
Masterarbeit

Ableitung von Bewegungsmodellen für Anwendungen in der
Schüttgutsortierung mittels Machine Learning

Tobias Hornberger

31. Dezember 2018

Referent: Prof. Dr.-Ing. Uwe D. Hanebeck

Betreuer: Dipl.-Inform. Florian Pfaff

Zusammenfassung

This work was a cooperation with the Fraunhofer IOSB and centered around their TableSort bulk material sorting system. The aim was to provide an alternative to the labour-intensive task of fine tuning motion models for particle tracking by hand. This was achieved by using neural networks, which were implemented using the Tensorflow framework.

Eidesstattliche Erklärung

Hiermit erkläre ich, die vorliegende Masterarbeit selbstständig angefertigt zu haben.
Die verwendeten Quellen sind im Text gekennzeichnet und im Literaturverzeichnis aufgeführt.

Karlsruhe, 31. Dezember 2018

Tobias Hornberger

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Notation	V
1 Einleitung	1
1.1 Motivation	1
1.2 Aufbau der Arbeit	2
2 Grundlagen	3
2.1 Neuronale Netze	3
2.1.1 Perzeptron	3
2.1.2 Aktivierungsfunktionen	4
2.1.3 Feedforward Netze	6
2.1.4 Backpropagation	7
2.1.5 Overfitting	7
2.1.6 Regularisierung	8
2.2 TableSort System	10
2.3 Stand der Technik	14
3 Datenverarbeitung	15
3.1 Datenformatierung	15
3.2 Eigene Aufnahmen	16
3.2.1 Versuchsaufbau	16
3.2.2 Schüttgüter-Typen	16
3.3 Datenpipeline	17
3.4 Simulierte Daten	21
3.4.1 Menge	21
3.5 Daten Postprocessing	21
3.6 Trainingsbeispiele	23

4 Umsetzung und Implementierung	27
4.1 Software	27
4.2 Code Struktur	27
4.3 Hyperparameter	29
4.3.1 Hyperparameter Tuning	31
4.3.2 Architektur des neuronalen Netzes	32
5 Evaluation	35
5.1 System	35
5.2 Vergleichsmodelle	35
5.2.1 Constant Velocity Modell	35
5.2.2 Contant Acceleration Modell	36
5.2.3 Bias-Corrected Constant Velocity Modell	37
5.2.4 Average Acceleration Modell	37
5.2.5 Identical Acceleration Modell	37
5.3 Next Step	37
5.4 Separator	38
6 Fazit und Ausblick	39
Literatur	41
A Anhang	45

Abbildungsverzeichnis

2.1 Einzelnes Neuron	4
2.2 Plot der Ausgabe einer ReLU	6
2.3 Visualisierung von unterschiedlichen Kapazitäten	9
2.4 Beispiel Dropout-Regularisierung[JMLR:v15:srivastava14a]	11
2.5 TableSort Schüttgutsortiersystem [TODO Quelle]	12
2.6 Schematische Darstellung des optischen Bandsortierers TableSort nach [Pfa+17].	13
3.1 Zur Aufnahme verwendete Kamera [TODO: Quelle Bild]	16
3.2 Verschiedene Schüttgüter auf dem Förderband	17
3.3 Bayer Matrix [TODO: Quelle]	18
3.4 Verschiedene gesammelte Schüttgüter	19
3.5 Visualisierung der DEM Simulation [TODO: QUELLE!]	21
3.6 Verteilung Schüttgut Elemente nach Sorte	22
3.7 Visualisierung Data Augmentation durch Spiegelung	24
3.8 Geometrische Bestimmung der Labels [Vorlage aus Florians Diss - wie QUELLE?]	25
4.1 Skizze Codestruktur	28
4.2 Architektur Neuronales Netz für die NextStep Prädiktion	32
4.3 Architektur Neuronales Netz für die Separator Prädiktion	33
5.1 Evaluation für die NextStep Prädiktion	38
5.2 Evaluation für die Separator Prädiktion	38

Notation

Konventionen

- x Skalar
- \boldsymbol{x} Zufallsvariable
- \hat{x} Erwartungswert der Zufallsvariable \boldsymbol{x} .
- \underline{x} Spaltenvektor
- $\underline{\boldsymbol{x}}$ Zufallsvektor
- $\hat{\underline{x}}$ Erwartungswert des Zufallsvektors $\underline{\boldsymbol{x}}$.
- \mathbf{A} Matrix
- $(\cdot)_k$ Quantität zum Zeitpunkt k .
- \mathbb{R} Menge der reellen Zahlen.
- \sim Verteilungsoperator.
 - Z.B. bedeutet $\boldsymbol{x} \sim \mathcal{U}$, dass \boldsymbol{x} gemäß der Verteilung \mathcal{U} verteilt ist.
- Ende eines Beispiels.
- Ende eines Beweises.

Abkürzungen

- KF Kalman Filter
- LRKF Linear Regression Kalman Filter
- RMSE Root Mean Square Error

KAPITEL 1

Einleitung

- Maschinelle Lernverfahren sind ein heißes Thema, dass bei vielen existierenden Problemen Anwendung findet
- Schüttgutsortierung ist ein wichtiges Thema
- Anwendungsbereiche Schüttgutsortierung

hier vielleicht dieses 10% der Energie Beispiel bringen?

Einleitungstext

1.1 Motivation

- State of the Art: große Sortierer
- Kooperation ISAS IOSB, *TrackSort* Projekt
- Flächenkamera
- 2 geteiltes Problem: Tracking und Prediction
- Fokus dieser Arbeit: Prediction
- Bewegungsmodelle für verschiedene Schüttgüter von Hand finetunen ist viel Aufwand und schwer
- Option: Neuronale Netze einsetzen!
- zwei verschiedene Problemstellungen:
 - 1. die Position des Teilchens im nächsten Zeitschritt. **NextStep**
 - 2. die Position (und die Zeit) die das Teilchen beim Passieren des Düsenarrays haben wird. **Separator**

arbeit motivieren. Schüttgutsortieren ist ein interessantes Feld, das sich

1.2 Aufbau der Arbeit

Das schreibe ich auf, wenn die Gliederung finalisiert ist.

aufbau Gliede-
rung beschrei-
ben. Ganz am
Ende dann,
wenn sich
nichts mehr
ändert

KAPITEL 2

Grundlagen

In diesem Kapitel soll eine kurze Einführung in die für das Verständnis der restlichen Arbeit benötigten Themengebiete gegeben werden. Primär sollen zunächst allgemein neuronale Netze und einige ihrer speziellere Aspekte betrachtet werden bevor ein kurzer Blick auf das bei den Experimenten verwendete Schüttgutsortiersystem *TableSort* geworfen wird.

2.1 Neuronale Netze

Als Neuronale Netze bezeichnet man in der Informatik Systeme aus künstlichen Neuronen, die heute eine wichtige Rolle im Feld des maschinellen Lernen einnehmen. Manchmal werden sie korrekter als *künstliche neuronale Netze* bezeichnet um sie von *natürlichen neuronalen Netzen* wie dem menschlichen Gehirn zu unterscheiden, nach deren biologischem Vorbild sie modelliert sind.

Die Grundsteine des Feldes wurde bereits 1943 von Warren McCulloch und Walter Pitts gelegt, die in ihrem Paper [MP43] ein Neuronenmodell vorschlugen, mit dem sich logische arithmetische Funktionen berechnen lassen. Nach einer Periode von relativ geringer Aufmerksamkeit der wissenschaftlichen Gemeinschaft während den 1970ern und folgenden Jahrzehnten haben einige bahnbrechende Ergebnisse um das Jahr 2010, unter anderem im Feld der Spracherkennung, das Interesse an dem Feld wieder entfacht.

2.1.1 Perzepron

Die kleinste Einheit eines neuronalen Netzes ist das Perzepron, wie es 1958 von Frank Rosenblatt beschrieben wurde [Ros58]. Es ist eine Art künstliches Neuron, dass eine Reihe an Eingaben entgegen nimmt und einen einzelnen Wert o ausgibt.

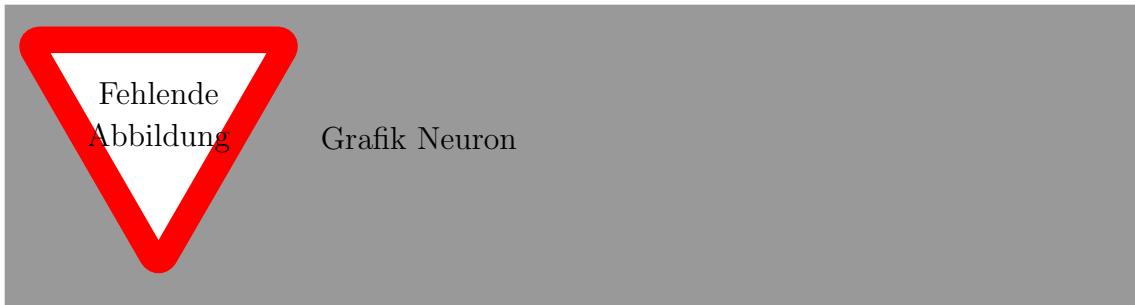


Abbildung 2.1: Einzelnes Neuron

Darstellung
Perzepron

Die einzelnen Eingaben x_i haben jeweils eine Gewichtung w_i . Es existiert ein sogenannter Schwellwert oder *bias*, der normalerweise durch eine zusätzliche Eingabe x_{m+1} mit dem Wert +1 und dem dazugehörigen Gewicht w_{m+1} modelliert wird. Den Ausgabewert y erhält man dadurch, dass man die gewichteten Eingaben aufsummiert und in die Aktivierungsfunktion ϕ des Perzepron giebt. Ein Überblick über verschiedene Aktivierungsfunktionen ist unter 2.1.2 zu finden.

Mathematisch ist die Ausgabe eines Perzepron also wie folgt definiert:

$$y = \phi\left(\sum_{i=0}^m w_i x_i\right)$$

Beim Lernen werden die Gewichte der einzelnen Eingaben so angepasst, dass die gewünschte Ausgabe erreicht wird. Ein einzelnes Perzepron mit zwei Eingängen kann zur Darstellung der logischen Operatoren AND, OR und NOT genutzt werden

Letztendlich ist ein solches Perzepron jedoch nur ein linearer Klassifikator und kann somit zum Beispiel den XOR Operator nicht auflösen. Dies zeigten Marvin Minksy und Seymour Papert 1969 in einflussreichen Buch *Perceptrons: an introduction to computational geometry*

quelle	Linear Trennbares / Nicht-linear Trennbares Problem (AND vs XOR)
	Solche, nicht linear-separierbare Probleme zu lösen müssen mehrere Schichten an Neuronen kombiniert werden.

2.1.2 Aktivierungsfunktionen

Entscheiden, ob diese Section weg soll, oder auf nur ReLU reduziert werden sollte

Es gibt verschiedene Aktivierungsfunktionen, die für den Einsatz in neuronalen Netzen in Frage kommen. Sie sind von essenzieller Wichtigkeit, da ohne eine Nicht-Linearität das Netz in eine einfache Regression kollabiert.

Eine Aktivierungsfunktion sollte leicht abzuleiten sein, da dies im Rahmen des Trainings mit dem Backpropagation Algorithmus häufig geschieht und sonst beträchtlicher Rechenaufwand entsteht.

In der Vergangenheit wurden einige verschiedene Aktivierungsfunktionen verwendet. Jede dieser Funktionen stellt eine Nicht-Linearität dar und nimmt eine einzelne Zahl, wendet eine bestimmte, festgelegte mathematische Operation auf diese an und gibt das Ergebnis zurück. Historisch ist häufig die Sigmoid-Funktion verwendet worden, da sie das Verhalten eines natürlichen Neurons gut nachbildet.

$$f(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^{x+1}}$$

In der Praxis jedoch haben sich einige Nachteile der Sigmoid-Funktion gezeigt. Einige dieser Probleme konnten mit der Verwendung des Tangens hyperbolicus behoben werden, durchgesetzt haben sich jedoch in letzter Zeit sogenannte *Rectified Linear Units*, oder kurz ReLUs.

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 0 & , \text{ falls } x < 0 \\ 1 & , \text{ falls } x > 0 \end{cases}$$

Abbildung 2.2 zeigt den Plot einer solchen ReLU. Die Aktivierung von ReLUs ist ein einfacher Schwellwert, der weit weniger rechenintensiv ist, als die aufwendigen Exponentialfunktionen von Sigmoid und tanh. In der Praxis hat sich gezeigt zudem gezeigt, dass ReLUs deutlich schneller konvergieren als Sigmoid- oder tanh-Neuronen. In der Praxis hat sich gezeigt zudem gezeigt, dass ReLUs deutlich schneller konvergieren als Sigmoid- oder tanh-Neuronen. In der Praxis hat sich gezeigt zudem gezeigt, dass ReLUs deutlich schneller konvergieren als Sigmoid- oder tanh-Neuronen. Krizhevsky et al. haben in ihrem Paper[KSH12] einen Geschwindigkeitsgewinn um Faktor 6 feststellen können. Ein Problem, das mit ReLUs jedoch existiert ist, dass einzelne Neuronen während dem Training „absterben“ können. Diese Neuronen sind dann für jeden beliebigen Input inaktiv und können niemals wieder etwas zur Ausgabe des Netzes beitragen. Durch die Wahl einer geeigneten Lernrate oder den Einsatz sogenannter Leaky ReLUs lässt sich dies jedoch vermeiden. Leaky ReLUs haben im

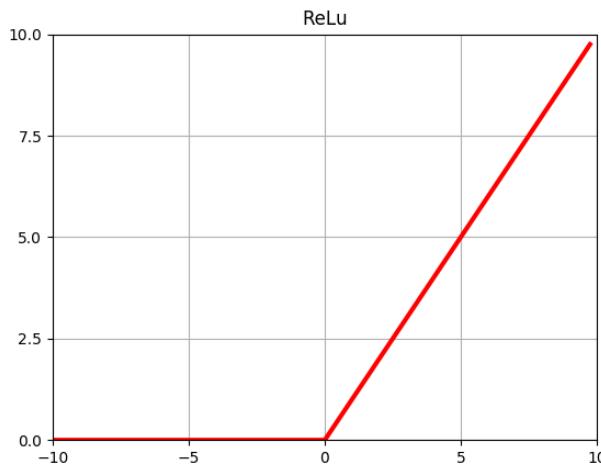


Abbildung 2.2: Plot der Ausgabe einer ReLU

Gegensatz zu normalen ReLUs eine kleine positive Steigung im negativen Bereich.

$$f(x) = \begin{cases} x & , \text{ falls } x > 0 \\ 0.01x & , \text{ falls } x \leq 0 \end{cases}$$

2.1.3 Feedforward Netze

- Definition (keine Kreise oder Schleifen)
- Gegenstück zum RNN
- grundlegende Architektur in Layern
- Aktivierungsfunktionen in Layern
- Outputlayer: Verschiedene Aktivierungsfunktionen:
- Linear für regression, z.B. Softmax für Wahrscheinlichkeitsverteilung Softmax
- (One Hot encoding?)

Als Feedforward Netz bezeichnet man ein neuronales Netz, zwischen dessen Knoten keine Kreise oder Schleifen existieren. Das entsprechende Gegenstück bezeichnet man als *Rekurrentes Neuronales Netz*. Die Informationen wandert in der Verarbeitungsrichtung von den Eingabeknoten zu den Ausgabeknoten. Für gewöhnlich sind die einzelnen Knoten in Schichten, sogenannten Layern, organisiert. Die Neuronen eines einzelnen Layers sind meist die Eingabe wird in ein Input Layer eingeben.

2.1.4 Backpropagation

Entscheiden ob die Subsection weg sollte

Der Backpropagation Algorithmus ist ein Verfahren mit denen künstliche neuronale Netze in der Lage sind komplizierte Zielfunktionen einzulernen. Es ist eine Methode, bei der effizient der Gradient der Fehlerfunktion in Abhängigkeit vom Gewicht der einzelnen Kanten im Netz bestimmt werden kann, was dann für einen Gradientenabstieg verwendet werden kann.

- Definition und Beschreibung
- Nur supervised learning: Gradient der Fehlerfunktion wird benötigt → Tatsächliches Ergebnis muss bekannt sein.
- „Finden einer Funktion, die am besten die Inputs auf die outputs mapt“

2.1.5 Overfitting

Das essenzielle Maß nachdenken neuronale Netze bewertet werden, ist wie gut sie mit neuen, unbekannten Daten umgehen, die nicht in den Trainingsdaten vorhanden waren. Diese Eigenschaft von Trainingsdaten auf unabhängige Testdaten zu schließen wird Generalisierung genannt.

Als Overfitting bezeichnet man es, wenn ein Modell sich zu sehr an ein gegebenes Datenset anpasst und dafür in Kauf nimmt zusätzliche oder zukünftige Daten schlechter zu repräsentieren. Das System wird also schlechter darin zu generalisieren.

Im Feld des überwachten Lernens beziehungsweise der neuronalen Netze ist Overfitting daran zu erkennen, dass die Qualität die Ausgaben des Netzes auf dem Trainingsdatenset sich weiter verbessert, während sie auf dem Testdatenset schlechter wird. Dies kann zum Beispiel der Fall sein, wenn das Modell Rauschen in den Testdaten als Teil der zugrundeliegenden Struktur interpretiert.

Dem Overfitting gegenüber steht das Underfitting. Als Underfitting bezeichnet man wenn das Netz nicht in der Lage ist eine ausreichend gute Performance auf den Trainingsdaten zu erreichen. Das kann passieren wenn das Modell nicht ausreichend komplex ist um die zugrundeliegende Struktur der Daten abzubilden.

Sowohl Overfitting als auch Underfitting hängen mit der Kapazität eines Netzes zusammen. Ist die Kapazität zu gering kann es sein, dass das Netz daran scheitert die Trainingsdaten zu lernen. Ist die Kapazität zu groß, so kann es passieren, dass das Netz, umgangssprachlich ausgedrückt, die Trainingsdaten einfach auswendig lernt.

Dies ist beispielhaft in Abbildung 2.3 zu sehen. Aus der zugrundeliegenden cos-Funktion werden Stichproben mit einem Rauschterm entnommen. Stellvertretend für das Lernen mit neuronalen Netzen wird hier lineare Regression benutzt, um die Parameter eines Polynoms zu lernen. Die Kapazität des Modells wird hier durch den Grad des Polynoms festgelegt.

In Abbildung 2.3a wird ein Polynom mit dem Grad 1 gelernt. Die Kapazität ist zu niedrig und dementsprechend schafft das Modell es nicht einmal die Stichproben akkurat zu repräsentieren.

In Abbildung 2.3b ist der Grad des Polynoms 4. Das resultierende Modell ist relativ dicht an der ursprünglichen Funktion.

In Abbildung 2.3c

- Methoden um Overfitting zu vermeiden:
- Mehr Trainingsdaten (z.B. durch Data-Augmentation)
- Regularisierung (L_1 , L_2 (siehe unten), dropout)

2.1.6 Regularisierung

- Regularisierung Definition
- Mathematische Formel Darstellung
- L_1 und L_2 Regularisierung
- L0 - warum nicht?
- (Maybe Dropout)
- Early Stopping? (das internet sagt, es ist eine art von Regularisierung und falls ich sie verwenden sollte, sollte ich sie hier erwähnen)

Als Regularisierung bezeichnet man eine Technik, die benutzt wird um ein Modell von Overfitting abzuhalten. Sie wird in der Hoffnung angewendet, dass das Modell mit Regularisierung besser generalisiert als ohne.

Eine Möglichkeit ist, dass zur Loss Funktion ein Regularisierungsterm R hinzugefügt wird, der die Kosten basierend auf der Komplexität des Systems erhöht.

$$\min_f \sum_{i=1}^m V(f(\underline{x}_i), \underline{y}_i) + \lambda R(f)$$

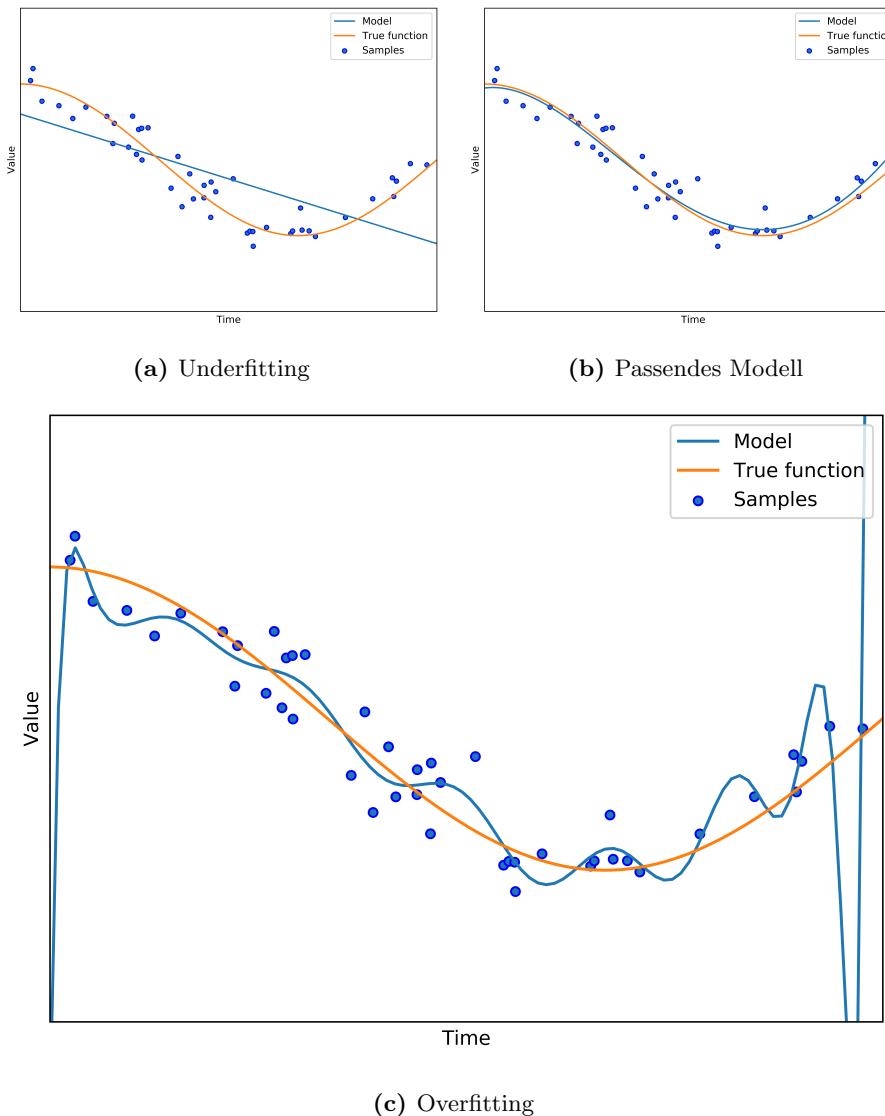


Abbildung 2.3: Visualisierung von unterschiedlichen Kapazitäten

Dabei ist V die Loss Funktion, beispielsweise *Mean-Square-Error* oder *Mean-Absolute-Error*. n ist die Anzahl der Feature-Label-Paare, x_i und y_i sind die einzelnen Eingabe-features und das dazugehörige Label. Die Funktion f ist in unserem Fall das neuronale Netz, das die Features entgegen nimmt. λ ist ein Parameter, der die Gewichtung des Regularisierungsterm festlegt. Wählt man diesen Parameter zu klein, so kann es sein, dass das Modell trotz Regularisierung noch immer overfittet. Wählt man ihn zu groß so kann es sein, dass das Modell das Problem nicht mehr korrekt abbildet und es zu Underfitting kommt. Der Regularisierungsterm R wird so gewählt, dass er die Komplexität der Funktion f wiederspiegelt. Ein gutes Maß für die Komplexität eines neuronalen Netzes sind die Gewichte zwischen den Neuronen. Beispiele für R wären zum Beispiel die L_1 - oder die L_2 -Regularisierung. Der entscheidende Unterschied

zwischen den beiden ist der unterschiedliche Strafterm, zu sehen in 2.1.6 für L_1 und 2.1.6 für L_2 . Die Fehlerfunktionen sind jeweils MSE mit dazugehörigen Strafterm.

$$J(X, Y) = \frac{1}{m} \sum_{i=1}^m (\underline{y}^{(i)} - \hat{\underline{y}}^{(i)})^2 + \sum_{j,k} (|\mathbf{W}_{j,k}|)$$

$$J(X, Y) = \frac{1}{m} \sum_{i=1}^m (\underline{y}^{(i)} - \hat{\underline{y}}^{(i)})^2 + \sum_{j,k} (\mathbf{W}_{j,k}^2)$$

Ein Regressionsmodell, dass L_1 -Regularisierung verwendet wird auch als Lasso-Regression bezeichnet, während ein Modell mit L_2 -Regularisierung als Ridge Regression beschrieben werden kann. Vergleicht man die beiden Ansätze, so schrumpft die L_1 Norm weniger wichtige Gewichte auf 0, was zu dünn besetzten Gewichtsvektoren führt. Dies kann eine wünschenswerte Eigenschaft sein. Im Gegensatz dazu hat die L_2 -Regularisierung, den Vorteil, dass sie effizienter berechnen kann. Der Strafterm von L_2 hat eine geschlossene Form und kann in Form einer Matrix angewendet werden, während die Funktion von L_1 auf Grund des Betrags eine nicht-differenzierbar ist.

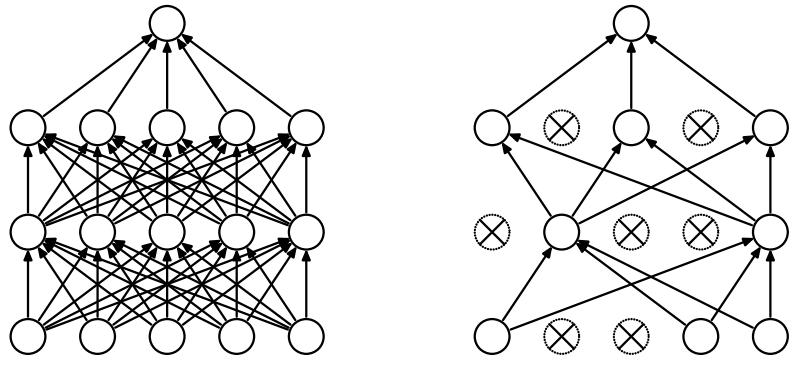
absatz zu (L0
und warum
man es nicht
benutzt?)

Eine weitere Regularisierungstechnik ist Dropout[JMLR:v15:srivastava14a]. Dabei werden während dem Training eines neuronalen Netzes mit einer festgelegten Wahrscheinlichkeit zufällig Neuronen und die dazugehörigen Verbindungen abgeschaltet, wie in Abbildung 2.4 dargestellt. Dies soll insofern Overfitting vermeiden, dass es übermäßige Kodaption von mehreren Neuronen erschwert. Dropout als Technik wird insbesondere bei tiefen neuronalen Netzen eingesetzt.

2.2 TableSort System

Viel stuff über das TableSort System

- kleiner, experimenteller Bandsortierer [Dol15]
- Entstanden in Kooperation zwischen dem Fraunhofer IOSB, Abteilung Sichtprüfsysteme, und dem Institut für Intelligente Sensor Aktor Systeme des Karlsruher Institut für Technologie.
- Im Rahmen des *TrackSort* Projekts
- Gedacht für Experimente, wenn es zu aufwendig ist das mit dem großen großen zu machen und zum Mitnehmen auf Messen.



(a) Unverändertes neuronales Netz

(b) Nach Dropout Anwendungen.

Abbildung 2.4: Beispiel Dropout-Regularisierung[JMLR:v15:srivastava14a]

- 2 Modi: mit Förderband und mit Rutsche
- Mit Flächenkamera für TrackSort als auch die Zeilenkamera sind dargestellt.
- Ringlicht (Refence später)
- Die Zeilenkamera wird zurzeit in industriellen Schüttgutsortieranlagen verwendet, ist aber nicht optimal (Siehe all die Literatur)

Das Table

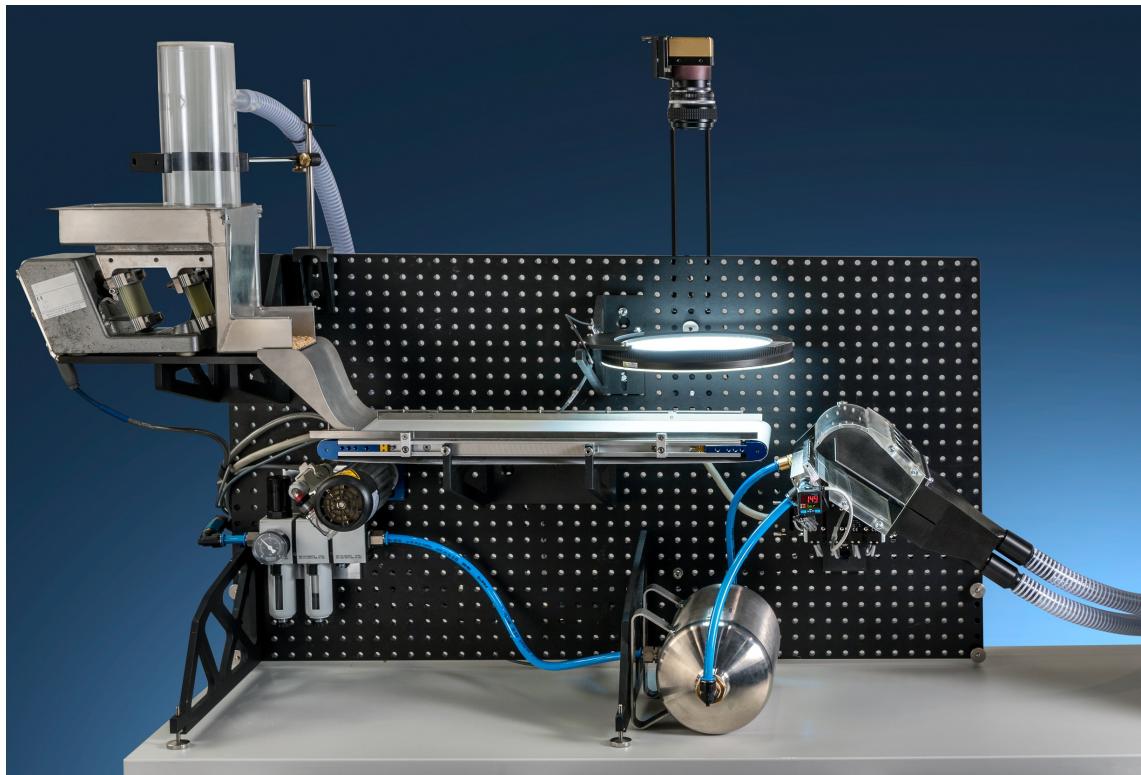


Abbildung 2.5: TableSort Schüttgutsortiersystem [TODO Quelle]

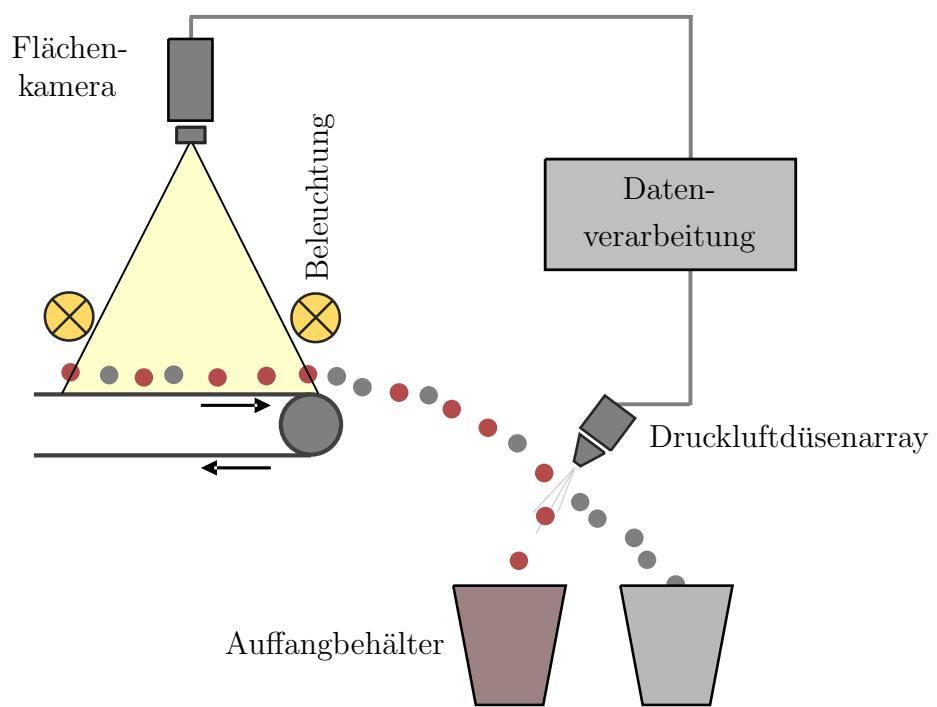


Abbildung 2.6: Schematische Darstellung des optischen Bandsortierers TableSort nach [Pfa+17].

2.3 Stand der Technik

Notizen:

Das aller aller dickste TODO

Wie wird das, was ich dann mit Neuronalen Netzen machen möchte aktuell gemacht?

Teilaspekt vom Tracksort Multi-Targettracking: Bewegungsprädiktion. Primär Florian's Dissertation: Kapitel 4. Einige

KAPITEL 3

Datenverarbeitung

Wie bei jeder Anwendung von maschinellen Lernverfahren sind die zugrundeliegenden Daten von äußerster Wichtigkeit. Im Rahmen dieser Masterarbeit wurden zweierlei Sorten von Daten benutzt: Einmal wurden am *TableSort* Schüttgutsortierer des Fraunhofer IOSBs Aufnahmen gemacht, die dann über mehrere Arbeitsschritte in das richtige Datenformat übersetzt wurden. Zudem existieren der DEM Datensatz

mehr details -
Quellen lesen

3.1 Datenformatierung

- Zu Beginn des Datenverarbeitungskapitel erstmal definieren wie unsere Feature-Label Paare aussehen.
- Features eigentlich immer gleich:
- die Positionen der letzten n Zeitschritte (FeatureSize Hyperparameter)
- also ein $2n$ Tupel, mit jeweils n X-Koordinaten und n Y-Koordinaten

Labels: Unterscheidung nach Anwendung:

NextStep: Label ist 2-Tupel, X und Y Koordinate Separator: gegeben ist eine Stelle entlang der Bewegungsrichtung der Teilchen an der der Separator angebracht ist. erstes element des Label ist die Koordinate entlang der orthogonalen Achse zur Bewegungsrichtung wo das Teilchen den Separator passiert zweites Element ist die Anzahl von Zeitschritten , die das Teilchen noch bis zum Separator braucht.

Important Point: Labels wurden normalisiert und Standardisiert ($\frac{\text{TrueVal} - \text{Mean}}{\text{StandardDeviation}}$) um auszugleichen, dass sich Position und Zeitschritte auf unterschiedlichen Skalen bewegen und dementsprechend unterschiedlich hohe gradienten haben.

Es ist implementiert, dass die verschiedenen Dimensionen unterschiedlich stark gewichtet werden können - Je nach Schüttgut/präzision des Separators Aber für die evaluierung ist keine Gewichtung vorgenommen worden.



Abbildung 3.1: Zur Aufnahme verwendete Kamera [TODO: Quelle Bild]

optional: Histogramme über die Daten (mehr Teilchen in der Mitte bei Location...)

3.2 Eigene Aufnahmen

3.2.1 Versuchsaufbau

- Am TableSort System, einmal Band, einmal Rutsche
- Beschreibung von der Bonito Kamera, stats usw.
- Umrechengröße pixel zu mm
- Bandgeschwindigkeit

Zur Aufnahme der Daten wurde eine Bonito CL-400 200 FPS Kamera benutzt, die in Abbildung 3.1 zu sehen ist. Die ist, wie in Abbildung 2.5 oberhalb des Förderbandes angebracht. Die Bilder, die von der Kamera aufgenommen werden, haben eine Auflösung von 2320x1726 Pixeln [All14].

Umrechengröße pixel zu mm, im weiteren Verlauf werden pixel benutzt

3.2.2 Schüttgüter-Typen

Aufgenommen wurden vier verschiedene Schüttgüter, die in Abbildung 3.4 zu sehen sind.

- Kugeln
- grüne Pfefferkörner
- Zylinder
- Weizenkörner

Die Kugeln und der Pfeffer sowie die Zylinder und die Weizenkörner bilden jeweils ein Paar aus einem geometrischen Körper und einem echten Objekt, das grob dessen Form ähnelt.

Die Kugeln bestehen aus Holz und haben einen Durchmesser von 5mm. Die Zylinder bestehen ebenfalls aus Holz. Sie haben eine Länge von 1cm und einen Durchmesser von 3mm. Die Schüttgüter sind in Abbildung 3.4 in Schüsseln und in Abbildung 3.2 auf dem Förderband zu sehen.

TODO: Details

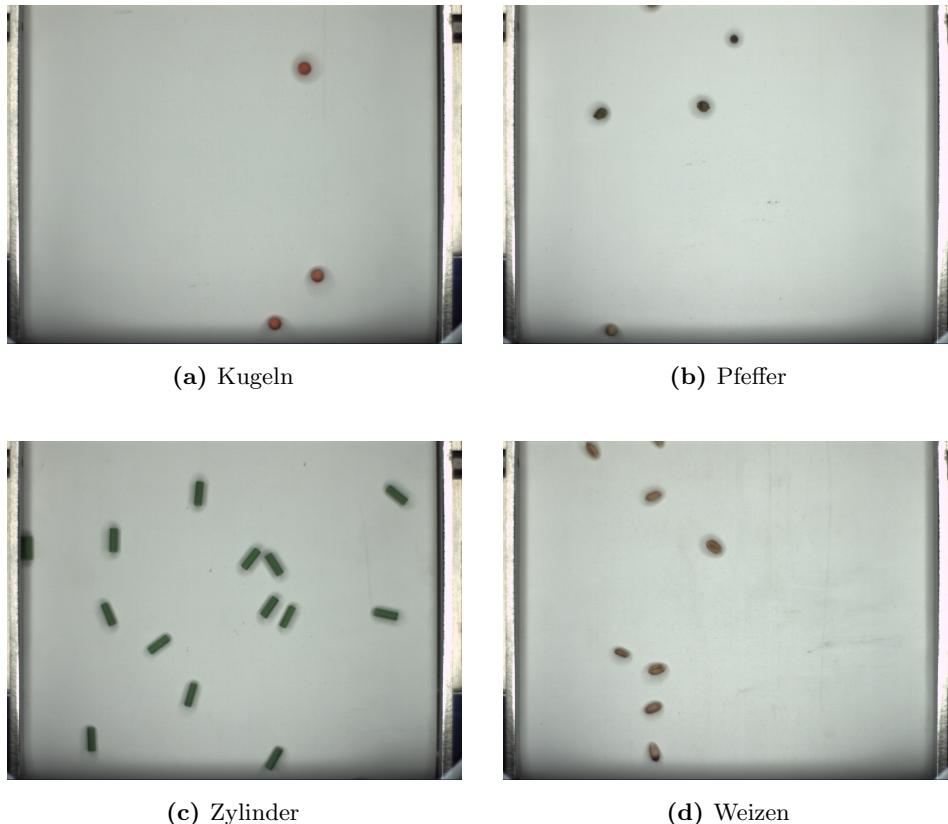


Abbildung 3.2: Verschiedene Schüttgüter auf dem Förderband

3.3 Datenpipeline

- Beschreiben wie aus den Bildern die relevanten Features extrahiert werden.

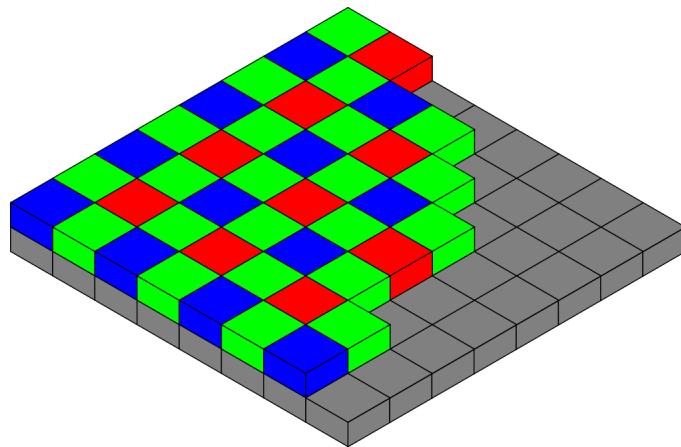


Abbildung 3.3: Bayer Matrix [TODO: Quelle]

- Ursprungs: Bayer Matrix Bitmap
- Konvert to RGB
- Segmentierungsskript zu CSV No.1
- TrackSort Algorithmus zuweisung zu CSV No.2
- Das ist dann der finale Punkt von wo es in meinen Code geladen wird und der rest dort passiert

Die Bonito Kamera nimmt Bilder in Form einer Bayer-Matrix auf, wie sie in 3.3 zu sehen ist. Diese werden dann in Batches von je 3500 gesammelt und in Bitmap Dateien geschrieben.

Auf Grund der Menge an Bildern wurden die Bilder zunächst in das png Dateiformat übertragen. Die Features, die für das Trainieren der Netze benutzt werden, sind die Koordinaten der Mittelpunkte der Objekte. Um diese zu bestimmten, müssen zunächst die Dateien mittels *demosaicing* rekonstruiert werden um gewöhnliche RGB Bilder zu erhalten. Die Open Source Computer Vision Library OpenCV hat eine Methode implementiert, die ein Bild von einem Farbraum in einen anderen übertragen kann. Diese wurde eingesetzt um die einzelnen Bilder in RGB Farbbilder zu konvertieren.

Skript ursprünglich von Georg, ein paar changes implementiert (bezüglich input und output.)

Auf diesen kann dann eine Segmentierung vorgenommen werden. Hierzu wurde erneut die Computer Vision Library OpenCV benutzt. Für jede Sorte von Schüttgut wurde ein eigenes Parameterprofil von Hand angepasst. Diese bestehen aus einem oberen und unteren Grenzwert in jedem Kanal des HSV-Raums und einer minimalen Fläche,

hier Bayer-
Matrix er-
klären und
Bild?

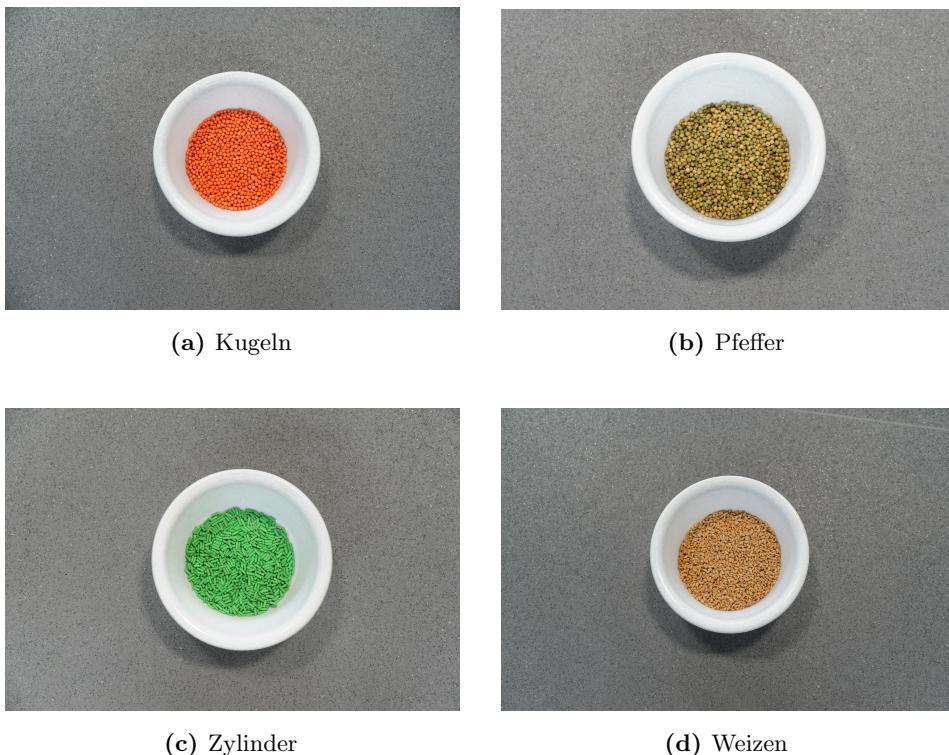


Abbildung 3.4: Verschiedene gesammelte Schüttgüter

die ein Teilchen umfassen muss. Entsprechend der durch das Profil festgelegten Parameter werden für die einzelnen Bilder Masken angelegt, ob die HSV-Werte der einzelnen Pixel innerhalb oder außerhalb der Grenzwerte liegen. Mit diesen Masken werden dann alle möglichen Konturen von Schüttgutpartikeln extrahiert, bevor diese noch einmal bezüglich ihrer Sinnhaftigkeit gefiltert werden. Im letzten Schritt wird nun der gewichtete Mittelpunkt der einzelnen Konturen bestimmt und abgespeichert. Das Ergebnis von diesem Segmentierungsscripts ist eine CSV Datei für jedes Batch. Ein Beispiel für einen Ausschnitt aus solch einer Datei ist in Tabelle 3.1 zu sehen. Eine Zeile repräsentiert jeweils ein Bild aus dem Batch, also einen Zeitschritt. Zu Beginn jeder Zeile steht zunächst die Frame Nummer, gefolgt von der Anzahl der detektierten Partikel und den X- und Y-Koordinaten der Mittelpunkte der detektierten Partikel.

Die Mittelpunkte in diesem CSV File werden nun mittels des in MATLAB implementierten TrackSort Algorithmus einzelnen Tracks zugeordnet, die dann wiederum in einem neuen CSV File gespeichert werden. Die einzelnen Tracks werden als Spaltenpaare dargestellt mit jeweils einer Spalte für die X- und Y-Koordinaten zu einem jeweiligen Zeitpunkt. Ein Ausschnitt aus einer solchen Datei ist in Tabelle 3.2 zu sehen.

Mehr details:
Tracksort track-
zuweisung As-
signment Pro-
blem. Referenz
Tobi MA?

Tabelle 3.1: Ausschnitt aus dem Ergebnis des Segmentierungsscripts

Frame	#MP	MP_1_x	MP_1_y	MP_2_x	MP_2_y	MP_3_x	MP_3_y
636	1	1222.9975	92.7641	NaN	NaN	NaN	NaN
637	1	1223.4063	182.9758	NaN	NaN	NaN	NaN
638	1	1223.6052	273.2425	NaN	NaN	NaN	NaN
639	1	1223.7067	364.0339	NaN	NaN	NaN	NaN
640	1	1224.0704	453.9057	NaN	NaN	NaN	NaN
641	2	1224.2051	544.5191	1692.4549	43.8822	NaN	NaN
642	2	1224.5793	634.7288	1696.6901	135.9595	NaN	NaN
643	2	1224.9082	726.0094	1700.451	229.1195	NaN	NaN
644	2	1225.2296	815.9663	1704.1472	321.2075	NaN	NaN
645	2	1225.4286	906.7078	1708.0593	414.2785	NaN	NaN
646	2	1225.7588	996.0286	1711.5309	506.0545	NaN	NaN
647	3	1226.0411	1086.5729	1714.8831	599.5417	961.8821	62.7111
648	3	1226.2337	1175.9271	1718.1401	691.6325	958.5526	154.3124
649	3	1226.2073	1265.7495	1721.6618	784.5927	955.3107	246.5241
650	3	1226.2543	1354.9362	1724.9158	876.7192	952.4919	338.1123
651	3	1226.2634	1444.5903	1728.3341	970.2909	949.2896	430.9692
652	3	1226.0845	1533.0901	1732.1745	1062.4624	946.3455	522.8667
653	3	1225.7319	1621.8461	1735.8759	1155.2937	943.3384	615.4545
654	2	1739.6714	1247.1867	940.2511	707.7306	NaN	NaN
655	2	1743.4279	1339.4146	937.2216	800.4557	NaN	NaN
656	2	1747.1525	1430.2501	934.5311	891.7249	NaN	NaN
657	2	1750.9771	1521.8102	931.6626	984.2284	NaN	NaN
658	2	1754.1491	1612.5565	928.7587	1076.4749	NaN	NaN
659	1	925.8463	1168.794	NaN	NaN	NaN	NaN
660	1	922.8752	1260.7461	NaN	NaN	NaN	NaN
661	1	920.2056	1352.8549	NaN	NaN	NaN	NaN
662	1	917.4051	1444.3431	NaN	NaN	NaN	NaN
663	1	914.6493	1535.5131	NaN	NaN	NaN	NaN
664	1	911.8565	1626.5341	NaN	NaN	NaN	NaN

Tabelle 3.2: Ausschnitt aus dem Ergebnis des *TrackSort* Algorithmus

TrackID_4_X	TrackID_4_Y	TrackID_5_X	TrackID_5_Y	TrackID_6_X	TrackID_6_Y
1036.4613	82.3719	1899.9239	83.2049	1654.4423	50.6811
1033.0189	174.9809	1896.8142	171.3283	1655.3193	143.9749
1029.6167	266.4979	1893.5937	259.8098	1656.0221	237.1573
1026.3908	358.4831	1890.3912	348.1731	1656.8966	329.8636
1023.0203	449.6429	1887.1035	436.4588	1657.6308	423.1592
1019.5391	542.2334	1883.7761	525.1073	NaN	NaN
NaN	NaN	1880.2716	613.0896	NaN	NaN
NaN	NaN	1876.6054	701.9719	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN



Abbildung 3.5: Visualisierung der DEM Simulation [TODO: QUELLE!]

3.4 Simulierte Daten

Die DEM Daten, wo sie herkommen, was der unterschied ist zu den selbstaufgenommenen Daten. Vorteile und Nachteile...[Pie+16], [Pie+17]

Original: 1000Hz, downsampled auf 200 Hz,

Nicht ganz so viele Partikel, aber dafür sehr lange tracks - Informationen auf dem gesamten Band, nicht nur auf dem Part wo die Kamera drauf schaut.

Vergleich bezüglich der Eignung für die verschiedenen Ansätze dann im Evaluations Kapitel

3.4.1 Menge

Insgesamt wurden 265451 Bilder aufgenommen. 177951 Bilder auf dem Förderband
87500 Bilder auf der Rutsche

Anzahl separate Tracks. Anmerkung: Simulationsdaten sind die echte Wahrheit, während die Werte von den Selbstgesammelten Daten hier auf die Korrektheit des Outputs vom Tracksort Algorithmus beruhen.

Es wurden 7538 Kugeln in 15 Batches, 7056 Pfefferkörner in 13 Batches, 17049 Zylinder in 11 Batches und 8549 Weizenkörner in 13 Batches aufgenommen.

das in eine Schöne Tabelle stecken und erzählen

Table mit Anzahl von Elementen in verschiedenen Batches

3.5 Daten Postprocessing

Cleanup : FilterTracksByAngle, FilterByVectorLengthChange

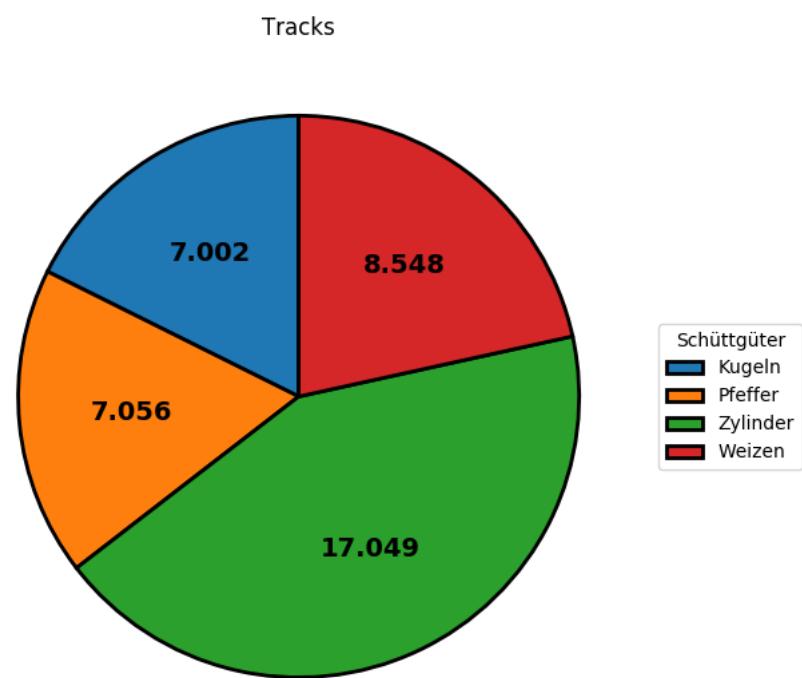


Abbildung 3.6: Verteilung Schüttgut Elemente nach Sorte

- Data Augmentation: Definition und Beschreibung
- bei Bildern normalerweise Rotieren, Translation, Ausschnitte...
- Hier: Spiegeln
- in einem Band - an der Mitte, nicht die Ränder mit nehmen - Kamera nicht perfekt zentriert
- führt zu: Beinah verdoppelung der Feature-Label-paare fürs training.

Als Data Augmentation bezeichnet man Verfahren, die die eigenen Daten erweitern ohne zusätzliche Daten aufzunehmen. Man generiert aus den bestehenden Daten zusätzliche, synthetische Daten, die dann im Trainingsset eingesetzt werden können. Ausreichend viele Trainingsbeispiele zu haben ist notwendig um mit neuronalen Netzen eine gute Performance zu erzielen. Die synthetischen Beispiele müssen jedoch plausibel sein, da sie sonst die Qualität der Ausgabe des Netzes negativ beeinträchtigen können.

Für Netze, die in der Bilderkennung eingesetzt werden gibt es einige weit verbreitete Techniken, zum Beispiel Rotation, Translation, Spiegeln und das Ausschneiden von Teilbildern.

Für den gegebenen Fall mit den Mittelpunkten von Schüttgut Partikeln als Features resultiert von diesen Techniken nur das Spiegeln in sinnvollen Daten. gespiegelt wird an der Mittellinie entlang der Bewegungsrichtung. Tracks, die [zu nah an den Rand kommen] werden ausgenommen, da zumindest bei den selbst aufgenommenen Daten, die Kamera nicht perfekt zentriert ist.

3.6 Trainingsbeispiele

Train - Test - Validation - Split: Train - test, 90% zu 10%. Validation nur für die sets auf denen das Hyperparameter Tuning gemacht wurde [ungefähres]

Features: NextStep einfach alle n -Tupel, die ein Track hergibt, sodass es noch ein Label geben würde Separator: Muss unterschieden werden - Filtern oder nicht filtern, danach ob es das letzte mögliche Tupel vor der prediction Phase ist. Mit filtern besseres ergebnis, aber auch deutlich weniger Trainingsbeispiele (Overfitting wird mehr zur Gefahr) Ohne Filtern: Flexibler und mehr Trainingsbeispiele - man könnte im Nachhinen den PredictionCutOff verlegen und einfach das Netz weiter verwenden ohne neu zu trainieren. Maybe ein Mittelding, das man nicht alle tupel nimmt aber auch nicht nur die letzten? Ausblick, zukunft

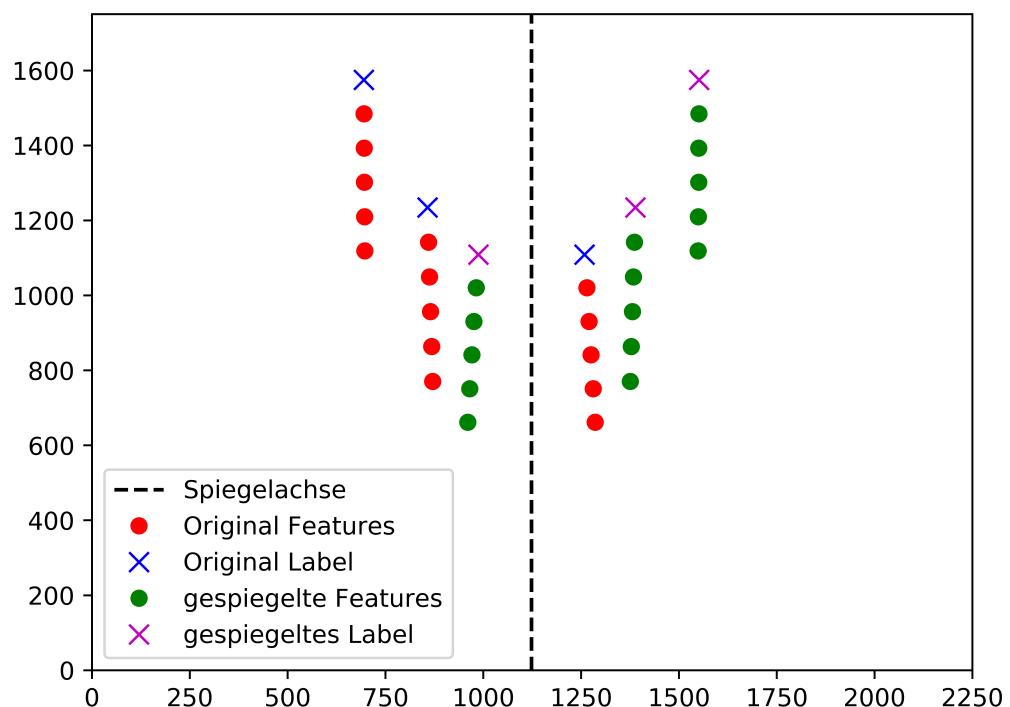


Abbildung 3.7: Visualisierung Data Augmentation durch Spiegelung

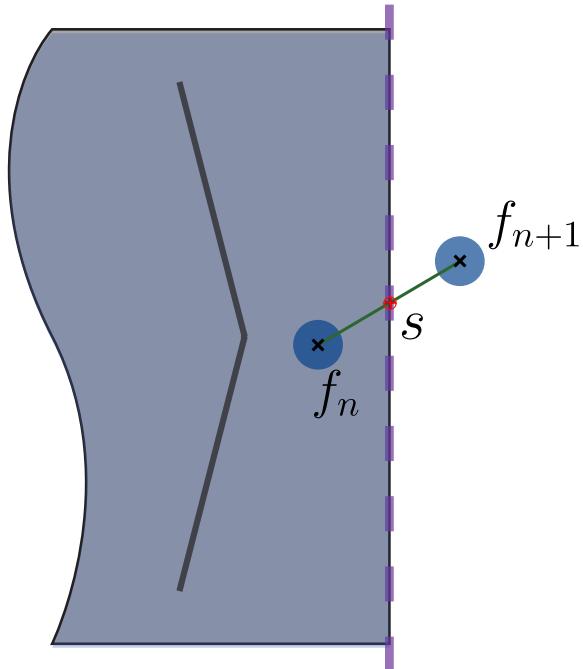


Abbildung 3.8: Geometrische Bestimmung der Labels [Vorlage aus Florians Diss - wie QUELLE?]

Example: Gegeben ein Track, was für Features würden da rausfallen

Labels: Sehr straight forward für NextStep (Literally), einfach die nächste Zeile im Track jeweils für X und Y

für separator slightly more complicated: Element des Tracks vor und hinter der Separator position (entlang der Travel Achse)

Schnittpunkt geometrisch bestimmen. Label für position ist die Position entlang der Achse orthogonal zur Bewegungsrichtung vom Schnittpunkt der Separatorlinie und der Strecke zwischen dem Element vor und dem element hinter. siehe 3.8

Verhältnis: Anzahl Feature-Label-Paare für verschiedene Beispiele und verschiedene Settings (FeatureSize, Filter Ja/Nein, Augmentation Ja/Nein) Als Tabelle?

OUTDATED: Bei einer FeatureSize von 5 ergeben sich bei den Kugeln so 98.966 Feature-Label Paare. Die Pfefferkörner haben dann 105.101 Feature-Label Paare, bei den Zylindern kommt man auf 244.422 Feature-Label Paare und bei den Weizenkörner 132.140 Feature-Label Paare.

table of size of different data sets - number of pictures...

KAPITEL 4

Umsetzung und Implementierung

4.1 Software

requirements.txt kann im Anhang gefunden werden mit der vollständigen Liste.

- Virtual Python environment.
- Implementiert in Tensorflow. (angefangen in version 1.8, später nach 1.11 upgedatet)
- Datenhandling: mit Pandas. Da Data Science ein wichtiger Part der Arbeit war, sehr wichtig erwähnen
- Matplotlib für Visualisierung (die meisten selbstgemachten grafiken hier in der Arbeit)
- OpenCV für Bilderdinge in der Pipeline (wie oben erwähnt)
- MATLAB, für Tracksort und die Ursprünglichen implementation der Vergleichsdinge für evaluationen

Aufpassen dass das nicht zu viel wird

4.2 Code Struktur

Vorgehen beschrieben in 4.1.

überlegen wie viel ich überhaupt dazu schreiben soll -

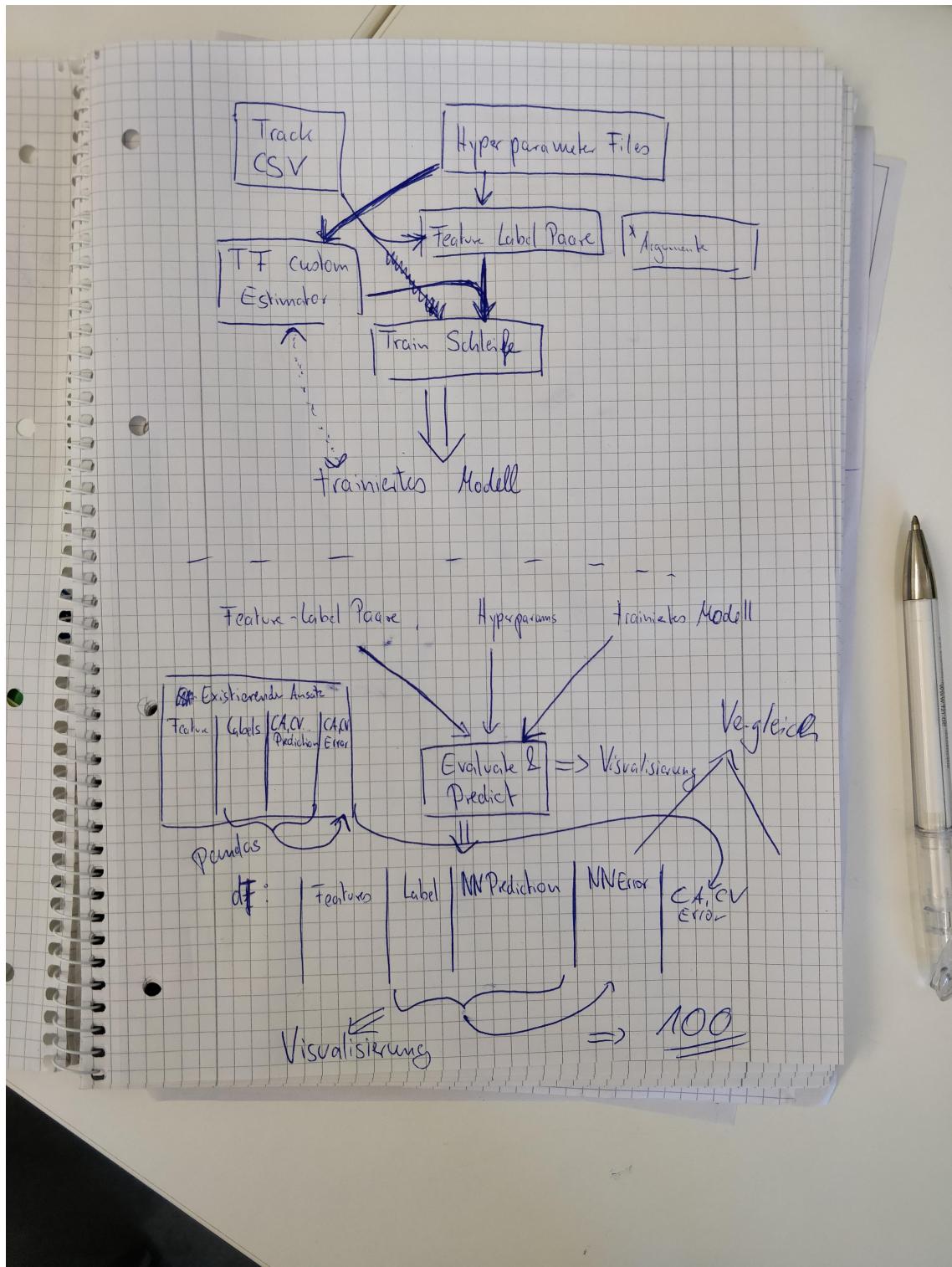


Abbildung 4.1: Skizze Codestruktur

4.3 Hyperparameter

Hyperparameter sind die Variablen, die die Struktur des Netzes bestimmen (Eg: Anzahl Layers, FeatureSize) sowie die Variablen, die festlegen wie das Netz trainiert (z.B. Lernrate, Anzahl Epochen)

Hyperparameter werden vor dem Beginn des Trainings festgelegt und bleiben während dem kompletten Training unverändert.

Vorgehen bei dieser Arbeit: Jeweils für NextStep und für Separator getrennte Konfigurationen finden.

```

1   {
2     "arch": {
3       "dropout_rate": 0.0,
4       "hidden_layers": [16, 16, 16],
5       "feature_size": 5,
6       "activation": "leaky_relu"
7     },
8     "problem": {
9       "data_path": "/home/hornberger/MasterarbeitTobias/data/
10      simulated/SpheresDownsampled",
11      "modelBasePath": "/home/hornberger/MasterarbeitTobias/
12      models/simulated/",
13      "imagePath": "/home/hornberger/MasterarbeitTobias/images/",
14      "separator": 0,
15      "separatorPosition": 1550,
16      "thresholdPoint": 1200,
17      "predictionCutOff": 1300
18    },
19    "train": {
20      "batch_size": 1000,
21      "epochs": 500,
22      "steps_per_epoch": 200,
23      "learning_rate": 0.01,
24      "optimizer": "Adam"
25    },
26    "data": {
27      "numberFakeLines": 500,
28      "testSize": 0.1,
29      "augmentMidpoint": 1123,
30      "augmentRange": 1000,
31      "direction": "x",
32      "unitLoc": "px",
33      "unitTime": "1/100 Frames",
34      "limits": [0.388, 0.788, 0.0, 0.18]
35    }
}

```

Listing 4.1: Beispiel eines Hyperparameter Files in JSON

- Architektur:
 - Dropout: Wahrscheinlichkeit für das zufälliges ausschalten von einzelnen Neuronen
 - Hidden Layer: ein Array an Zahlen repräsentiert die Architektur der Hidden

Layers. Jede Zahl ist ein FC Layer mit so vielen neuronen

- FeatureSize: Wie viele Positionen bekommt das Netz als Input (→ Größe des Inputlayers = 2x FeatureSize)
- Activation: Aktivierungsfunktionen für die neuronen der Hidden Layers

- Problem:

- DataPath: wo liegen die CSV Dateien zum die Daten rausladen
- ModelPath: wo soll das Netz hingespeichert werden/Hergeladen - mit Checkpoints usw.
- ImagePath: wo sollen Bilder hingespeichert werden, z.B. von Plot
- separator: 0 oder 1, jenachdem ob es den nächsten Schritt (0) oder zum Düsenbalken (1) prädizieren soll

Falls Separator 1:

- separationPosition: Koordinate des Düsenbalken und Ziel der Prädiktion
- ThresholdCutoff
- predictionCutOff: Koordinate hinter der keine FeatureTupel mehr genommen werden

verify

- Train:

4.3.1 Hyperparameter Tuning

Als Hyperparameter Optimierung oder auch Hyperparameter Tuning bezeichnet man den Vorgang das am besten geeignete Set an Hyperparametern für einen Lernalgorithmus zu wählen.

Vorgehen um optimale Hyperparameter zu finden: Baseline wird (über den Daumen gepeilt). Training mehrere Modelle mit variierenden Werten für einen einzelnen Parameter, während die restlichen Parameter unverändert bleiben.

So viele modelle zu trainieren ist sehr zeitintensiv, deshalb: Suchen auf Simulierten Daten (Spheres) und dann auf allen Daten verwenden.

Nicht optimal... Maybe ausblick?

Aktueller Stand: NextStep: kein Overfitting gefunden →L1 und L2 Regularisierung haben keinen positiven Effekt.

Adam Optimizer ist am besten.

erklären was genau der Adam optimizer eigentlich macht

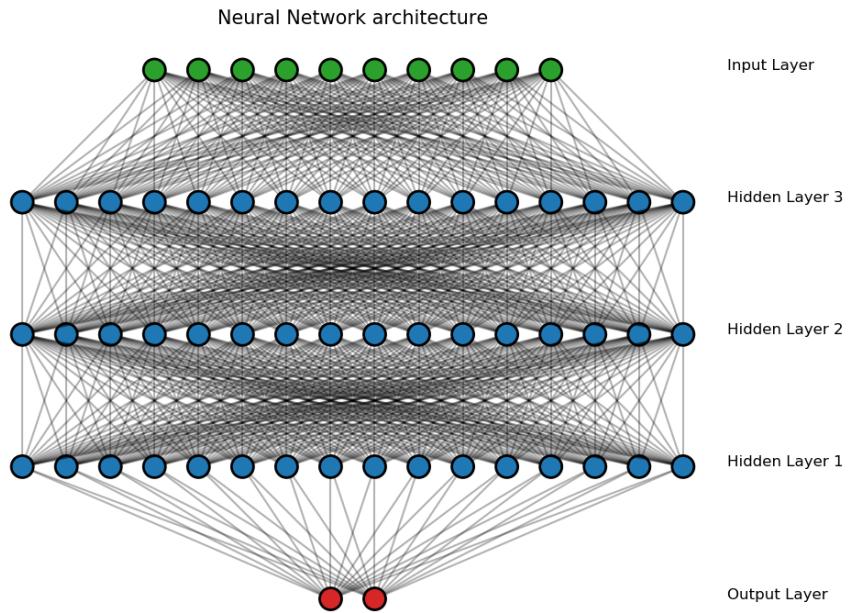


Abbildung 4.2: Architektur Neuronales Netz für die NextStep Prädiktion

leaky_relu reigns supreme.

Learning Rate Decay ist eine gute Idee. (Höherer Wert → langsamerer Zerfall).

FeatureSize 5 ist gut, weniger wird schlechter und Mehr ist auch schlechter geworden (potenziell weil weniger Beispiele?) Dropout macht es kaputt, also auf 0.0 setzen

Separator:

4.3.2 Architektur des neuronalen Netzes

Input layer: $2 * FeatureSize$ Neuronen

N hiddenlayer (as determined by Hyperparameter tuning) mit jeweils m Neuronen.
Fully connected!

Output layer: Linear activation weil regression. 2 Neuronen, eins für die eine Label Dimension und eins für die andere.



Abbildung 4.3: Architektur Neuronales Netz für die Separator Prädiktion

KAPITEL 5

Evaluation

Im vorhergegangenen Kapitel wurde beschrieben, [wie die Netze designed wurde]
Jetzt bewerten wie gut sie das eigentlich mache.

5.1 System

Trainiert und evaluiert wurde auf einem Ubuntu 18.04 Linux System. Intel i7-7700k
CPU @ 4.20 GHz, NVidia GForce 1080Ti, 11GB GDDR5X, 32GB RAM, SSD

Stats verifizieren

5.2 Vergleichsmodelle

Notation und Definition bei allen diesen Modellen nach Florian's Diss [Pfa11], mit Ausnahme von Average Acceleration. Sei $x(t)$ die Position des Partikels entlang der Bewegungsrichtung in Abhängigkeit von der Zeit (continuous-time equation). Sei $y(t)$ die Position des Partikels orthogonal zur Bewegungsrichtung in Abhängigkeit von der Zeit. t^{Last} ist der Zeitpunkt der Beobachtung des letzten Features. Sei $\Delta t = t - t^{\text{Last}}$.

5.2.1 Constant Velocity Modell

Soll

Zustandsvector für CV:

$$\underline{x}_t = \begin{bmatrix} x_t \\ \dot{x}_t \\ y_t \\ \dot{y}_t \end{bmatrix}$$

$$\dot{\underline{x}}(t) = \mathbf{A}\underline{x}(t), \quad \mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Definition Positionsgleichungen:

$$x(t) = x^{\text{Last}} + (t - t^{\text{Last}})\dot{x}^{\text{Last}}$$

$$y(t) = y^{\text{Last}} + (t - t^{\text{Last}})\dot{y}^{\text{Last}}$$

5.2.2 Contant Acceleration Modell

Zustandsvector für CA:

$$\underline{x}_t = \begin{bmatrix} x_t \\ \dot{x}_t \\ \ddot{x}_t \\ y_t \\ \dot{y}_t \\ \ddot{y}_t \end{bmatrix}$$

$$\dot{\underline{x}}(t) = \mathbf{A}\underline{x}(t), \quad \mathbf{A} = \begin{bmatrix} \mathbf{A}_x & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_y \end{bmatrix}, \quad \mathbf{A}_x = \mathbf{A}_y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Definition Positionsgleichungen:

$$x(t) = x^{\text{Last}} + (t - t^{\text{Last}})\dot{x}^{\text{Last}} + \frac{1}{2}(t - t^{\text{Last}})^2 \ddot{x}^{\text{Last}}$$

$$y(t) = y^{\text{Last}} + (t - t^{\text{Last}})\dot{y}^{\text{Last}} + \frac{1}{2}(t - t^{\text{Last}})^2 \ddot{y}^{\text{Last}}$$

Vergleich über Boxplots: Balken: 25%-Quantil bis 75%-Quantil. Roter Balken ist der Median. Der obere und der untere Whisker gehen bis zum höchsten bzw. niedrigsten Wert, der nicht mehr als 2.7 Standardabweichungen vom Median abweicht. Die Outlier werden nicht angezeigt.

5.2.3 Bias-Corrected Constant Velocity Modell

Bias in der Zeit am Trainingsset bestimmen und neuen Wert für Ortsprädiktion benutzen.

$$t^{\text{Pred, CVBC}} = t^{\text{Pred, CV}} - t^{\text{Bias}}$$

5.2.4 Average Acceleration Modell

Für alle Elemente des Trainingssets: Bestimme die Beschleunigungen. Sei \ddot{x}^{Median} der Median von all diesen Beschleunigungen. Benutze ihn als Beschleunigung wie im CA Modell

(Basicly CV Modell + eine Beschleunigung basierend auf den Trainingsdaten)

überhaupt erwähnen? Er ist way better als er irgendein right hat

$$x(t) = x^{\text{Last}} + (t - t^{\text{Last}}) \dot{x}^{\text{Last}} + \frac{1}{2}(t - t^{\text{Last}})^2 \ddot{x}^{\text{Median}}$$

5.2.5 Identical Acceleration Modell

Upgrade zu CVBC: Correction Term, der den die Letzte Position des Partikels einbezieht.

Annahme: Abweichungen bezüglich dem Zeit Label wird von einer zusätzlichen Beschleunigung verursacht. Für jedes Partikel i aus dem Trainingsset lösen wir die Gleichung

$$x^{\text{PredTo}} = x^{\text{Last}, i} + (t^{\text{GT}, i} - t^{\text{Last}, i}) \dot{x}^{\text{Last}, i} + \frac{1}{2}(t^{\text{GT}, i} - t^{\text{Last}, i})^2 \ddot{x}^{\text{Optimal}, i}$$

um herauszufinden mit welcher zusätzlichen Beschleunigung $\ddot{x}^{\text{Optimal}, i}$ es optimal die Zeit, die es noch braucht, vorhersagen würde. Nun sei \ddot{x}^{Avg} der Durchschnitt von allen $\ddot{x}^{\text{Optimal}, i}$.

fertig machen

5.3 Next Step

Netz Variante 1: den nächsten Schritt vorhersagen Δt ist immer 1.

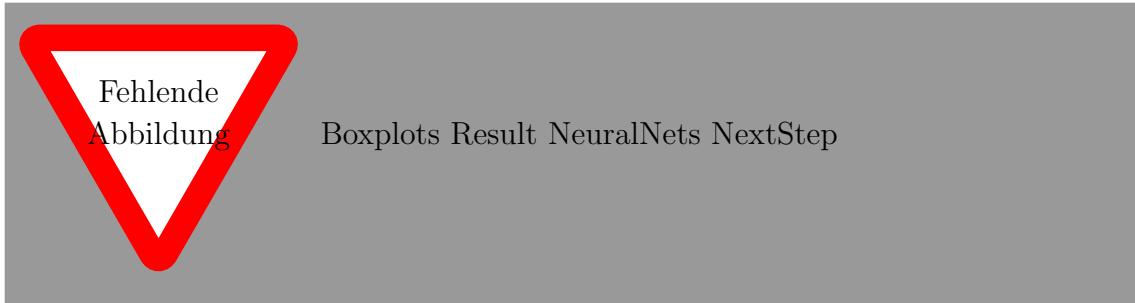


Abbildung 5.1: Evaluation für die NextStep Prädiktion

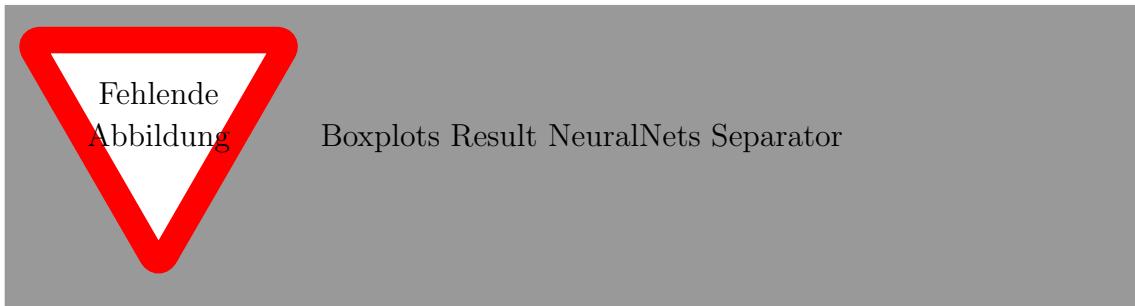


Abbildung 5.2: Evaluation für die Separator Prädiktion

Für Nextstep gesucht: $x(t^{\text{Last}} + 1)$

Latex Tabelle des pandas Dataframe mit den Spalten GroundtruthX, GroundtruthY, NNPrädiktionX, NNPrädiktion, CV_X, CV_Y, CA_X, CA_Y

Evaluation nach Euklidischer Distanz zu Groundtruth:

$$\begin{aligned}x^{\text{Err}} &= x^{\text{Pred}} - x^{\text{GT}} \\y^{\text{Err}} &= y^{\text{Pred}} - y^{\text{GT}} \\e^{\text{Total}} &= \sqrt{x^{\text{Err}}{}^2 + y^{\text{Err}}{}^2}\end{aligned}$$

- CV, CA - Ergebnis Netz - Ergebnis Lineare Regression

5.4 Separator

CV, CA quasi wie oben. zusätzlich: CVBC, AA und IA

- Ergebnis NN - Ergebnis Lineare Regression

KAPITEL 6

Fazit und Ausblick

einführungstext in Fazit und Ausblick

Im Rahmen dieser Arbeit wurde gezeigt, dass Neuronale Netze ein Werkzeug sind, mit dem man (was anfangen kann in dem Kontext)

blick darauf wie es gelaufen ist...

Mehr daten für separator!

Zylinder war eher so Meh.

was man noch so machen könnte...

Ende zu Ende lernen: Sollte das Problem mit dem segmentieren lösen, das ich hatte (Sprengt aber vielleicht den Rahmen einer MA)

Orientierung als Feature, das man noch reinnehmen könnte?

Lernen während dem laufenden Betrieb?

Mehrere Partikel gleichzeitig betrachten? Irgendwie mit Kollisionen umgehen

Literatur

- [MP43] W. S. McCulloch und W. Pitts, „A Logical Calculus of the Ideas Immanent in Nervous Activity“, *The bulletin of mathematical biophysics*, Jg. 5, Nr. 4, S. 115–133, 1943.
- [Ros58] F. Rosenblatt, „The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.“, *Psychological review*, Jg. 65, Nr. 6, S. 386, 1958.
- [KSH12] A. Krizhevsky, I. Sutskever und G. E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks“, in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou und K. Q. Weinberger, Hrsg., Curran Associates, Inc., 2012, S. 1097–1105. Adresse: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- [Dol15] F. Doll, „Konzeption, Planung, Konstruktion Und Integration Eines Miniaturisierten, Modularen Schüttgutsortierers“, KIT, Aug. 2015.
- [Pfa+17] F. Pfaff, G. Kurz, C. Pieper, G. Maier, B. Noack, H. Kruggel-Emden, R. Gruna, U. D. Hanebeck, S. Wirtz, V. Scherer, T. Langle und J. Beyerer, „Improving Multitarget Tracking Using Orientation Estimates for Sorting Bulk Materials“, IEEE, Nov. 2017, S. 553–558, ISBN: 978-1-5090-6064-1. DOI: [10.1109/MFI.2017.8170379](https://doi.org/10.1109/MFI.2017.8170379).
- [All14] Allied Vision Technologies GmbH, *User Manual for Bonito CL-400 200 Fps High Speed Camera*. Nov. 2014, Rev. I.
- [Pie+16] C Pieper, H Kruggel-Emden, S Wirtz, V Scherer, F Pfaff, B Noack, U. Hanebeck, G Maier, R Gruna, T Langle und others, „Numerical Investigation of Optical Sorting Using the Discrete Element Method“, in *International Conference on Discrete Element Methods*, Springer, 2016, S. 1105–1113.

- [Pie+17] C Pieper, F Pfaff, G Maier, H Kruggel-Emden, S Wirtz, B Noack, R Gruna, V Scherer, U. Hanebeck, T Längle und others, „Numerical Modelling of an Optical Belt Sorter Using a DEM–CFD Approach Coupled with Particle Tracking and Comparison with Experiments“, 2017.
- [Pfa11] F. Pfaff, „Multitarget Tracking Using Orientation Estimation for Optical Belt Sorting“, Dissertation, Karlsruher Institut für Technologie, Karlsruhe, 13.11.2018 (to appear).
- [Ema+18] P. Emami, P. M. Pardalos, L. Elefteriadou und S. Ranka, „Machine Learning Methods for Solving Assignment Problems in Multi-Target Tracking“, *arXiv preprint arXiv:1802.06897*, 2018.
- [Mil+17] A. Milan, S. H. Rezatofighi, A. R. Dick, I. D. Reid und K. Schindler, „Online Multi-Target Tracking Using Recurrent Neural Networks.“, in *AAAI*, 2017, S. 4225–4232.
- [Pfa+15] F. Pfaff, M. Baum, B. Noack, U. D. Hanebeck, R. Gruna, T. Längle und J. Beyerer, „TrackSort: Predictive Tracking for Sorting Uncooperative Bulk Materials“, in *2015 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, Sep. 2015, S. 7–12. DOI: [10.1109/MFI.2015.7295737](https://doi.org/10.1109/MFI.2015.7295737).
- [Pfa+16] F. Pfaff, C. Pieper, G. Maier, B. Noack, H. Kruggel-Emden, R. Gruna, U. D. Hanebeck, S. Wirtz, V. Scherer, T. Längle und others, „Improving Optical Sorting of Bulk Materials Using Sophisticated Motion Models“, *tm-Technisches Messen*, Jg. 83, Nr. 2, S. 77–84, 2016.
- [Cos+17] H. Coskun, F. Achilles, R. S. DiPietro, N. Navab und F. Tombari, „Long Short-Term Memory Kalman Filters: Recurrent Neural Estimators for Pose Regularization“, *CoRR*, Jg. abs/1708.01885, 2017. Adresse: <http://arxiv.org/abs/1708.01885>.
- [LSZ09] J. Langford, R. Salakhutdinov und T. Zhang, „Learning Nonlinear Dynamic Models“, *CoRR*, Jg. abs/0905.3369, 2009. Adresse: <http://arxiv.org/abs/0905.3369>.
- [LJ03] X. R. Li und V. P. Jilkov, „Survey of Maneuvering Target Tracking. Part I. Dynamic Models“, *IEEE Transactions on aerospace and electronic systems*, Jg. 39, Nr. 4, S. 1333–1364, 2003.
- [Abb+05] P. Abbeel, A. Coates, M. Montemerlo, A. Y. Ng und S. Thrun, „Discriminative Training of Kalman Filters“, in *In Proceedings of Robotics: Science and Systems*, 2005.

- [KSS15] R. G. Krishnan, U. Shalit und D. Sontag, „Deep Kalman Filters“, 16. Nov. 2015. arXiv: 1511.05121 [cs, stat]. Adresse: <http://arxiv.org/abs/1511.05121> (besucht am 14.05.2018).
- [WF09] R. Wilson und L. Finkel, „A Neural Implementation of the Kalman Filter“, in *Advances in Neural Information Processing Systems*, 2009, S. 2062–2070.
- [Fra] Fraunhofer IOSB, *TableSort Schüttgutsortierer*.
- [GBC16] I. Goodfellow, Y. Bengio und A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [LBH15] Y. LeCun, Y. Bengio und G. Hinton, „Deep Learning - Nature Article“, *Nature*, Jg. 521, S. 436, 27. Mai 2015. Adresse: <https://doi.org/10.1038/nature14539>.
- [Nie15] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination press USA, 2015, Bd. 25.

ANHANG A

Anhang

Hier ist der Anhang. Hier kommen Dinge Rein, wie Evaluationsergebnisse, die den Hauptteil zu voll machen würden, Tabellen mit daten, die nur begrenzt was mit der Arbeit zu tun haben,