhere Präzision.

generellen Werten zu behandelnd. Dadurch sollen insbesondere Ausreißer oder fehlerhafte Daten einen kleineren (bis keinen) Einfluss auf das Training des Modells besitzen. Dadurch, dass die hier genutzten Daten auf geometrisch und numerisch exakten Berechnungen basieren, wird nicht von fehlerhaften Daten und großen Ausreisern ausgegangen (bis auf die beschriebenen kleine Sampling Artefakte). Im folgenden Abschnitt wird auf die Verteilung der Werte in dem exportieren Datensatz eingegangen.

Die beschriebenen Methoden könnten jedoch bei einem zufälligen Sampling von Strahlen im Raum die Verteilung dieser verbessern.

Für das Training mit den ermittelten Daten wurden folgende drei Feature Vektoren entworfen und erstellt.

Feature Vektor #1
[origin.x, origin.y, origin.z, direction.x, direction.y, direction.z]

Der Feature Vektor besteht aus genau den identischen Daten, die aus den exportieren Raw-Daten entnommen wurden. Er besitzt eine Länge von 6 Floating-Werten, die ersten drei Werte sind der Ursprung des Strahls die folgenden drei Parameter beziehen sich auf die Richtung des Strahls.

Feature Vektor #2 (mit synthetischen Features)
[origin.x, origin.y, origin.z, direction.x, direction.y, direction.z, angle.x, angle.y, angle.z]

Eine zweite Variante beinhaltet, zusätzlich zu den Felder des Feature Vektors #1, 3 weitere Felder. Bei diesen Feldern handelt es sich um synthetische generierte Richtungskosinus [10] des Richtungsvektors. In Abbildung 17 stellt der Vektor $\vec{v}$ den ermittelten (euklidischen) Richtungsvektor in $\mathbb{R}^3$ dar und $\overrightarrow{e_x}$, $\overrightarrow{e_y}$, $\overrightarrow{e_z}$ die Standardbasisvektoren.

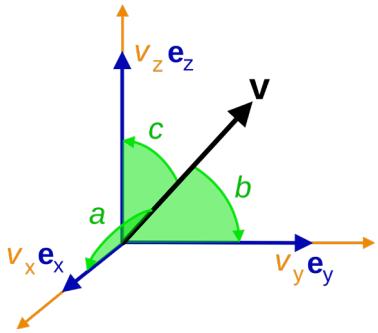

Abbildung 17 Vektor v im $\mathbb{R}^3$
Quelle: Direction cosine, en.Wikipedia.org

$$\alpha = cosa\ ;\ a = \frac{\vec{v} \cdot \overrightarrow{e_x}}{|\vec{v}||\overrightarrow{e_x}|} = \frac{v_x}{|\vec{v}|} = \frac{v_x}{\sqrt{{v_x}^2 + {v_y}^2 + {v_z}^2}}$$

Formel 4.1 Berechnung des Richtungskosinus zwischen dem Richtungsvektor ($\vec{v}$) und Standardbasisvektor $\overrightarrow{e_x}$

Formel 4.1 zeigt die Berechnung des Richtungscosinus zwischen einem Vektor $\vec{v}$ dem Standardbasisvektor $\overrightarrow{e_x}$, der die Welt X-Achse der Szene beschreibt $\overrightarrow{e_x} = (1,0,0)$, diese basiert auf dem Kosinus Satz. Die Berechnungen der Richtungscosinus β und γ ist äquivalent mit den entsprechenden Basisvektoren und wird hier nicht explizit notiert.

Da es sich bei den Richtungsvektoren um Einheitsvektoren handelt ist der Divisor bei der Formel 4.1 immer 1, sodass eine rechenintensive Berechnung der Quadratwurzel entfällt. Dadurch muss lediglich auf die einzelnen Komponenten des Richtungsvektoren die Arkuskosinus Funktion (Umkehrfunktion zum Kosinus) angewendet werden, um den Winkel zu erhalten.

Die Erstellung dieser synthetischen Features kann vor dem Modelltraining mithilfe der gegebenen Richtungsvektoren und der obengenannten Formel für alle drei Standardbasisvektoren durchgeführt werden für jeden Strahl in dem Datensatz.

Die berechneten Richtungswinkel sind in Rad nach der Anwendung der Arkuskosinus Funktion, eine Konvertierung in Grad wurde für die Modelltrainingsversuche als nicht notwendig erachtet.

Es ist zu bedenken, dass die Umformung auch bei späterem Testen und der Anwendung des Modells durchgeführt werden müsste.

Feature Vektor #3 (mit Feature Kreuzungen)

[origin.x, origin.y, origin.z,
crossOrigin [= cross(origin.x, origin.y, origin.z)],
direction.x, direction.y, direction.z,

crossDirection [= cross(direction.x, direction.y, direction.z],
crossOrigDir [= cross(crossOrigin, crossDirection)]]

Ein alternativer Ansatz ist es die semantisch zusammengehörenden Daten mittels Feature Kreuzung (engl. Feature Crossing) zu kombinieren. Insbesondere findet Feature Kreuzung dann statt, wenn es sich um nicht lineare Probleme handelt, die mit dem Modell gelöst werden sollen. Dabei werden zwei oder mehr Features miteinander Multipliziert und es entsteht dadurch ein neues Feature welches beim Training wie die anderen Features ein Gewicht besitzt welches erlernt wird.
Um den Zusammenhang der 3 Werte des Ursprungs im $\mathbb{R}^3$ zu verdeutlichen, wird eine Feature Kreuzung (*crossOrigin*) zwischen den Werten origin.x, origin.y und origin.z erstellt. Dasselbe gilt für den Richtungsvektor, für den eine Feature Kreuzung (*crossDirection*) zwischen den Werten direction.x, direction.y und direction.z erstellt wird. Um den geometrischen Zusammenhang zwischen dem Ursprung und dem ausgehenden Richtungsvektor zu verdeutlichen wird eine weiterer Feature Kreuzung (crossOrigDir) zwischen den neu erstellen Feature Kreuzungen (crossOrigin & crossDirection) erstellt.
Wie auch bei dem Feature Vektor #2 muss auch bei dieser Feature Kreuzung beachtet werden, dass sämtliche für die Trainingsdaten erzeugten synthetischen Features in gleicherweise für die Testdaten und in der späteren Anwendung erzeugt werden.

Neben den hier vorgestellten Feature Vektoren gibt es noch viele andere Möglichkeiten die Daten zu verarbeiten und Varianten für die Feature Vektoren zu erstellen.
Letztendlich ist beim maschinellen Lernen (insb. wenn keine neuronalen Netze verwendet werden) die Kreierung von synthetischen Features einer der wichtigsten Schritte und notwendig um Modelle für komplexere Probleme (die nicht linear lösbar sind) zu erlernen. Das hier beschriebene Feature Engineering nimmt in der Regel am meisten Zeit in Anspruch beim Erstellen von Systemen des maschinellen Lernens incl. der Datengenerierung und Daten Auswertung, letztes wird in dem folgenden Abschnitt beschrieben.

Als Label für das Training und die Validierung wurde jeweils der “hit”-Wert aus den Raw-Daten verwendet. Der binäre Wert gibt an, ob der Strahl die Szenen Geometrie schneidet.
Bei den Klassifizierung Modellen wird der hit-Wert verwendet um daraus kategorische Label zu erstellen, wie in der Einleitung dieses Abschnitts beschrieben steht der Wert 0 dafür, dass der Strahl die Szenen Geometrie nicht trifft und wird also Kategorie 0 (negativ) gewertet. Die Kategorie 1 wird als positiv gewertet und beschreibt, dass der Strahl die Szenengeometrie trifft, in den Raw-Daten ist dieser mit dem hit-Wert 1 gekennzeichnet.

4.3. Datenteilung

Der komplette Datensatz aller Schnittest beim Rendern der Szene ist eine CSV-Datei mit insgesamt ~16.8 Mio. Schnitttest, die Dateigröße beträgt ~1.2 Gb.
Um die Speichergröße zu reduzieren, wurde ein kleinerer Datensatz der aus eine Teilmenge des kompletten Datensatz besteht erstellt. Der kleinere Datensatz umfasst ungefähr eine Million Daten und besitzt eine Größe von etwas mehr als 70 MB. Dieser Datensatz wurde für die in Abschnitt 5 beschriebenen Modell-Trainings genutzt um die Trainingszeit in einem überschaubaren Rahmen zu halten.
Die Datenplots in den folgenden Unterabschnitten zeigen die Verteilung der Features innerhalb der Datensätze und beschreiben diese.

4.3.1. Kompletter Datensatz

Der Komplette Datensatz enthält insgesamt 16.844.643 Daten in dem Datensatz. Die Daten wurden mit der in der Einleitung in Abschnitt 4.1 beschriebenen Szene erstellt.

Die Schnitttests die exportiert wurden, sind die die beim Rendern der Szene mit den beschrieben Parametern verwendet wurden. Die Folgende Abbildung zeigt die Verteilung der Ursprungs-Koordinaten der Strahlen.

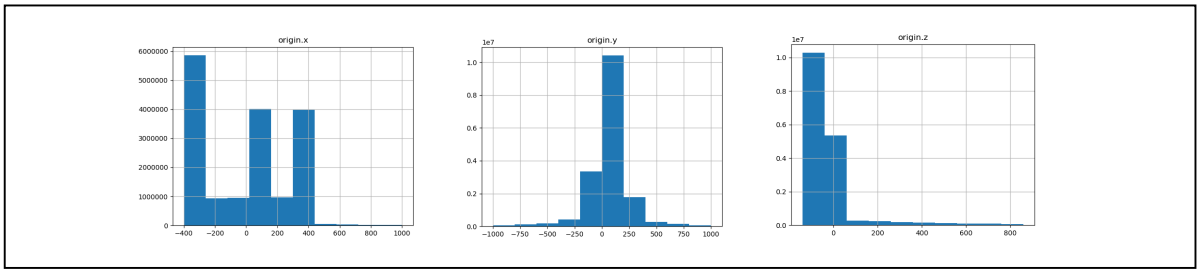

Abbildung 18 Plot Ursprungskoordinaten der Strahlen; Kompletter Datensatz

Es ist zu erkenne, dass ein großer Teil der x-Koordinaten der Ursprungspunkte ungefähr bei den Werten -400 und 400 liegen. Vergleicht man Tabelle 3 wird ersichtlich, dass diese Werte der Kameraposition (400) und der Hintergrundflache (-400) entsprechen. Die weiteren Szenen Geometrien sind durch den mittleren Peak zwischen 0 und 200 ist auf die Lichtquelle zurückzuführen, die sich an der Position x=150, y=120 & z= 20 befindet (vgl. Tabelle 3). Die weiteren kleineren Verteilungen repräsentieren, die weiteren Objekte in de Szene.
An dem Plot der Z-Koordinate in Abbildung 18 kann erkannt werden, dass sich

sämtliche Ursprungspunkte um den 0 Wert befinden, insbesondere sind keine extremen Ausreißer bei der Z-Richtung in die negative Richtung zu erkennen. An dieser Stelle ist nochmals zu beachten, dass es bei der Z-Achse in dieser Szene um den Up-Vektor handelt und damit die „Höhen“-Werte beschreibt. Somit beschrankt, die Grundfläche der Szene die Ursprungspunkte auf ein Minimum von -140 (vgl. Abbildung 18).
Die Verteilung auf der Y-Achse zeigt eine Art Normalverteilung um den Wertebereich 0-250. Der Wertebereich von -1000 bis 1000 lasst sich durch die Größe der Grundflache erklären.

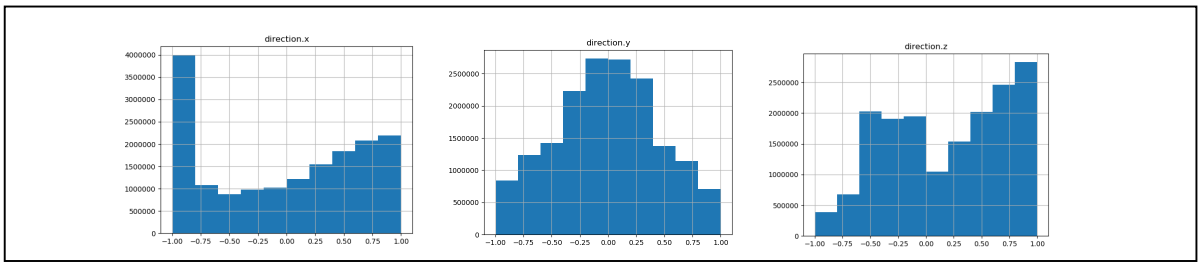

Abbildung 19 Plot Richtungsvektoren der Strahlen; Kompletter Datensatz

Abbildung 19 zeigt die Verteilung der Richtungsvektoren bzw. deren Komponenten.
Es ist zu sehen, dass ein großer Anteil an Richtungsvektoren in die negative x-Richtung zeigt, dies ist zurückzuführen auf die Orientierung der Kamera, die von dem Punkt 400 in Richtung der negativen X-Achse gerichtet ist.
Es ist zu erkennen, dass ein sehr geringer Anteil an Strahlen eine negative Richtung von -1 besitzen, die Strahlen haben tendenzielle eine Ausrichtung in positiver z-Achse, die Verteilung im Intervall [-0.5, 0.0] ist hauptsachlich auf die initiale Neigung der Kamera zurückzuführen.
Der Plot der Y-Richtungen weist wieder eine Normalverteilung auf, wie bei den Y-Ursprungs-Koordinaten und zeigt, die hauptsächlich frontale Verteilung der Strahlen.

Label

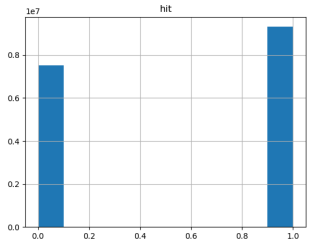

Abbildung 20 Plot Verteilung positive & negativer Schnitttests der Strahlen; Kompletter Datensatz

Der Plot in Abbildung 20 zeigt die Verteilung der Schnittest Ergebnisse, mit anderen Worten ob ein Strahl die Szenen Geometrie getroffen hat (1) oder nicht (0).
Es ist zu erkennen, dass die Verteilung nicht ganz ausgewogen ist, da eine Tendenz positiver Schnittesten besteht. (Dies spricht für den Verwendeten Raytracing Algorithmus, da negative Schnittests Zeitkosten und möglichst zu minimieren sind).
Von den ~16 Mio. Schnittesten besitzen ~7 Mio. ein negatives Label und ~9. Mio. ein positives.

Tabelle 4 Übersicht Parameter-Verteilung im kompletten Datensatz

	Origin.x	Origin.y	Origin.z	Direction.x	Direction.y	Direction.z	Hit
Count	16844643.0	16844643.0	16844643.0	16844643.0	16844643.0	16844643.0	16844643.0
Mean	-12.8	56.6	-42.3	0.0	0.0	0.2	0.6
Std	315.9	185.5	144.6	0.7	0.5	0.5	0.5
Min	-400.0	-999.9	-140.0	-1.0	-1.0	-1.0	0.0
25%	-400.0	1.2	-150.0	-0.8	-0.3	-0.3	0.0
50%	64.0	54.8	-91.6	0.1	0.0	0.3	1.0
75%	226.9	117.5	30.0	0.6	0.3	0.7	1.0
Max	1000.0	1000.0	860.0	1.0	1.0	1.0	1.0

An der Tabelle 4 sowie den vorherigen Verteilungsdiagrammen kann erkannt werden, dass sämtliche Features und Labels frei von unerwarteten Werten („Magic Values“) sind, die außerhalb der erwarteten Wertebereiche liegen, sodass dies nicht bei Feature Engineering und Training besonders beachtet werden muss. (Dies ist zurückzuführen, dass es sich bei den Daten um geometrische Berechnungen und

nicht zufällige Erhebungen handelt).

In Tabelle 4 sind die beschriebenen Verteilungen numerisch zusammengefasst und es wurden jeweils der Mittelwert (engl. Mean), die Standardabweichung (Std), der Minimum und Maximum-Wert sowie der 25-,50- und 75-Percentil bestimmt und in der Tabelle eingetragen.

4.3.2. Kleiner Datensatz

Von dem Kompletten Datensatz aus Abschnitt 4.3.1 wurde eine Teilmenge erstellt, die die ersten 1.048.575 Daten des Datensatzes enthält. Dies wurde durchgeführt, um die Menge an Trainings- und Testdaten geringer zu machen. Insbesondere ist die dadurch neu erstellte CSV-Datei lesbar für Programme wie Microsoft Excel[8] und es wird ermöglicht, einen schnelleren, detaillierten Einblick in den Datensatz zu bekommen. Es wurden für diese Teilmenge die oberen 1.048.575 Daten verwendet. Da sich dies auf die Verteilung auswirken könnte, sind im Folgenden die Verteilungen dieser geplottet.

Features

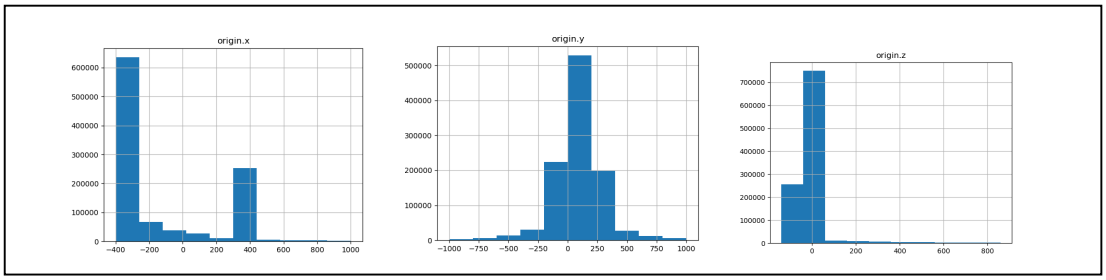

Abbildung 21 Plot Ursprungskoordinaten der Strahlen; Kleiner Datensatz

[8] In Microsoft Excel ist ein Tabellenblatt auf 1.048.576 Zeilen und 16.384 Spalten beschränkt. Da die erste Zeile in der CSV-Datei die Spalten beschreibt, bleiben 1.048.576 Zeilen für die reinen Daten. https://support.office.com/en-ie/article/excel-specifications-and-limits-1672b34d-7043-467e-8e27-269d656771c3

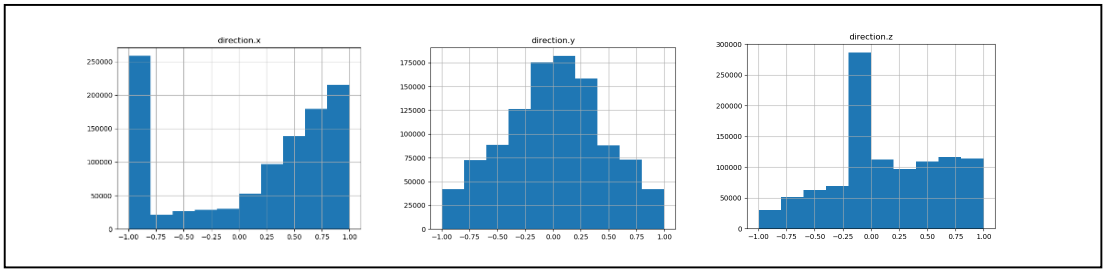

Abbildung 22 Plot Richtungsvektoren der Strahlen; Kleiner Datensatz

Abbildung 21 und Abbildung 22 zeigen die Verteilungen der Ursprungskoordinaten und der Richtungsvektoren in dem kleinen Datensatz.
Es fällt auf, dass sich der Plot der Ursprungs-X-Koordinate stark von dem Plot dieser im gesamten Datensatz (vgl. Abbildung 18) unterscheidet. Insbesondere ist festzustellen, dass viele der Daten in dem Bereich zwischen -200 und 200 nicht vertreten sind. Da sich in diesem Bereich um die Szenen Geometrie wie die beiden Mesh-Objekte (die Tiere) befinden, beschreibt dieser Teil der Daten die indirekte Reflexion und direkten Licht Einflüsse, die auf die Objekte wirken. Insbesondere die Position der Lichtquelle ist sehr unterrepräsentiert.
Wenn ein System erstellt werden soll, dass eine Beschleunigung für die Berechnung der Sichtbarkeit zwischen Lichtquellen und Objekten (z.B. Schattenstrahlen) genutzt werden soll, sind genau diese Daten relevant.
Daraus ist zu erkennen, dass insbesondere die Strahlen, der unteren Rekursionslevel repräsentiert sind.
Dieser Umstand verändert jedoch nicht, dass diese Forschung auch für Schattenstrahlen gesehen werden kann, wenn die Kamera bzw. die Wand im Hintergrund als Lichtquelle gesehen werden, dann sind die ausgehenden Strahlen diese gut repräsentiert.
Ausgehenden von der Beobachtung, dass hauptsächlich die Strahlen ausgehend von der Kamera und der Hintergrundfläche vertreten sind, kann auch an der Verteilung der Komponenten der Richtungsvektoren erkannt werden, dass die von diesen Objekten ausgehen, sodass eine größere Ansammlung von Richtungsvektoren mit der X-Richtung +1 und -1 in Abbildung 22 zu erkennen ist.
Bei dem Richtungsvektor Plot ist ebenfalls festzustellen, dass bei der Z-Koordinate der Bereich zwischen -0.25 und 0 sehr stark vertreten ist im Vergleich zu den restlichen Werten. Diese starke Ansammlung in dem Bereich ist im kompletten Datensatz nicht so stark zu erkennen.
Es ist zu vermuten, dass durch die Rotation der Kamera sowie die Look-At Matrix und

der sehr stark präsenten Kamerastrahlen in diesem Datensatz, diese Verteilung resultiert.

Label

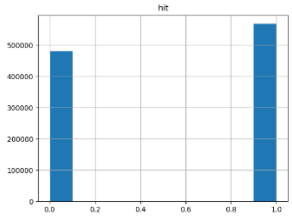

Abbildung 23 Plot Verteilung positive & negativer Schnitttests der Strahlen; Kleiner Datensatz

Die Verteilung der positiven und negativen Schnitttestergebnisse, ähnelt in der Verteilung der des vollständigen Datensatzes (Abbildung 20). Der Datensatz enthält konkret ca. 470.000 negative Schnittest Ergebnisse und ca. 530.000 positive Schnitttest Ergebnisse.

Tabelle 5 Übersicht Parameterverteilung im kleinen Datensatz

	Origin.x	Origin.y	Origin.z	Direction.x	Direction.y	Direction.z	Hit
Count	1048575.0	1048575.0	1048575.0	1048575.0	1048575.0	1048575.0	1048575.0
Mean	-141.7	76.5	-4.5	0.1	0.0	0.1	0.5
Std	348.1	207.0	105.6	0.7	0.5	0.5	0.5
Min	-400.0	-999.9	-140.0	-1.0	-1.0	-1.0	0.0
25%	-400.0	-10.5	-11.0	-0.8	-0.1	-0.1	0.0
50%	-400.0	54.8	18.6	0.4	0.0	0.0	1.0
75%	396.7	186.3	30.0	0.8	0.5	0.5	1.0
Max	999.9	999.9	860.0	1.0	1.0	1.0	1.0

Zusammenfassend für diesen Abschnitt zeigt Tabelle 5 die Verteilung der Strahlen des kleinen Datensatzes mit konkreten Werten für den Mittelwert, die Standardabweichung, das Minimum und Maximum sowie die Perzentile.
Diese Teilmenge des vollständigen Datensatzes wird anhand der Verteilung fuer diese Forschungsarbeit als repräsentativ angesehen und dient als Datensatz für die folgenden Forschungsdurchläufe.

5.

AI Modelle, Training & Evaluiierung

Dieser Abschnitt beschreibt die verschiedenen Modelle die Trainiert wurden in diversen Testdurchläufen. Insbesondere ist auf die verschiedenen genutzten Parameter zu achten die verwendet wurden.

In allen Trainings-/Testdurchläufen wurden die in Abschnitt 4 genannten Parameter (Ursprungspunkt und Richtung) für jeden Strahl als Features für das Modell verwendet. Der binäre Wert Hit wurde als Label verwendet für das Training und nachfolgende Evaluierung. Als Erinnerung ein mit 0 gelabelter Datensatz bedeutet, dass der Strahl die Geometrie nicht trifft, ein mit 1 gelabelter Datensatz bedeutet, dass der Strahl die Szenen-Geometrie schneidet. Ein Großteil der Modelle wurde nach dem Training Exportiert um für mögliche weitere Test zur Verfügung zu stehen, die Modell-ID hilft die Modelle den Testdurchläufen zuzuordnen und ist in den kommenden Abschnitt als oberstes bei den einzelnen Testen notiert.

Die Beschreibungen der einzelnen Durchläufe der Modelle ist hier nach Kategorie der Modelle sortiert, in der Forschungsarbeit selber wurde jedoch nach Reihenfolge der Modell-IDs vorgegangen. Insbesondere wurde zwischen lineare Classifier und Neural Networks öfters gewechselt, um diese beiden Modelltypen besser vergleichen zu können.

Es ist ebenfalls zu beachten, dass das gesamte Training in Perioden unterteilt wurde, sodass nach jeder Periode das der Fortschritt des Modelltrainings analysiert werden kann. Das aktuelle Modell wird mit dem Trainings- und Test-Set Evaluiert und die Ergebnisse werden für das Plotten der Konvergenz Kurven zwischen gespeichert. Durch diesen Monitor-Schritt verlängert sich das Modelltraining um ca. den doppelten bis dreifachen Zeitfaktor.

5.1. Linearer Regressor

Die ersten Testversuche wurden mit einem Linearen Regressor durchgeführt.
Ein linearer Regressor liefert ein floating Point-Wert zurück der jedoch keine direkte Aussage über mögliche Schnitte mit der Geometrie aussagt. Durch manuelles Testen konnten Werte außerhalb des erwarteten Werte Bereiches [0,1].

```
1.  import tensorflow as tf
2.
3.  # erstellen eines linearen Regressors
4.  estimator = tf.estimator.LinearRegressor(feature_columns)
```

Listing 5.1 Tensorflow Linearer Regressors

Test 1 - Linearer Regressor

Modell-ID: 1

Bei diesem 1. Test wurde ein naives herangehen gewählt. Hauptsächlich ging es darum eine generelle Pipeline zu erstellen um die in Abschnitt 4 Beschriebenen Datensätze so zu formatieren damit diese in einem Training mit Tensorflow genutzt werden können. Außerdem wurden hier die Trainings-Set und Validation-Set Größen sehr klein gewählt, damit generelle Programmier-Fehler schneller erkannt werden konnten. Es wurde der für lineare Regression typische Gradient Descent Optimizer gewählt um den Verlust zu verringern. Da das Modell ungeclampte floating Point Werte zurückgeliefert, wurde für das Training-Monitoring der RMS-Error verwendet. Zu beachten ist, dass keine Feature Crossing gewählt wurde noch wurden die Input-Parameter in anderer Form verarbeitet. Die nachfolgenden Tabelle 6 zeigt die verwendeten Parameter des Modells.

Tabelle 6 Modell-Parameter Linearer Regressor Test 1

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size
10	0,0001	500	10	400000	100000

Das Training diese Modells dauerte 1 Stunde und 10 min.
Abbildung 24 zeigt die Konvergenzkurven der beiden Datensätze über die 10 Perioden des Trainings.

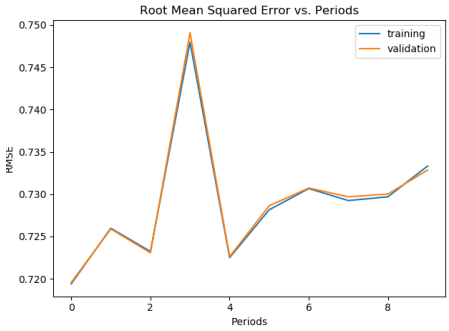

Abbildung 24 Linearer Regressor Test 1 Konvergenz-Kurve

Es kann klar erkannt werden, dass dieser naive Ansatz kein gutes Ergebnis liefert, da der Verlust im Laufe der Periode immer größer wird von einem Anfänglichen Log Loss von .72 bis zu einem Finalen Log Loss von .73. Auch ist der Peak in Periode 3 zu beachten. Das Modell scheint ein recht zufälliges Ergebniss zurück zu liefern, was insbesondere an den steigenden Konvergenz Kurven zu erkennen ist und zusätzlich an den Schwankenden Abständen der Trainings- und Validation-Kurve.
Anhand dieser sehr negativen Ergebnisse wurde auf eine zusätzliche Evaluiierung anhand eines Trainings-Datensatzes verzichtet. Jedoch konnte an dieser Stelle die Infrastruktur für weitere Modelle erstellt werden, die mithilfe der Definierten Input-Funktionen und Tensorflow arbeiten können. Da es sich bei diesem Modell um ein erster Prototyp handelte, wurde dieses Modell in diesem Schritt nicht exportiert.

Test 2 – Linear Regressor

Model-ID: 2

In einem weiteren Versuch wurde auf der bereits beschriebenen Struktur aufgebaut und hauptsächlich wurden die Größe des Datensatzes verändert.
Die generellen Modellparameter wurden nicht verändert, wie in der Parameter Tabelle 7 zu sehen ist. Es wurde weiterhin der Gradient Descent Optimizer genutzt um den Verlust zu minimieren.
Lediglich der Programmcode der Input-Funktion wurde etwas verändert, was sich ggf. auf die Resultate des Modells auswirkt, jedoch sind die Parameter dieselben wie in Absatz 4.2 beschrieben. Zusätzlich wurde in diesem Test eine Permutation der Testdaten erstellt, sodass eine größere Zufälligkeit vorliegt.

Tabelle 7 Modell-Parameter Linearer Regressor Test 2

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,0001	500	10	671088	167772	209715

Tabelle 7 zeigt insbesondere, dass die Größe des Trainings-Sets um den Faktor 1.6 erhöht wurde.

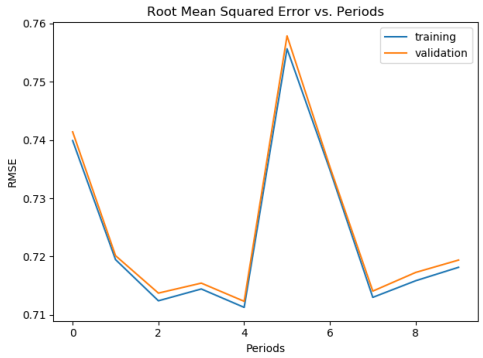

Abbildung 25 Linearer Regressor Test 2 Konvergenz-Kurve

Durch die gestiegene Größe des Trainings-Sets verlängerte sich bei diesem Testdurchlauf ebenfalls die Trainings-Dauer. Die gesamte Dauer des Trainings und des Monitorings während des Trainings betrug bei diesem Durchlauf 6 Stunden.
Das Modell wurde nach dem Training mit dem Testdatensatz evaluiert und ein finaler Root Mean Squared Error von 0.73 konnte ermittelt werden. Dieser ist geringfügig schlechter als der RMSE der in der letzten Periode des Trainings an dem Trainings-Set und Validation-Set ermittelt werden konnte.

Abbildung 25 zeigt, dass sich das Modell durch das Training in den ersten 4 Perioden verbessert. Der Peak zwischen der 4 und 7 Periode ist wieder ein Indiz, dafür, dass ein lineares Regressions Modell an dieser Stelle scheinbar willkürlich die Gewichte ändert dies einen fall um knapp 0.05 Schritte verursacht. Jedoch bleibt im Gegensatz zu den Fluktuationen im Test 1 die Kurve des Validations-Sets wie zu erwartend kontinuierlich etwas über der Trainings-Set-Kurve.

Anhand der beschriebenen Testfälle konnte erkannt werden, dass sich ein lineares Regressions Modell nicht optimal für die Lösung dieses Problems eignet,

insbesondere kann hier nicht eindeutig die Ausgabe des Modells gedeutet werden, da der Wertebereich wie in der Einleitung beschrieben an einigen Stellen stark von der Binären Einstufung ob ein Strahl die Geometrie schneidet oder nicht abweicht. Weitere Versuche, das Problem mit einem linearen Regressor (ggf. mit anderen Parametern) wurden nicht unternommen.

5.2. Linear Classifier

```
1.  import tensorflow as tf
2.
3.  # erstellen eines linearen Klassifizierers
4.  estimator = tf.estimator.LinearClassifier(feature_columns)
```

Listing 5.2 Tensorflow Linearer Classifier

Da es sich bei dem vorliegenden Problem generell um ein Klassifizierungsproblem handelt ist ein linearer Klassifizierer der zweite Ansatz der für dieses Problem gewählt wurde. Hierbei wird das Label als Kategorien betrachtet, es gibt demnach die Kategorie 0, die für ein nicht treffen der Szenen Geometrie steht. Die Kategorie 1 steht für ein treffen der Szenen Geometrie. Das Modell liefert eine Wahrscheinlichkeitsprognose der Zugehörigkeit der Stichprobe (des Strahls) zu jeder der beiden Kategorien, die Summe beider Prozentwerte summiert sich auf 1 auf. Wie in Absatz 0 beschrieben wird für lineare Klassifizierungsmodelle als Verlust Funktion Log Loss verwendet, dies bedeutet auch, dass die Plots nicht direkt mit den Konvergenzkurven aus den Testdurchläufen des Linearen Regressors verglichen werden können.

Test 3 – Linear Classifier

Der erste Test mit einem linearen Klassifizier Modelle wurde mit ähnlichen Parametern wie denen des linearen Regressor durchgeführt. Es wurde weiterhin der Gradient Decent Optimizer zur Minimierung des Verlustes beim Training gewählt. Tabelle 8 liefert eine Übersicht über die genutzten Parameter.

Tabelle 8 Modell-Parameter Linear Classifier Test 3

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size
10	0,0001	500	10	671088	167772

Die Dauer des Trainings dieses Modell betrug 2 Stunden und 30 min.

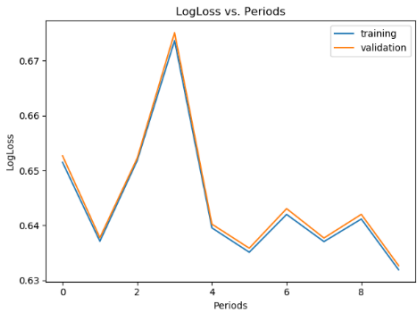

Abbildung 26 Linear Classifier Test 3 Konvergenz-Kurve

Abbildung 26 zeigt, wie das sich Modell mit seinen Prognosen über das Training entwickelt. Wieder ist zu sehen, dass das Modell etwas schlechter beim Validation-Set ist als beim Trainings-Set, was jedoch erwartet wurde. Für die Evaluierung wurde (wie bei der Verlust Funktion des Modells) der Log Loss verwendet, daher ist hier, wie in der Einleitung erwähnt, ein neuer Wertebereich auf der Y-Achse des Plots, als bei dem Linearen Regressor was ein Vergleich das bisherige Modell mit diesem erschwert. Zwar kann sich das Modell über die 10 Perioden von einem Wert von 0.65 auf knapp 0.63 verbessern, jedoch sind die Schwankungen und ebenfalls der Peak in Periode 3 zu beachten. Eine kontinuierliche Verbesserung des Modells, welche an einer gleichmäßig sinkenden konvergenz-Kurve zu erkennen wäre, ist aus diesem Testlauf nicht ersichtbar. Eine Evaluierung an dem Test-Set wurde in diesem Durchlauf nicht durchgeführt.

Test 4 – Linear Classifier

Dieser Test baut auf dem vorherigen Test 3 auf. Die Parameter wurden beibehalten, wie in Tabelle 9 Modell-Parameter Linear Classifier Test 4 erkennbar. Außerdem wurde weiterhin der Gradient Descent Optimizer verwendet.
Zusätzlich wurde nach abgeschlossenem Training die Genauigkeit (engl. accuracy) sowie die Fläche unter der Grenzwertoptimierungskurve (AUC) berechnet. Die genaue Bedeutung des Accuracy-Wertes wird in Absatz 6**.** diskutiert, für die folgende Testfälle kann die Accuracy als Qualitätsmaß für das Modell angesehen werden.

Tabelle 9 Modell-Parameter Linear Classifier Test 4

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size
10	0,0001	500	10	671088	167772

Das Modelltraining dauerte an dieser Stelle 3 Stunden. Die zeitliche Schwankung bei gleichen Modell-Parametern & -Struktur lässt sich durch das Betriebssystem sowie parallel ausgeführte Anwendungen erklären.

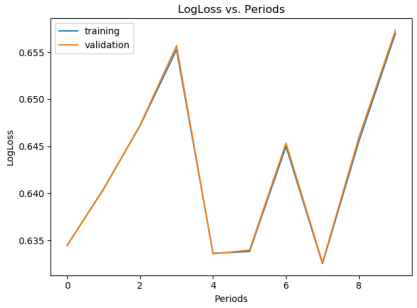

Abbildung 27 Linear Classifier Test 4 Konvergenz-Kurve

Es ist an Abbildung 27 zu erkennen, dass es keine extremen Peaks bei diesem Training gibt (die Skalierung der Log Loss Ausgabe ist feiner), jedoch ist das Modell weiterhin sehr fluktuierend und innerhalb der 10 Perioden ist keine klare Verbesserung erkennbar sein.
Die Accuracy nach dem Modell Training konnte mit 0.64 festgestellt werden. Dementsprechend liegt das Modell mit 64% der Prognosen richtig, dies ist jedoch nur marginal besser als ein Algorithmus der Zufällig die Strahlen klassifiziert und im Schnitt 50% richtige Ergebnisse liefert. Die Fläche unter der ROC Kurve konnte als der Wert 0.68 ermittelt werden.

Test 6 – Linear Classifier

In diesem Test wurden dieselben Parameter wie in Test 4 genutzt. Dieser Test galt der Stabilität des Modells, es sollte untersucht werden ob es große Schwankungen insbesondere bezüglich der neu eingeführten AUC- und Accuracy-Werte gibt. Da sich die Parameter nicht unterscheiden und wieder der Gradient Descent Optimizer genutzt wurde wird auf die genauere Analyse dieser an dieser Stelle verzichtet.

Tabelle 10 Modell-Parameter Linear Classifier Test 6

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size
10	0,0001	500	10	671088	167772

Das Training des Modells dauerte ebenfalls 3 Stunden.

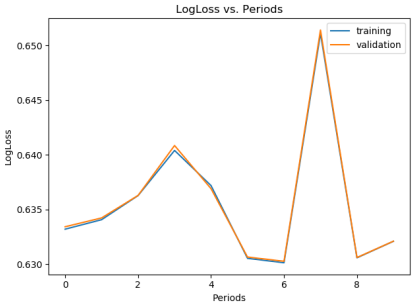

Abbildung 28 Linear Classifier Test 6 Konvergenz-Kurve

Der finale AUC-Wert betrug 0.68 und der Accuracy-Wert 0.66.
An dem Ergebnis dieses Trainings (siehe Abbildung 28) ist festzustellen, dass sich das Modell nur bedingt von dem vorherigen unterscheidet. Hier ist eine Stärke des linearen Klassifizierungsmodells fest zu stellen, bei ähnlichen Parametern, erlernt sich dieses Modell in einer ähnlichen weise bei individuellen Trainings, insbesondre im Vergleich zu Neuronalen Netzwerken ergibt sich hier ein Vorteil dieser Modelle. Eine genauere Analyse und Diskussion ist in Abschnitt 6 zu finden.

Test 8 – Linear Classifier

In diesem Durchlauf wurden in besondere ein alternativer Optimierung Algorithmus gewählt, der AdamOptimizer. Beim Adam (adaptive moment estimation) Algorithmus handelt es sich um eine Abwandlung des stochastischen Gradient Descent Algorithmus, der laut der Autoren effizient ist, wenig Speicherverbrauch hat und insbesondere für Probleme mit großen Datenmengen vorgesehen ist [11].
Insbesondere wird die Learning Rate für jedes Gewicht des Netzwerkes individuell angepasst während des Trainings.
Des Weiteren wurden die Batch-Size in diesem Durchlauf erhöht, sodass mehr Werte in den Optimierungsschritt einbezogen werden. Die Learning Rate wurde initial auf

0,001 gesetzt, was einer Erhöhung um den Faktor 10 gegenüber den bisherigen Durchläufen ist, insbesondere soll dadurch das Modell schneller konvergieren.
Auf der anderen Seite wurde jedoch die Schrittanzahl auf 800 erhöht, was generell die Trainingszeit verlängert. Zusätzlich zur Evaluierung mit einem Validation-Set wurde ein Test-Set gewählt, das zum unabhängigen Vergleich für verschiedene Modelle dienen soll.

Tabelle 11 Modell-Parameter Linear Classifier Test 8

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,001	800	40	671088	167772	209715

Durch die veränderten Parameter verlängerte sich die Trainings Zeit des Modells, das Training dauerte 3 Stunden und 30 Minuten.

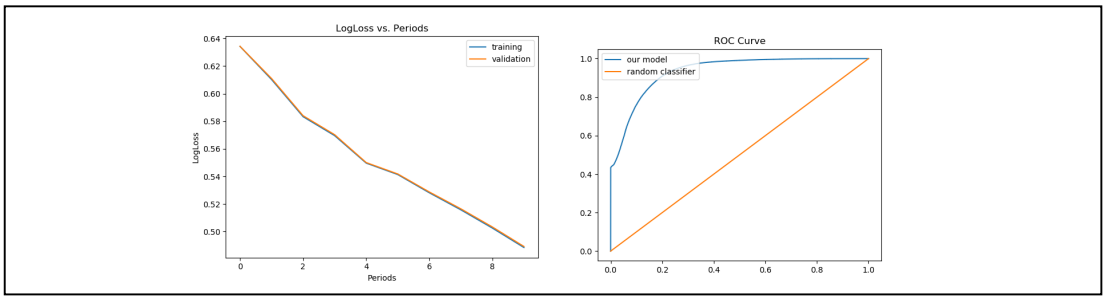

Abbildung 29 Linear Classifier Test 8 Konvergenz-Kurve (links), ROC Kurve (rechts)

Das Modell lieferte nach abgeschlossenem Training einen AUC-Wert von 0.93 und einen Accuracy-Wert von 0.80 beim Validation-Set, die ROC des Validation-Set ist in Abbildung 29 (rechts) geplottet. Beim Test-Set konnte ebenfalls ein AUC-Wert von 0.93 ermittelt werden, der Accuracy-Wert ist minimal schlechter und beträgt 0.79, was jedoch zu erwartet wurde.
Dieser Durchlauf mit dem Adam Algorithmus zur Optimierung konnte sehr positive Werte liefern, insbesondere kann an der Konvergenz Kurve auf der linken Seite in Abbildung 29 ein erwartetes Trainingsverhalten des Modells erkannt werden. Es konnte eine kontinuierliche sinkende Konvergenz Kurve ermittelt werden über die gesamten Trainingsperioden. Über 10 Perioden konnte der Log Loss von 0.64 auf 0.50 gesenkt werden. Eine Korrektheit der Prognosen von ca. 80% konnte bei dem

Validation-Set sowie dem Test-Set ermittelt werden.
Die Ursprung der starke Verbesserung der Modellergebnisse zu den vorherigen Durchläufen wurden durch die Änderung der Optimierungs-Algorithmus angenommen.

Test 9 – Linear Classifier

Aufbauend auf den sehr positiven Ergebnissen des Test 8 wurde in diesem Durchlauf weiterhin der AdamOptimizier genutzt insbesondere zur Validierung der Thesis, dass dieser die starke Verbesserung gegenüber dem generell Gradient Descent Optimizer gebracht hat. Hauptsächlich wurden in diesem Durchlauf minimal die Learning Rate und die Schrittanzahl des Optimier Algorithmus angepasst, alle weiteren Parameter wurden wie in Tabelle 12 ersichtlich gegenüber dem vorherigen Durchlauf beibehalten.

Tabelle 12 Modell-Parameter Linear Classifier Test 9

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,002	900	40	671088	167772	209715

Es konnte wieder eine Verlängerung der Trainingsdauer gemessen werden, die Trainingsdauer betrug in diesem Durchlauf 4 Stunden, dies ist hauptsächlich durch die Erhöhung der Schrittzahl um 100 zurückzuführen.

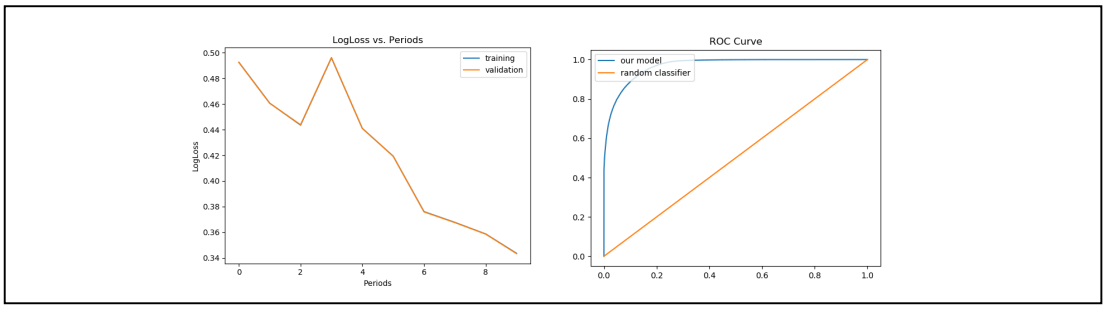

Abbildung 30 Linear Classifier Test 9 Konvergenz-Kurve (links), ROC Kurve (rechts)

Beim Validation-Set konnte nach abgeschlossenem Training des Modells ein AUC-Wert von 0.97 und eine Accuracy von 0.87 ermittelt werden, dieselben Werten konnte bei dem Test-Set ermittelt werden.

Im Vergleich zum vorhergehenden Test konnten die Genauigkeit um weitere 7% zum vorhergehenden Modell verbessert werden. In Abbildung 30 kann erkannt werden, dass die ROC Kurve stärker steigt und sich der AUC-Wert weiter dem Wert 1.0 annähert. Insbesondere kann dieser Kurve entnommen werden, dass knapp 70% der negativen Klassifizierungen korrekt sind. Die Konvergenzkurve sinkt stetig über das Training abgesehen von dem Peak in Periode 3, es ist jedoch fest zu stellen, dass sich die Kurve des Trainings-Set kaum von der des Validation-Sets unterscheidet, dies wurde dadurch erklärt, dass eine generell starke Ähnlichkeit zwischen allen Strahlen besteht von denen 64% Prozent für das Training verwendet wurden (siehe Tabelle 12).
Es ist zu bemerken, dass das Training dieses Modells bereits bei einem Log Loss-Wert von knapp 0.50 startet, dieses ist der Wert, den das Model des vorhergehenden Tests 8 in der letzten Periode erreicht. Hier wird angenommen das das Modell 9 auf den des Modells 8 aufbaut und dessen Werte initial übernimmt, da sich die beiden Modelle in demselben temporären Speicherbereich befanden. Dafür spricht auch, dass Modell 8 bis zur letzten Periode eine stetig sinkende Konvergenzkurve besitzt.
Außerdem könnte diese Hypothese den Peak in Periode 3 des Modells 9 erklären, der dadurch entsteht, dass die Historie des Modells 8 nicht übernommen wurde.
Mit knapp 88% Trefferquote konnte dieses Modell als bestes Modell dieser Testreihe bewertet werden.

Test 13 – Linear Classifier

Für das Modell in Test 13 wurde der Feature Vektor #2 (vgl. Abschnitt 4.2) verwendet. Dieser besitzt als synthetische Feature die Winkel zu den Standardbasisvektoren in Radiant.
Um ein Vergleich mit vorherigen Modellen zu haben wurden die in weiteren Modellparametern (vgl. Tabelle 13) der vorherigen Durchlaufe beibehalten und es wurde wieder der AdamOptizimer gewählt.

Tabelle 13 Modell-Parameter Linear Classifier Test 13

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,002	900	40	671088	167772	209715

Das Modelltraining dauerte wie bei den Vorgängern um die 4 Stunden und 30 min, es ist demnach kein sehr großer Anstieg der Trainingsdauer durch die Vergrößerung der Input-Feature-Komplexität zu erkennen.

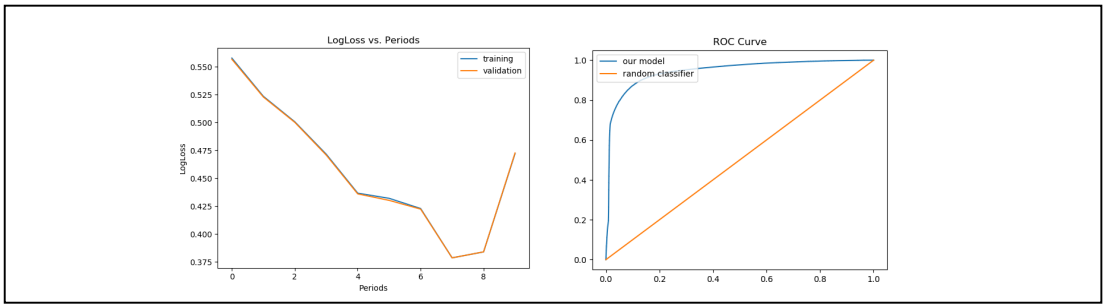

Abbildung 31 Linear Classifier Test 13 Konvergenz-Kurve (links), ROC Kurve (rechts)

An Abbildung 31 ist zu erkennen, dass das Modell beim Training eine kontinuierliche Verbesserung über die 10 Trainingsperioden erfahren kann.
Das Optimum ist bei Periode 7 von welchem Punkt die Konvergenz-Kurven wieder zu steigen beginnen. Es kann im Vergleich zu den Trainingsdurchläufen mit dem Feature-Vektor #1 (vgl. Test 8) erkannt werden, dass bei den Konvergenz-Kurven eine minimale Verbesserung innerhalb des Trainings bis zu dem Optimum (logloss ~0.38) in Periode 7 des Modells in Test 13 zu sehen ist.
Die finale Evaluierung mit dem Test-Set und dem Validierungs-Set ergab folgenden Werte. Der AUC-Wert bei beiden Sets Betrug 0.95 die Genauigkeit dieser Betrug beim Validation-Set ein Wert von 0.73 und beim Test-Set ein etwas besseren Wert von 0.75. An der ROC-Kurve in Abbildung 31 ist zusätzlich zu erkennen, dass die Kategorie Zuweisungsgenauigkeit des Modells (insbesondere bei der positiven Kategorie) nicht so sehr abrupt ist wie bei Test 8 (vgl. Abbildung 29).
Es ist zu beachten, dass die Evaluierung auf dem Modellzustand der letzten Trainingsperiode beruht und daher eine voraussichtlich schlechteres Ergebnis lieferte als, wäre es mit dem Zustand beim Optimum in Periode 7 validiert worden.
Dementsprechend kann allgemein festgestellt werden, dass das synthetische Feature, welches auf den Richtungscosinus basiert, zu einer leichte Verbesserung des Modells beim Training führt.

5.3. Neuronal Network

In diesem Abschnitt wird die Testreihe des Deep-Neural-Network Classifiers an dem Datensatz zur Lösung des vorliegenden Problems beschrieben.
Der folgende Code Ausschnitt zeigt den in der Tensorflow-API integrierten Deep-Neural Network Classifier. Im Vergleich zu den bisherigen linearen Modellen wird an diesen ein Array (*hidden_units*) übergeben. Die Größe des Arrays gibt die Anzahl der Layer im Neuronalen Netz an, die jeweiligen Werte innerhalb des Arrays geben jeweils die Anzahl der Neuronen in jedem Layer an.

```
1.  import tensorflow as tf
2.
3.  # erstellen eines DNNClasifier regressor Objektes
4.  # hidden_units gibt die Anzahl der Layer und die jeweils
5.  # enthaltenen Neuronen an
6.  estimator = tf.estimator.
    DNNClassifier(feature_columns,hidden_units=[512,512,24])
```

Listing 5.3 Tensorflow Deep Neural Network Klassifizierer

Bei der Verwendung von neuronalen Netzen ist generell davon auszugehen, dass das Training dieser länger dauert als von linearen Modellen, insbesondere aufgrund der Komplexität dieser. Zusätzlich kann sich die zufällige Verteilung der initialen Gewichte stark auf das Netzwerk auswirken, sodass trotz gleichen Parametern und Input-Daten je nach Durchlauf verschiedene Ergebnisse herauskommen können.

Test 5 – Deep Neural Network Classifier

In einem ersten Versuch wurde ein Neuronales Netzwerk mit den in Tabelle 14 gezeigten Parametern trainiert. Wie in den ersten Testen der linearen Modelle wurde auch bei diesem Neuronalen Netze zuerst ein Gradient Descent Optimizer verwendet. Insbesondere ist zu beachten, dass eine sehr hohe Schrittzahl und eine große Batch-Size gewählt wurde. Als Netzwerk Struktur wurden in diesem Durchgang 3 Layer erstellt bei denen das 1. Layer 1024 Neuronen besitzt, im 2. Layer wurden 512 und im 3. Layer 256 definiert. Das *hidden_units* Array sieht dementsprechend folgendermaßen aus: [1024, 512, 256].

Tabelle 14 Modell-Parameter Deep Neural Network Classifier Test 5

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size

10	0,001	1000	40	671088	167772	209715

Das Training dieses Modells dauerte 5 Stunden und 30 Minuten. Wie in der Einleitung dieses Absatzes beschrieben ist eine längere Trainingszeit bei Neuronalen Netzen zu erwarten, zusätzlich kann aus Tabelle 14 entnommen werden, dass eine sehr hohe Optimierungs-schrittzahl für das Training gewählt wurde.

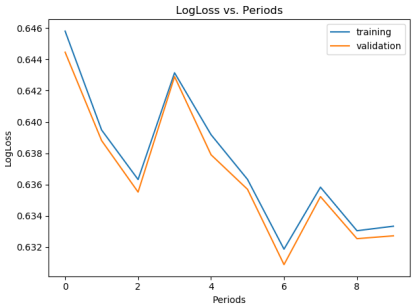

Abbildung 32 Deep Neural Network Classifier Test 5 Konvergenz-Kurve

Der finale Accuracy Wert beim Validation-Set konnte als 0.59 ermittelt werden, der AUC-Wert betrug 0.75.
Das Modelle wurde ebenfalls mit einem Test-Set evaluiert und der Accuracy-Wert bei diesem Durchlauf betrug 0.58 und der AUC-Wert ebenfalls 0.75. Dieser minimale Unterschied zwischen den beiden Sets lässt sich an dieser Stelle dadurch erklären, dass ein Genauigkeitswert von 0.59 nur minimal besser ist als bei einem Random Classifier.
Die Konvergenz-Kurve in Abbildung 32 zeigt ein Lernfortschritt des Modells über die 10 Perioden, die zwei Peaks besitzt bei denen die Modell Qualität nachlässt, trotzdem kann eine geringe Verbesserung von einem Log Loss von 0.645 auf 0.635 beobachtet werden.
Für die Länge der Trainingsdauer ist das Ergebnis dieses Modells jedoch sehr ernüchternd. Das Ergebnis könnte auf suboptimale Initialparameter zurückzuführen sein, aufgrund der langen Trainingsdauer und des sehr geringen Erfolgswertes wurde jedoch auf eine Wiederholung des Trainings mit diesen Parametern verzichtet.

Test 7 – Deep Neural Network Classifier

Nach einem wenig erfolgsversprechendem 1. Versuch mit einem Neuronalen Netzwerk als Modell wurde ein alternativer Optimierungsalgorithmus in diesem Test

gewählt. Es wurde der FTRL Optimizer gewählt für die Optimierung des Netzwerkes, zusätzlich wurde eine L1 Regulierung mit einer Regulierungsstärke von 0.1 gewählt aufgrund der Komplexität, die ein Neuronales Netzwerk annehmen kann.
Während die Parameter (siehe Tabelle 15) zum vorherigen Testdurchlauf nicht verändert wurden, wurde in der Netzwerkstruktur ein weiteres Layer mit 512 Neuronen ergänzt. Dadurch besteht das Netzwerk im 1. Layer aus 1024 Neuronen, im 2. & 3. Layer aus jeweils 512 Neuronen und im letzten Layer aus 256 Neuronen.
Das *hidden_units* Array sieht dementsprechend folgendermaßen aus: [1024, 512, 256].

Tabelle 15 Modell-Parameter Deep Neural Network Classifier Test 7

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size
10	0,001	1000	40	671088	167772

Das Training dieses Modells dauerte besonders lange, insgesamt betrug die Dauer über die 10 Periode 8 Stunden.

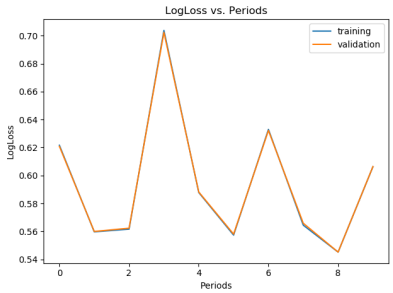

Abbildung 33 Deep Neural Network Classifier Test 7
Konvergenz-Kurve

Nach erfolgreichem Training wurde das Modell an dem Validation-Set evaluiert und es konnte ein Accuracy-Wert von 0.54 ermittelt werden sowie ein AUC-Wert von 0.5. Abbildung 33 zeigt den Trainingsverlauf dieses Modell, es ist zu beobachten, dass dieses sehr unruhig verläuft. Es sind Unterschiede des Log Loss-Wertes von knapp 0.15 zu sehen insbesondere zwischen der 2. und 3. Trainingsperiode steigt der Log Loss extrem.
Diese extremen Schwankungen sind auf den verwendeten Optimierungsalgorithmus

des Modells zurück zu führen. Der FTRL-Algorithmus ist speziell für sich ändernde Daten, wie z.B. User Meinungen/Befragungen vorgesehen, über das Training passt dieser sich insbesondere der neuen Daten an und es werden z.T. erlernte Werte eher verworfen [12]. Insbesondere ist das Modell für online Modelle (Modelle die kontinuierlich lernen) vorgesehen. Da es sich bei dem vorliegenden Problem um ein geometrisches Problem handelt und sich die Gesetzmäßigkeiten für dieses nicht (wie z.B. Ansichten/Meinungen) über verschieden Perioden verändern ist der verwendete Optimierungsalgorithmus nicht optimal hierfür.
Insbesondere sagt der AUC-Wert von 0.5 aus, dass es sich um eins der schlechtesten Modelle handelt, da es sich dabei um den schlecht möglichsten AUC-Wert handelt, der erreicht werden kann.
Aufgrund der geringen Erfolgsquote des Modells wurde auf eine Evaluierung dieses anhand eines Test-Sets verzichtet.

Test 10 – Deep Neural Network Classifier

In Test 10 wurde der AdamOptimizer verwendet, dieser ist bereits bekannt aus den Linear Classifier Testreihen in Abschnitt 5.2.
Die Parameter diese Testes sind aufbauend auf dem Test 9 mit dem Linearen Classifier, bei dem ein Wert von 0.89 als finale Accuracy gemessen werden konnte (siehe Tabelle 16).
Die Struktur dieses Modell wurde stark verändert im Vergleich zu den vorherigen neuronalen Netzwerken. Es wurden 5 Layer erstellt, bei denen die ersten 2 Layer jeweils 30 Neuronen besitzen. Das 3. & 4. Layer besitzen jeweils 20 Neuronen und das letzte Layer besitzt 10 Neuronen.
Das hidden_units Array sieht dementsprechend folgendermaßen aus: [30, 30, 20, 20, 10].

Tabelle 16 Modell-Parameter Deep Neural Network Classifier Test 10

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,002	900	40	671088	167772	209715

Die Trainingsdauer dieses Modell betrug 5 Stunden.

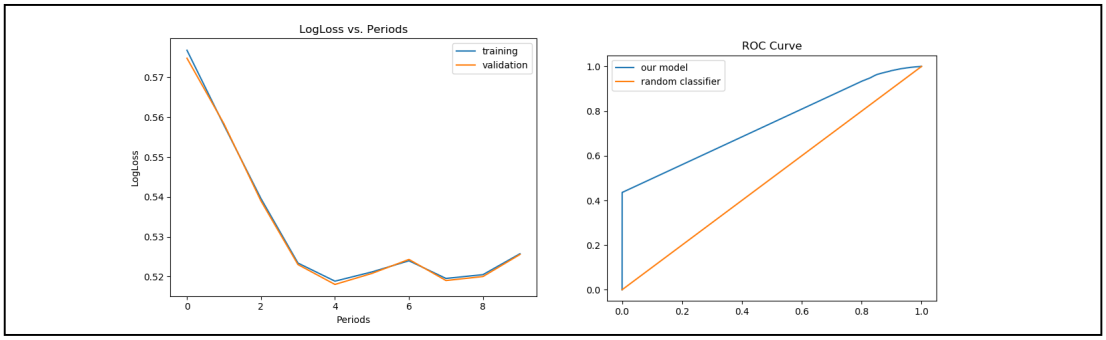

Abbildung 34 Deep Neural Network Classifier Test 10 Konvergenz-Kurve (links), ROC Kurve (rechts)

Abbildung 34 zeigt den Verlauf des Trainings über die 10 Periode sowie die ROC-Kurve die anhand des Validations-Sets ermittelt wurde.
Die finalen Wert des AUCs beim Validation- und Test-Sets betrugen beide 0.74, der Accuracy Wert unterscheidet sich minimal, 0.70 beim Validation-Set und 0.69 beim Test-Set.
Wie beim Linearen Classifier konnten bei der Nutzung des AdamOptimziers auch beim Neuronalen Netzwerk Verbesserungen gegenüber den anderen Optimierungsalgorithmen beobachtet werden.
Die ermittelten Genauigkeits-Wert des Neuronalen Netzwerkes liegen weit über einem Random Classifier und den bisher ermittelten Werten der vorhergehenden Neuronalen Netzwerke.
An der Konvergenz Kurve des Trainings kann bis zur 4. Periode eine kontinuierliche Verbesserung erkannt werden, jedoch kann ab Periode 4 keine weitere Verbesserung erzielt werden. Die ROC Kurve zeigt, dass ca. 40% der Positiven Klassifizierung korrekt identifiziert werden können, während das Modell lediglich knapp 10% der negativen Klassifizierungen korrekt identifiziert.

Test 11 – Deep Neural Network Classifier

Dieser Test baut auf dem vorherigen Test auf und es werden sämtliche Parameter beibehalten (siehe Tabelle 17), lediglich die Struktur des Netzwerkes wurde verändert. Die Anzahl der Neuronen wurde in jedem der 5 Layer jeweils um den Faktor 10 erhöht im Vergleich zum vorherigen Modell. Daher wird folgendes Array für die hidden_units verwendet: [300, 300, 200, 200, 100].

Tabelle 17 Modell-Parameter Deep Neural Network Classifier Test 11

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,002	900	40	671088	167772	209715

Durch die Vergrößerung der Netzwerk Struktur erhöht sich die Dauer des Trainings und eine gesamt Dauer von 6 Stunden wurde gemessen.

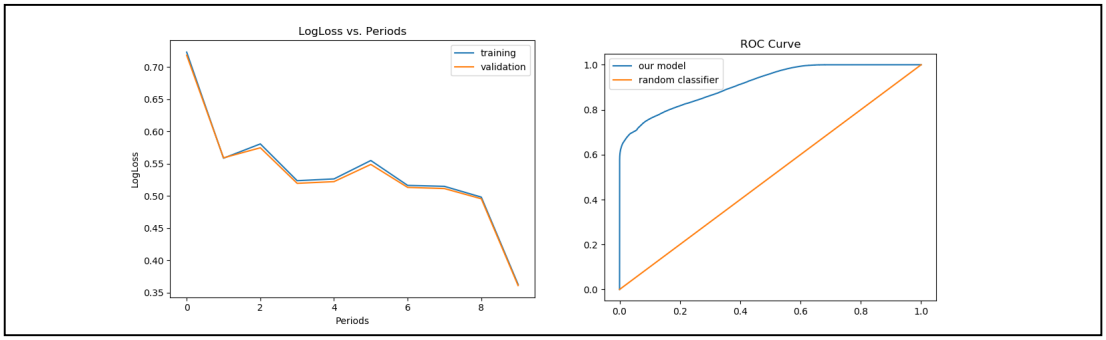

Abbildung 35 Deep Neural Network Classifier Test 11 Konvergenz-Kurve (links), ROC Kurve (rechts)

Das Modell kann nach abgeschlossenem Training beim Validation-Set einen AUC-Wert von 0.91 erreichen und eine Genauigkeit von 0.82. Beim Test-Set kann sogar ein AUC-Wert von 0.94 und eine Genauigkeit von 0.85 ermittelt werden.
Die Konvergenz-Kurve in Abbildung 35 zeigt, dass das Modell abgesehen von einigen kleinen Schwankungen eine kontinuierliche Verbesserung erreicht und sich der Log Loss über die 10 Perioden von 0.70 auf knapp 0.35 verbessern kann. Insbesondere können 2 größere Sprünge in diesem Training beobachtet werden, diese befanden sich zwischen der 0. und 1. Periode sowie von der 8. zur 9. Periode.
An der ROC Kurve in derselben Abbildung kann eine erkannt werden, dass knapp 60% der positiven Klassifizierungen sind und knapp 40% der negativen Klassifizierungen korrekt sind. Insgesamt konnte jedoch eine Erhöhung der Neuronen Anzahl zu einer Verbesserung des Ergebnisses führen.

Test 12 – Deep Neural Network Classifier

Aufbauend auf der Beobachtung in Test 11, dass eine erhöhte Neuronen Anzahl/Netzkomplexität zu einer Verbesserung der Lernergebnisses des Modells beitrug, wurde die Netzkomplexität in diesem Test erneut vergrößert.
Die Neuronen-Anzahlen wurden in den 5 Layern jeweils um den Faktor 2 erhöht.

Dies führt zu folgendem Array der hidden_units: [600, 600, 400, 400, 200].
Wie aus der Tabelle 18 zu entnehmen ist, wurden die Parameter des Modells weiterhin beibehalten. Dasselbe gilt für die Größen der 3 verwendeten Datensätze.

Tabelle 18 Modell-Parameter Deep Neural Network Classifier Test 12

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,002	900	40	671088	167772	209715

Das Training verlängerte sich auch in diesem Durchlauf und dies führte zu einer Gesamtdauer von ca. 6 Stunden und 30 Minuten. Auch in diesem Fall wird diese Verlängerung auf die gestiegene Komplexität des Netzes zurückzuführen sein.

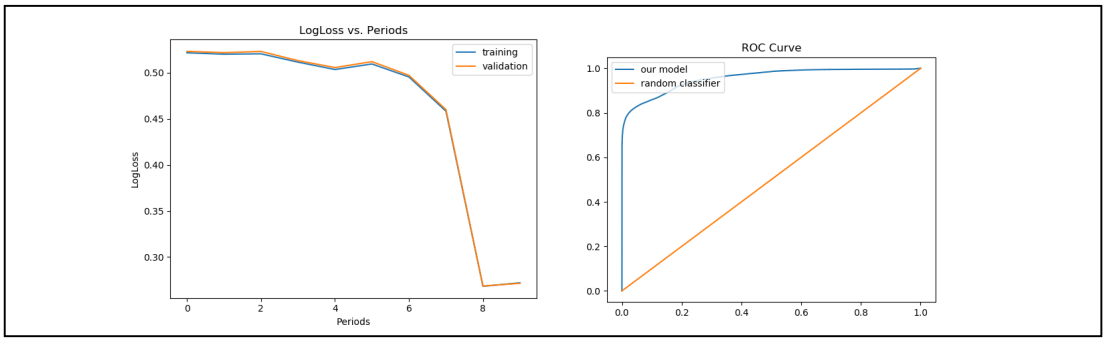

Abbildung 36 Deep Neural Network Classifier Test 12 Konvergenz-Kurve (links), ROC Kurve (rechts)

Nach abgeschlossenem Training konnte für das Validation-Set ein AUC-Wert von 0.96 ermittelt werden und eine Genauigkeit von 0.87. Für das Test-Set konnte ein AUC-Wert von 0.95 ermittelt werden und ein Genauigkeits-Wert von 0.86.
Die Abbildung 36 zeigt die Konvergenz-Kurve des Modells über die 10 Trainingsperioden. Insbesondere kann ein großer Sprung des Log Loss von der 7. zur 8. Periode erkannt werden, innerhalb dieser Trainingsperiode sinkt der Log Loss von ca. 0.45 zu knapp 0.29.
Dies könnte auf die Zufälligkeit von Neuronalen Netzen zurück zu führen sein, es ist jedoch auffällig, dass diese Verbesserung erst in der 7. Periode Auftritt. Es ist jedoch zwischen der 5. und 7. Periode bereits einen geringes Absinken des Log Loss zu beobachten welches auf diese starke Verbesserung hindeuten könnte.
Des Weiteren kann dem Plot entnommen werden, dass sich die Trainings und die

Validation Kurve sehr stark beieinander sind.
An der ROC Curve kann erkannt werden, dass das Modell wieder besser die positiven Ergebnisse Klassifiziert als die negativen, jedoch konnte sich dies zu einer knapp 80% korrekten positiven Klassifizierung und ca. 65-70% korrekten negativen Klassifizierung.

Test 14 & 15 – Deep Neural Network Classifier

Die Tests 14 und 15 sind aufeinander aufbauend und unterscheiden sich lediglich daran, dass der Test 15 mit erlernten Werten von Modelltraining 14 weiter lernt. Anders gesagt wurde das Training mit doppelter Länge durchgeführt.
Es wurde weiterhin der AdamOptimizer gewählt und die Batch-Size wie in Test 12 beibehalten. (hidden_units: [600, 600, 400, 400, 200])
Als Variation wurde für diesen Durchlauf der Feature Vektor #2 gewählt und dadurch die Richtungscosinus mit beachtet.
Diese Testreihe sollte insbesondere durch die zwei aufeinander aufbauenden Trainings eine maximale Genauigkeit unter Vernachlässigung der Zeitkomponente erforschen.
Tabelle 19 zeigt die für jeweils beide Trainings verwendeten Parameter, es ist zu beachten, das insgesamt Trainings 20 Perioden durchgeführt wurden.

Tabelle 19 Modell-Parameter Deep Neural Network Classifier Test 14 & 15

Periods	Learning Rate	Steps	Batch_Size	Training Size	Validation Size	Test Size
10	0,002	900	40	671088	167772	209715

Bei den neuronalen Netzen wirkte sich das Feature Engineering stärker auf die die Dauer des Modelltrainings aus, sodass für Test 14 ca. 8 Stunden Trainingszeit benötigt wurden und für Test 15 weitere 9 Stunden, sodass eine Gesamtsumme von 17 Stunden Trainingszeit bei diesen beiden Tests gemessen wurde.

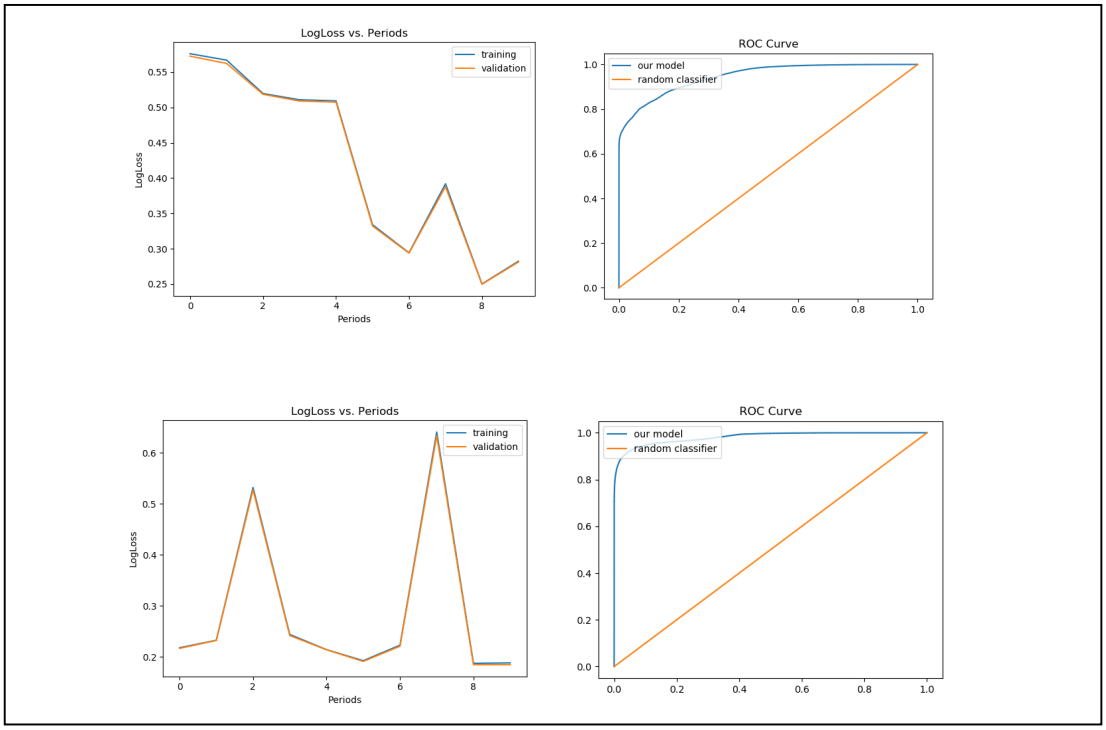

Abbildung 37 Deep Neural Network Classifier Test 14 und 15 Konvergenz-Kurve (Test 14 oben links, Test 15 unten links), ROC Kurve (Test 14 oben rechts, Test 15 unten rechts)

An den Abbildung 37 kann erkannt werden, dass sich während des Trainings des Modells 14 dies fast kontinuierlich eine verbessern kann, einzige Ausnahme ist die Periode 7 in der sich das Modell im Vergleich zur vorherigen Periode verschlechtert. An der ROC Kurve dieses Trainings kann erkannt werden, dass sich das trainierte Modell tendenziell wieder eine stärke Sicherheit bezüglich der positiven Kategorie besitzt.

Es kann ein finaler Genauigkeitswert des Modells von 0.86 gemessen werden und ein AUC-Wert von 0.95 bei dem Training- sowie dem Validation-Set.

Im anschließenden Test 15 kaum eine größere weitere Verbesserung erkannt werden an den Konvergenz-Kurven. Insbesondere deuten die enormen Schwankungen darauf hin, dass das Training starker durch Random-/Zufalls-faktoren beeinflusst wurden und das Training eher auf Try-And-Error Aufbaut als einer strukturellen Verbesserung zu folgen, dies entspricht stark dem Charakter von neuronalen Netzen im Vergleich zu linearen Klassifizier Modellen. Nichts desto trotz konnte in Periode 10 eine Verbesserung des Genauigkeits-Wertes von einem absoluten Wert 0.92 gemessen werden. Der AUC-Wert ist auf 0.98 gestiegen.

Im direkten Vergleich zu dem vorherigen Test mit dem neuronalen Netzwerk Modell (mit identischen Trainingsparametern) konnte keine Verbesserung erkannt werden

durch die Nutzung der synthetisch generierten Richtungskosinus.
Viel entscheidender war in dieser Versuchsreihe die Doppelte Trainingsdauer, die zu dem in der Forschungsarbeit besten Wert von 0.92 führte, es ist jedoch zu beachten, dass insgesamt knapp 17 Stunden Training erforderlich waren, dieses Resultat zu erreichen.
Da die 3 weiteren Felder im Feature Vektor anhand der Ergebnisse keinen weiteren positiven Einfluss auf das Modelltraining hatten, ist Anzunehmen, dass das Modell in Testdurchlauf 12 ein ähnliches finales Ergebnis bei doppeltem Durchlauf liefern wurde bei geschätzten 13 Stunden Trainingszeit. Die Zeitdifferenz ist an dieser Stelle auf die unterschiedliche Feature Vektor Komplexität und damit verbundene Speichergröße zurück zu führen.

6.

Geschwindigkeit, Genauigkeit, Auswertung

Dieser Abschnitt beschreibt und vergleicht die Genauigkeit und Geschwindigkeiten der Modelle beziehungsweise der Testreihen in Abschnitt 5.

Von den 15 vorgestellten[9] Testreihen der Modelle konnte Modell Nummer 15 in Abschnitt 5.3 am besten abschneiden. Ein finales Genauigkeits-Ergebnis von 0.92 bezüglich der richtig zugeordneten positiven Kategorien konnte jedoch lediglich durch einen Zeitaufwand von 20 Stunden Trainingszeit erreicht werden.

Es ist jedoch festzustellen, dass lineare Klassifizierungsmodelle bereits eine gute vergleichbare Genauigkeit bei einer geringeren Trainingszeit erreichen (vgl. Abschnitt 5.2), z.B. konnten Test 8 und 9 Genauigkeitswerte zwischen 0.8 und 0.9 in unter 5 Stunden erreichen.

Während der Testreihe konnte erkannt werden, das insbesondere die Nutzung des AdamOptimizers als Optimierungsalgorithmus zu einer erhebliche Verbesserung des Trainings führt.

Viele Optimierungsalgorithmen sind darauf ausgelegt z.B. User-Inputs, Interessen oder auch Texte zu analysieren. Dies sind alles nicht deterministische Werte und daher nicht immer eindeutig bestimmbar. Bei diesen Daten kommt es aufgrund dieser Eigenschaft zu einer gehäuften Unregelmäßigkeit (auch Bezeichnet als engl. Noise). Viele Optimierungsalgorithm sind darauf ausgelegt, gerade mit solchen Daten zu arbeiten und dies beim Modelltraining zu beachten. Dies kann auf der anderen Seite unter Umständen Probleme geben, bei genauen (deterministischen) Daten wie denen die hier vorliegen. Zum Beispiel konnte der generelle Gradient Descent Algorithmus bei vorherigen Versuchen auf anderen Daten Sätzen die, die beschriebenen nicht deterministischen Werte enthalten, gute Ergebnisse liefern. Dieser erwies sich jedoch

[9] Es wurden im Rahmen dieser Arbeit weitere Trainings (insgesamt fast 30) durchgeführt, jedoch wurden lediglich 15 Modell vorgestellt, die ein repräsentatives und interpretierbares Ergebnis lieferten.

für diese Forschungsreihe als nicht optimal.
Insbesondere für neuronale Netzwerke ist der Adam Optimizer ein oft verwendeter Algorithmus, der sich aktuell durchgesetzt[10] hat.

Neben dem Optimierungsalgorithmus ist zu beachten, dass das Training mit einer großen Anzahl an Schnitttests als Input durchgeführt wurde. In fast allen Durchläufen wurden mehr als 800,000 Feature-Vektoren für das Training generiert und genutzt.
Dies ist eine sehr große Anzahl an Schnitttests für die, in Abschnitt 4.1 beschriebene und verwendete, sehr simplen Szene.
In der Regel handelt es sich jedoch um komplexere Szenen, die beim Raytacing beachtet werden und da wird eine große Anzahl an Schnitttests benötigt um die Szene akkurat zu beschreiben.
Das Modell erlernt durch das Training mithilfe der Samples die Geometrie der Szene und soll für ähnlichen Strahlen dieses Wissens nutzen, um das Auftreten eines Schnittes mit der Szenen Geometrie und dem Strahl vorher zu sagen.
Bei den verwendeten Strahlen kann an der Verteilung der Strahlen (vgl. Abschnitt 4.3.1 und 4.3.2) festgestellt werden, dass viele Strahlen sich sehr ähneln. Bei einer genaueren Analyse einzelner Samples aus dem Datensatz kann erkannt werden, dass sich die Daten an vielen Stellen lediglich am Nachkommaanteil unterscheiden und selbst dort erst bei der x-ten Stelle (vgl. Tabelle 20).

Tabelle 20 Ausschnitt Schnittest Datensatz (Exportiert aus der PBRT Killeroo-Szene [1])

Origin.x	Origin.y	Origin.z	Direction.x	Direction.y	Direction.z	hit
396.734711	54.7862091	30	-0.942072153	0.33537069	0.00517117977	1
-399.999817	338.417938	34.3734436	0.958314896	1.24889672e-08	-0.285714269	1
-399.999817	338.417938	34.3734436	0.745356023	-0.302660376	0.594004273	0
396.734711	54.7862091	30	-0.94228375	0.334784895	0.00452622771	1
-399.999817	337.858826	33.827179	0.218455017	0.922786713	0.317398936	0
-399.999817	337.858826	33.827179	0.885163188	-0.375686049	0.274492413	0
396.734711	54.7862091	30	-0.94217515	0.33508727	0.00475978851	1

[10] https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

Generell wird in solchen Fällen beim maschinellen Lernen eine Skalierung vorgenommen. Diese wird aber in diesem Fall als nicht sinnvoll angesehen wie in Abschnitt 4.2 erläutert. Zusammengefasst wird beim Raytracing eine enorme Genauigkeit für die Schnitttests benötigt, sodass eine Skalierung evtl. diese Genauigkeit zerstören könnte.
In der Regel wird erwartet, dass ein Modell besser auf die Trainingsdaten angepasst ist als auf Validierungs- oder Testdaten. Die Trainingsergebnisse dieser Versuchsreihen zeigten jedoch, dass der Unterschied zwischen diesen Daten-Samples kaum erkennbar ist.
Aufgrund der beschrieben Menge an Strahlen mit starker Ähnlichkeit und der Verwendung eine sehr große Menge an Trainingssamples, führte dies in den Testversuchen zu dem Phänomen, dass die Trainings-, Validierungs- und sogar Test-Evaluierungen sehr ähnlich und zum Teil identisch waren. Dies kann an den Konvergenz-Kurven und Genauigkeits-Werten in Abschnitt 5 erkannt werden.
Es ist anzunehmen, dass bei komplexeren Szenen und einer geringeren Anzahl an Trainingsschnittesten dieser Anteil stärker divergiert.

Ein weiterer Punkt der bei den Ergebnissen zu beachten ist, ist dass ein hoher Genauigkeitswert generell ein positives Zeichen ist, jedoch lediglich eine Aussage über die richtig zugeordneten Positiven Kategorien macht.
Wie Tabelle 21 zeigt, kann eine Ausgabe des Modells zu 4 verschiedenen möglichen Resultaten führen.

Tabelle 21 Raytracing Modell Confusion Matrix

True Positive (TP): Ein Strahl, der die Szenen Geometrie trifft, wird als ein solcher prognostiziert.	**False Positive (FP):** Ein Strahl, der die Szenen Geometrie nicht trifft, wird als fälschlicherweise als ein Strahl prognostiziert, der die Szene schneidet.
False Negative (FN): Ein Strahl, der die Szenen Geometrie trifft, wird fälschlicherweise als ein Strahl prognostiziert, der die Szene nicht schneidet.	**True Negative (TN):** Ein Strahl, der die Szenen Geometrie nicht trifft, wird als ein solcher prognostiziert.

Die beiden, in grün gekennzeichneten Möglichkeiten, sind das optimale Ergebnis.
Bei einem Genauigkeitswert von 1.0 liefert das Modell immer die richtigen Zuordnungen und dementsprechend eines der beiden grünen gekennzeichneten Ausgänge.
In der Regel besitzen Modelle die mittels maschinellem Lernen trainiert wurden meist keinen genauen Wert von 1.0 und auch die, in dieser Forschungsarbeit trainierten Modelle haben eine gewisse Fehlerqoute.
In Bezug auf die Strahlen des Raytracing ist anzunehmen, dass diese Fehlerqoute hauptsächlich, von den Randfällen wie zum Beispiel geometrischen Kannten stammt. Bei diesen ist auch beim regulären Raytracing eine sehr hohe Präzision erforderlich.
Um die beste Zuordnung zu erreichen, müssen in diesem Fall auch die falschen Zuordnungen beachtet werden. Je nach Anwendungszweck muss der Threshold so gewählt werden, dass die falschen Zuordnungen entweder ausgewogen sind oder so, dass die Kategorie mit einem gravierenderem negativ Effekt minimiert wird.
Um dies bewerten zu können, werden ROC-Kurven verwendet, wie die, die in Abschnitt 5 geplottet und analysiert sind.
Anhand dieser ROC-Kurven kann erkannt werden, dass fast alle im Rahmen dieser Arbeit erstellen Modell eine stärkere Tendenz besitzen, die positive Kategorie richtig bzw. mit einer höheren Sicherheit zu prognostizieren. Bei der negativen Kategorie besitzen die Modelle eine höhere Varianz und dementsprechend eine größere Unsicherheit bei der Zuordnung. Dies kann daran erkannt werden, dass die Kurve sehr kontinuierlich Abfällt auf der X-Achse (es wird für die negative Kategorie das Komplement betrachtet). Im Gegensatz besitzen die meisten Plots auf der Y-Achse eine konstante Steigung auf der Y-Achse (positive Achse) bis zu einem bestimmten Wert, ab dem die Kurve sehr schnell abfällt (in Betrachtung der Y-Achse).
Dies führt dazu, dass das Modell eine gute Arbeit macht, die positiven Ergebnisse zu finden und Kategorisieren, das Modell kann jedoch etwas schlechter die Negativen Ergebnisse zu klassifizieren.
Dies muss in möglichen Anwendungszwecken beachtet werden.

7.

Praktische Anwendungen & Ergebnisbewertung

Dieser Abschnitt diskutiert die praktische Anwendbarkeit der aus dieser Arbeit resultierenden Erkenntnisse.

Die Modelle dieser Forschungsarbeit sind darauf trainiert für einen gegeben Strahl[11] zu prognostizieren, ob dieser die Szenen Geometrie trifft oder nicht.

Es wird ein Genauigkeitswert zurückgeliefert und mit einem Schwell-Wert (engl. Threshold) wird dieser dann in einen bool‘schen Wert konvertiert.

Dabei sagt der Wert True aus, dass die Szenengeometrie getroffen wurde. Der Wert False gibt im Gegenteil dazu an, dass der Strahl die Szenengeometrie nicht trifft.

In Abschnitt 2.1 wurde die generelle Struktur des Raytrace Algorithmus beschrieben und auf die Szenentraversierung eingegangen.

Durch die in Abschnitt 2.3 beschriebenen Hierarchien der Szene kann diese Traversierung beschleunigt werden auf eine logarithmische Zeitkomplexität.

Das Ziel eines Modells ist es daher, diese Dauer bzw. Komplexität zu minimieren und sollte (abgesehen vom Training) eine Konstante Zeit besitzen.

In Betrachtung dieses Kriteriums liefert Modell 15 zwar einen Genauigkeitswert von ca. 91%, jedoch kann an dem Plot der ROC-Kurve erkannt werden, dass hauptsächlich die positiven Schnittest richtig klassifiziert werden, bei dem negativen

[11] Wie in Abschnitt 2.1 beschrieben besteht ein Strahl aus Ursprung und Richtung.

Schnitttest, hat das Modell jedoch Problem, was zu einer höheren Anzahl an False Negative Klassifizierungen führt.

<TODO>

Insbesondere, da bisher eine „große Menge an Forschung in Beschleunigungsstrukturen geflossen ist“ (Kapitel 4.2 [1]).

8.

Weiterführende Arbeit

<TODO>

Literaturverzeichnis

[1] M. Pharr, W. Jakob und G. Humphreys, Physically Based Rendering: From Theory To Implementation, Morgan Kaufmann, 2016.

[2] Frauenhofer-Institut, „MASCHINELLES LERNEN - EINE ANALYSE ZU KOMPETENZEN, FORSCHUNG UND ANWENDUNG," 2018.

[3] A. S. Kaplanyan, C. R. A. Chaitanya, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai und T. Aila, „Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder," [Online]. Available: https://research.nvidia.com/sites/default/files/publications/dnn_denoise_author.pdf. [Zugriff am April 2019].

[4] M. v. Übel, „All3DP 25 Best 3D Rendering Software Tools in 2019," April 2019. [Online]. Available: https://all3dp.com/1/best-3d-rendering-software/.

[5] Wikipedia, „Raytracing," [Online]. Available: https://de.wikipedia.org/wiki/Raytracing. [Zugriff am Juli 2019].

[6] H. Pritchett und R. Tamstorf, „Disney Moana Island Scene Read-Me," Disney, http://datasets.disneyanimation.com/moanaislandscene/island-README-v1.1.pdf, 2016.

[7] I. G. a. Y. B. a. A. Courville, Deep Learning, MIT Press, 2016.

[8] Google Developers, „Machine Learning Crash Course," Google, 2019. [Online]. Available: https://developers.google.com/machine-learning/crash-course/. [Zugriff am July 2019].

[9] Wackerly, Mendenhall und Scheaffer, Mathematical Statistics with Applications 7th Edition, Thomson Brooks/Cole, 2008.

[10] E. W. Weisstein, „"Direction Cosine." From MathWorld,“ A Wolfram Web Resource., [Online]. Available: http://mathworld.wolfram.com/DirectionCosine.html. [Zugriff am 07 2019].

[11] D. P. Kingma and J. Ba, "Adam: A Method for stochastik Optimization," ICLR, 2015.

[12] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos und J. Kubica, „Ad Click Prediction: a View from the Trenches,“ Google.

[13] A. S. Glassner, An Introduction to Ray Tracing, Morgan Kaufmann, 1989.

[14] M. De Berg, O. Cheong, M. van Kreveld und M. Overmars, Computational Geometry - Algorithms and Applications, 3rd Edition, Springer-Verlag Berlin Heidelberg, 2008.

[15] „IEEE Standard for Binary Floating-Point Arithmetic," in ANSI/IEEE Std 754-1985,“ IEEE, 12 Oct. 1985.

[16] P. Shirley, Ray Tracing in One Weekend, 2016.

[17] E. Haines, P. Hanrahan, R. L. Cook, J. Arvo, D. Kirk and P. S. Heckbert, An Introduction to Ray Tracing (The Morgan Kaufmann Series in Computer Graphics), Academic Press; 1st edition, (February 11, 1989).

Eidesstaatliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Ort, Datum	Unterschrift (Vor- und Nachname)