

---

# 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**

Abstract.



# **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

<b>Abbildungsverzeichnis</b>	<b>III</b>
<b>Notation</b>	<b>V</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Aufbau der Arbeit . . . . .	1
<b>2 Grundlagen</b>	<b>3</b>
2.1 Das Kalman-Filter . . . . .	3
2.2 Neuronale Netze . . . . .	4
2.2.1 Perzeptron . . . . .	4
2.2.2 Aktivierungsfunktionen . . . . .	5
2.2.3 Feedforward Netze . . . . .	7
2.2.4 Backpropagation . . . . .	8
2.2.5 Overfitting . . . . .	8
2.2.6 Regularisierung . . . . .	9
2.3 TableSort System . . . . .	10
2.4 Tensorflow . . . . .	10
2.5 Related Work . . . . .	11
<b>3 Datenverarbeitung</b>	<b>13</b>
3.1 Datenformatierung . . . . .	13
3.2 Eigene Aufnahmen . . . . .	14
3.2.1 Versuchsaufbau . . . . .	14
3.2.2 Schüttgüter-Typen . . . . .	14
3.3 Datenpipeline . . . . .	15
3.4 Simulierte Daten . . . . .	16
3.4.1 Menge . . . . .	17
3.5 Daten Postprocessing . . . . .	17
3.6 Trainingsbeispiele . . . . .	19

<b>4 Umsetzung und Implementierung</b>	<b>21</b>
4.1 Software . . . . .	21
4.2 Code Struktur . . . . .	21
4.3 Hyperparameter . . . . .	21
4.3.1 Hyperparameter Tuning . . . . .	23
4.3.2 Architektur des neuronalen Netzes . . . . .	23
<b>5 Evaluation</b>	<b>25</b>
5.1 System . . . . .	25
5.2 Next Step . . . . .	25
5.3 Separator . . . . .	26
<b>6 Fazit und Ausblick</b>	<b>27</b>
6.1 Fazit . . . . .	27
6.2 Ausblick . . . . .	27
<b>Literatur</b>	<b>29</b>

# Abbildungsverzeichnis

2.1	Plot der Sigmoid Funktion . . . . .	6
2.2	Plot der Tanh Funktion . . . . .	7
2.3	Plot der Ausgabe einer ReLUs . . . . .	8
2.4	Häufig verwendete Aktivierungsfunktionen . . . . .	8
2.5	Beispiel für Overfitting . . . . .	9
2.6	TableSort Schüttgutsortiersystem [TODO Quelle] . . . . .	10
3.1	Zur Aufnahme verwendete Kamera [TODO: Quelle Bild] . . . . .	14
3.2	Verschiedene Schüttgüter . . . . .	15
3.3	Bayer Matrix [TODO: Quelle] . . . . .	16
3.4	Verschiedene Schüttgüter auf dem Förderband . . . . .	17
3.5	Verteilung Schüttgut Elemente nach Sorte . . . . .	18
4.1	Architektur NN NextStep [TODO Quelle] . . . . .	24



# 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

Einleitungstext

### 1.1 Motivation

arbeit  
motivieren.  
Schüttgutsortier  
ist ein inter-  
essantes  
Feld, das sich  
potenziell für  
ML anbietet.

### 1.2 Aufbau der Arbeit

auf jeden fall  
separator und  
nextStep Netze  
unterscheidung  
erwähnen

aufbau Gliede-  
rung beschrei-  
ben



## 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 Das Kalman-Filter

Als Kalman-Filter bezeichnet man ein mathematisches Verfahren mit dem Messfehler in realen Messwerten reduziert werden können und nicht messbare Systemgrößen geschätzt werden können.

[vergangene, aktuelle und zukünftige Systemzustände schätzen] [Einschränkung Linearität (Extended Kalman) und Gauß rauschen]

Der Zustand des Systems zum Zeitschritt  $t$  wird als  $y_t$  und die Messung im Zeitschritt  $t$  als  $z_t$  bezeichnet.

$$y_t = Ay_{t-1} + w, w \sim N(0, Q)$$

$$z_t = Hy_t + v, v \sim N(0, R)$$

Dabei ist  $A$  die Zustandsübergangsmatrix, die den Übergang von einem Zustand in den nächsten beschreibt.  $H$  ist die Messmatrix, die beschreibt wie Messungen aus dem Zustand entstehen und  $Q$  und  $R$  sind die Kovarianzmatrizen des Systemrauschens beziehungsweise des Messrauschens.

Das Kalman-Filter funktioniert mittels abwechselnd ausgeführter *predict* und *update* Schritte.

$$\hat{y}'_t = A\hat{y}'_{t-1}$$

$$\hat{P}'_t = A\hat{P}'_{t-1}A^T + Q$$

## 2.2 Neuronale Netze

Als Neuronale Netze bezeichnet man in der Informatik Systeme aus künstlichen Neuronen, die heute eine wichtige Rolle im Feld des maschinellem 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 inspiriert sind.

Die Grundsteine des Feldes wurde bereits 1943 von Warren McCulloch und Walter Pitts gelegt, die in ihrem Paper [[mcculloch1943logical](#)] 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.

### 2.2.1 Perzeptron

Die kleinste Einheit eines neuronalen Netzes ist das Perzeptron, 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 *ausgibt*.

Darstellung des  
Perzeptrons

F

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 erhält man dadurch, dass man die gewichteten Eingaben aufsummiert und in die Aktivierungsfunktion des Perzeptrons gibt. Ein Überblick über verschiedene Aktivierungsfunktionen ist unter 2.2.2 zu finden.

Mathematisch ist die Ausgabe eines Perzeptrons 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 Perzeptron mit zwei Eingängen kann zur Darstellung der logischen Operatoren AND, OR und NOT genutzt werden.

Letztendlich ist ein solches Perzeptron jedoch nur ein linearer Klassifikator und kann somit zum Beispiel den XOR Operator nicht auflösen. Dies zeigten Marvin Minsky und Seymour Papert 1969 in einflussreichen Buch *Perceptrons*

Um solche, nicht linear-separierbare Probleme zu lösen müssen mehrere Schichten an Neuronen kombiniert werden.

quelle

Linear Trennbares / Nicht-linear Trennbares Problem (AND vs XOR)

## 2.2.2 Aktivierungsfunktionen

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

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

Einige häufig verwendete Aktivierungsfunktionen sollen hier vorgestellt werden. 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.

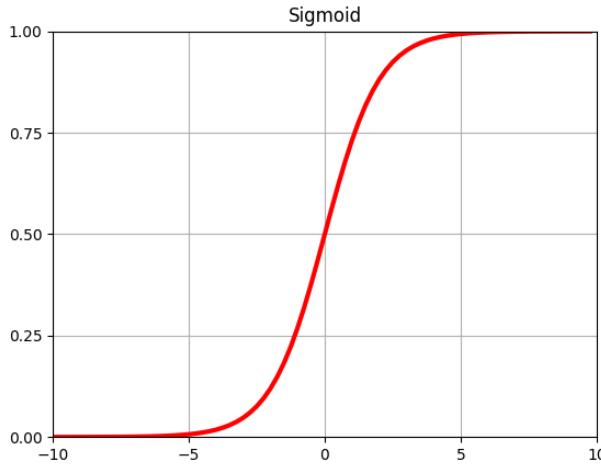
### Sigmoid-Funktion

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

$$f'(x) = f(x) * (1 - f(x))$$

Die mathematische Form der Sigmoid Aktivierungsfunktion ist in Abbildung 2.1 zu sehen. Sie bildet die reellen Zahlen  $\mathbb{R}$  auf das Intervall  $(0, 1)$  ab. Für betragsmäßig größer werdende negative Zahlen nähert sich der Rückgabewert 0 an, ebenso wie für größer werdende positive Zahlen sich der Rückgabewert an 1 annähert.

Die Sigmoid Funktion ist eine historisch häufig genutzte Funktion, da sie das Verhalten eines natürlichen Neurons, der biologischen Motivation für künstliche Neuronen, gut nachbildet: komplette Inaktivität eines Neurons bei Ausgabe 0 bis zum Feuern mit maximaler Frequenz bei Ausgabe 1.



**Abbildung 2.1:** Plot der Sigmoid Funktion

In der Praxis jedoch haben sich einige Nachteile der Sigmoid Funktion gezeigt, weshalb sie quasi nicht mehr genutzt wird. Der wichtigste von diesen ist, dass ihre Ableitung bei großen Werten beinahe 0 ist. Dies führt dazu, dass während der Ausführung des Backpropagation-Algorithmus beinahe keine Änderungen passieren und dementsprechend das Netz sehr langsam lernt.

### TanH

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

$$f'(x) = 1 - f(x)^2$$

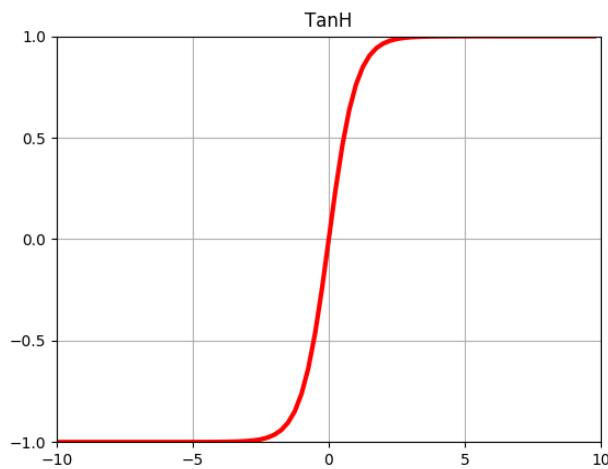
Die  $\tanh$  Aktivierungsfunktion ist in Abbildung 2.2 dargestellt. Im Gegensatz zur Sigmoid Funktion bildet sie die reellen Zahlen  $\mathbb{R}$  auf das Intervall  $(-1, 1)$  ab. Weil sie zentriert um den Nullpunkt ist, wird sie bei realen Anwendungen der Sigmoid Funktion vorgezogen. Das Saturationsproblem der Sigmoid Funktion besteht jedoch immer noch.

### ReLU

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

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

Abbildung 2.3 zeigt den Plot einer *Rectified Linear Unit*, oder kurz ReLU. Die Aktivierung von ReLUs ist ein einfacher Schwellwert, der weit weniger rechenintensiv ist, als die aufwendigen Exponentialfunktionen von Sigmoid und



**Abbildung 2.2:** Plot der Tanh Funktion

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

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

### 2.2.3 Feedforward Netze

**absatz über Feedforward Netze. Basic** Als Feedforward Netz bezeichnet man ein neuronales Netz, zwischen dessen Knoten keine Kreise oder Schleifen existieren. 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 eingegeben.

- Outputlayer: Verschiedene Aktivierungsfunktionen: . Linear für regression, z.B. Softmax für Wahrscheinlichkeitsverteilung Softmax [One Hot encoding?]

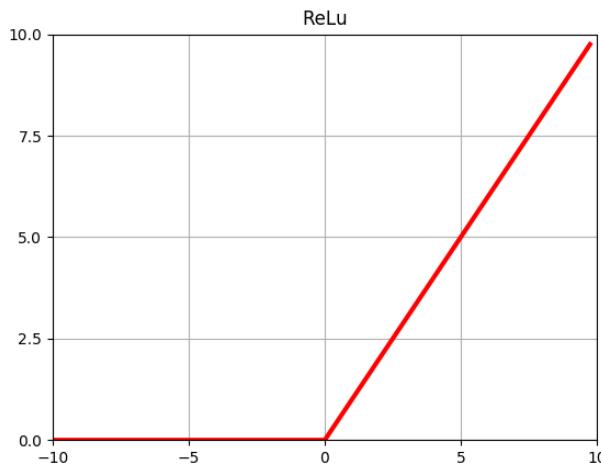


Abbildung 2.3: Plot der Ausgabe einer ReLUs

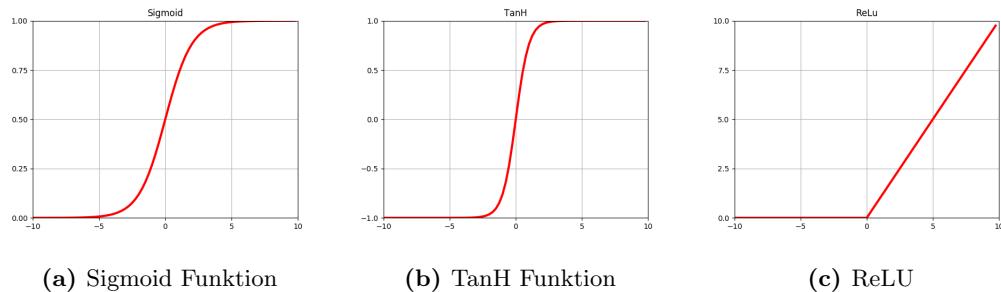


Abbildung 2.4: Häufig verwendete Aktivierungsfunktionen

## 2.2.4 Backpropagation

ein Absatz über lernen mit dem Backpropagation Algorithmus

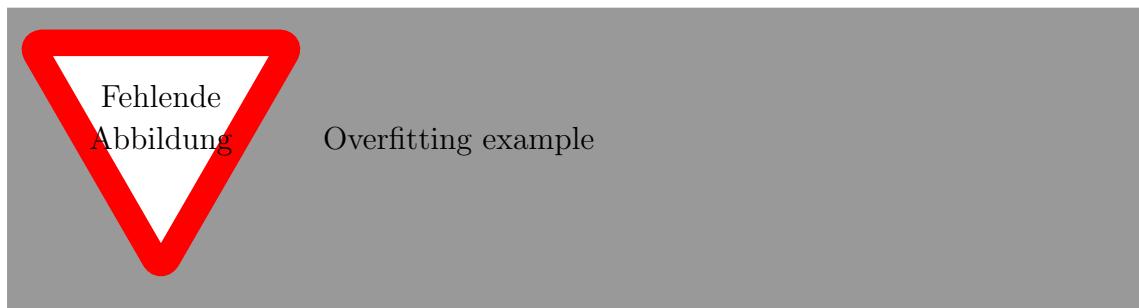
Der Backpropagation Algorithmus ist ein Verfahren mit denen künstliche neuronale Netze in der Lage sind komplizierte Zielfunktionen einzulernen. Es ist eine Methode, bei der effizient der Gradient der Fehlerfunktion in Abhängigkeit vom Gewicht der einzelnen Kanten im Netz bestimmt werden kann, was dann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

- 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“

## 2.2.5 Overfitting

Absatz zu Overfitting

Overfitting: Wenn das System schlechter darin wird zu generalisieren: (von bekanntem auf unbekanntes schließen/Schlechte Prediktion) Erkennen die Performance auf dem



**Abbildung 2.5:** Beispiel für Overfitting

Trainingsset weiterhin besser wird, aber die Performance auf dem Testset schlechter wird. z.B. wenn Rauschen als teil der zugrundeliegenden Struktur interpretiert wird.

Gegenteil von Underfitting, wenn das Modell nicht ausreichend komplex ist um die zugrundeliegende Struktur der Daten abzubilden. Beispiel underfitting: Lineares Modell auf nicht-lineare Daten fitten.

Methoden um Overfitting zu vermeiden: Mehr Trainingsdaten (z.B. durch Data-Augmentation) Regularisierung (L1, L2 (siehe unten), dropout für Classifier(?))

## 2.2.6 Regularisierung

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.

Die Grundidee ist, dass zur loss function ein Regularisierungsterm hinzugefügt wird, der die Kosten basierend auf der Komplexität des Systems erhöht.

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

Dabei ist  $V$  die loss function, beispielsweise *Mean-Square-Error* oder *Mean-Absolute-Error*.  $n$  ist die Anzahl der Feature-Label-Paare,  $\underline{x}_i$  und  $\underline{y}_i$  sind die einzelnen Eingabe-features und das dazugehörige Label. Die Funktion  $f$  ist in unserem Fall das neuronale Netz, das die Features entgegen nimmt.  $\lambda$  ist ein Parameter, der die Gewichtung des Regularisierungsterm festlegt. Wählt man diesen Parameter zu klein, so kann es sein, dass das Modell immer noch 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. Beispiele für  $R$  wären zum Beispiel die L1- oder die L2-Regularisierung die jeweils mit der L1 beziehungsweise mit der L2 Norm arbeiten.

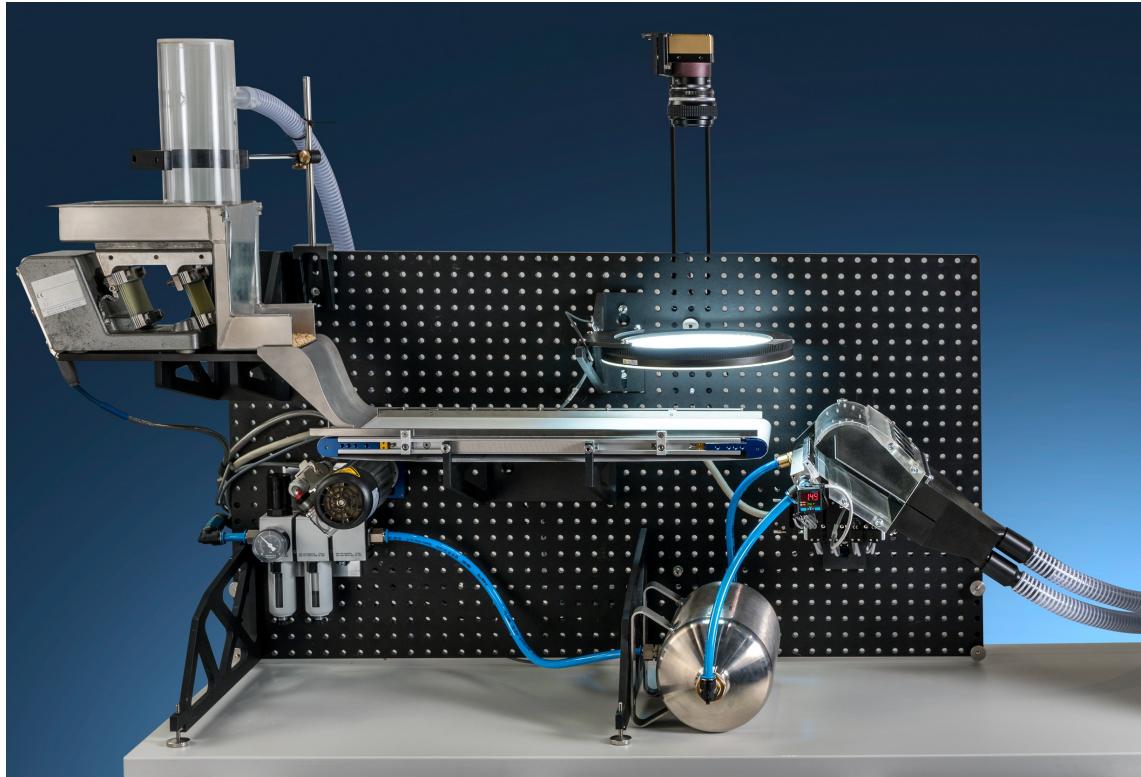


Abbildung 2.6: TableSort Schüttgutsortiersystem [TODO Quelle]

Ein gutes Maß für die Komplexität eines neuronalen Netzes sind die Gewichte zwischen den Neuronen.

Formel für MSE mit L1 Regularisierung findet man in 2.2.6.

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

absatz zu regularization - L1, L2, (L0 und warum man es nicht benutzt?)

## 2.3 TableSort System

[Viel stuff über das TableSort System]

kleiner, experimenteller Bandsortierer [Dol15]

## 2.4 Tensorflow

Was ist Tensorflow, wo kommt es her.

## 2.5 Related Work

Notizen:

Long Short-Term Memory Kalman Filters: Recurrent Neural Estimators for Pose Regularization von Huseyin et al. - Motion Model und Measurement Model mittels LSTM Netzen lernen. - Am Beispiel Pose-Estimation. - 3 Separate LSTMs, je eins für Motionmodel, System und Messrauschen. (A = Transition Matrix, H = Measurement Matrix, Q = Covariance of process noise, R = Covariance of sensor noise)



## 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 Format übersetzt werden. Zudem existieren der DEM Datensatz

mehr details  
- was tut der  
überhaupt

### 3.1 Datenformatierung

Zu Beginn des Datenverarbeitungskapitels 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.

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



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

## 3.2 Eigene Aufnahmen

### 3.2.1 Versuchsaufbau

[Beschreibung von der Bonito Kamera, stats usw.] 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 3.1 oberhalb des Förderbandes angebracht. Die Bilder, die von der Kamera aufgenommen werden, haben eine Auflösung von 2320x1726 Pixeln.

### 3.2.2 Schüttgüter-Typen

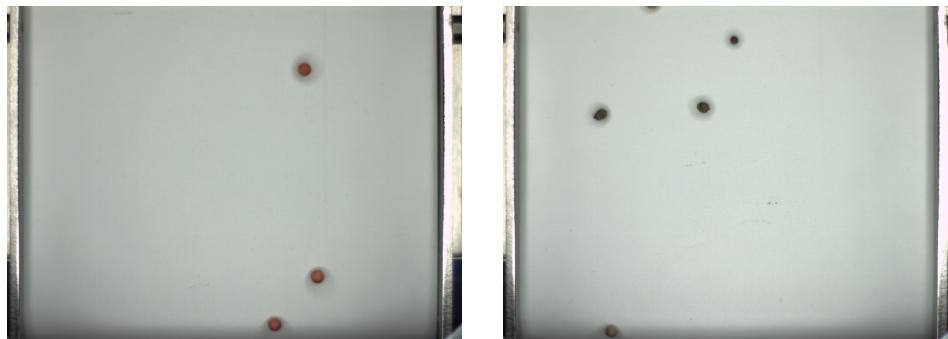
Aufgenommen wurden vier verschiedene Schüttgüter, die in Abbildung 3.2 zu sehen sind.

- 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.

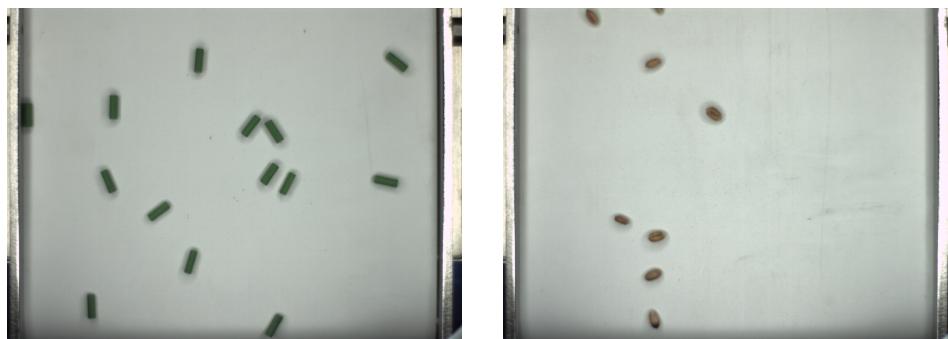
Die Kugeln bestehen aus Holz und haben einen Durchmesser von 5mm. Die Zylinder bestehen ebenfalls aus Holz. Sie haben eine Länge von 1cm und einen Durchmesser von 3mm. Die Schüttgüter sind in Abbildung 3.2 in Schüsseln und in Abbildung 3.4 auf dem Förderband zu sehen.

TODO: Details



(a) Kugeln

(b) Pfeffer



(c) Zylinder

(d) Weizen

**Abbildung 3.2:** Verschiedene Schüttgüter

### 3.3 Datenpipeline

Die Bilder wurden in Batches von je 3500 gesammelt. Die Bonito Kamera schreibt sie in Form einer Bayer-Matrix in Bitmap Dateien.

hier Bayer-Matrix erklären und Bild?

Auf Grund der Menge an Bildern war es sinnvoll die Dateien in das png Dateiformat zu 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 und die eingesetzt wurde um die einzelnen Bilder in RGB Farbbilder zu konvertieren.

Skript ursprünglich von Georg, ein paar changes implementiert bezüglich input

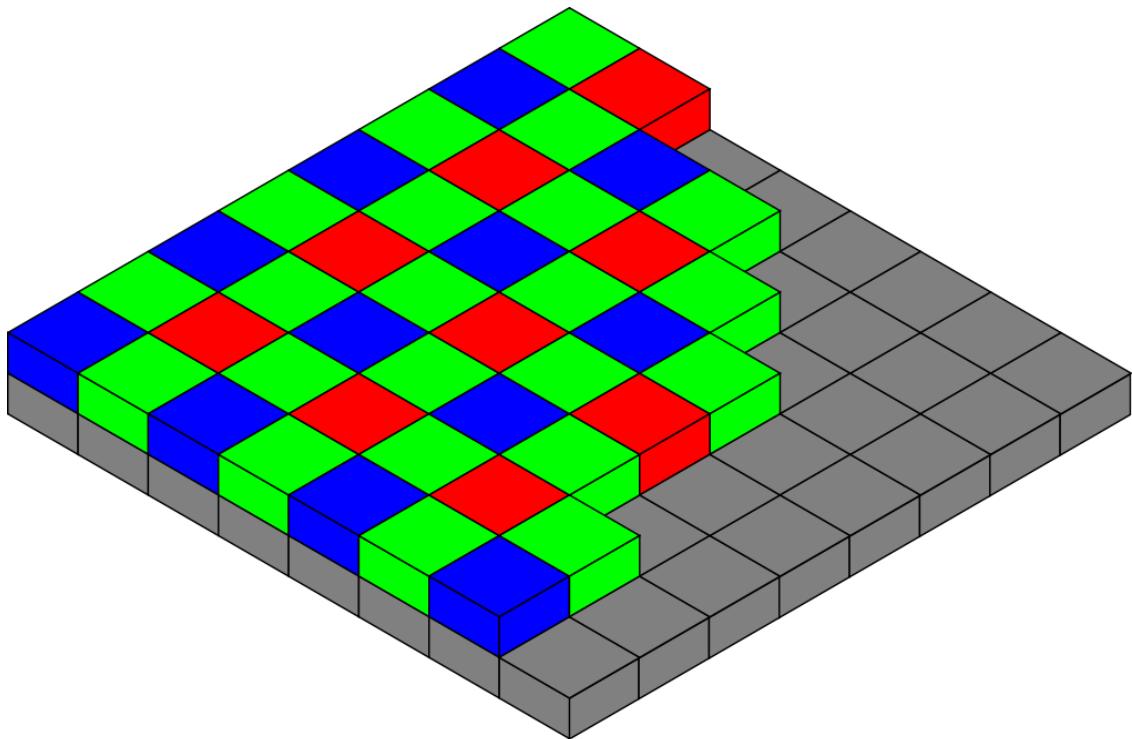


Abbildung 3.3: Bayer Matrix [TODO: Quelle]

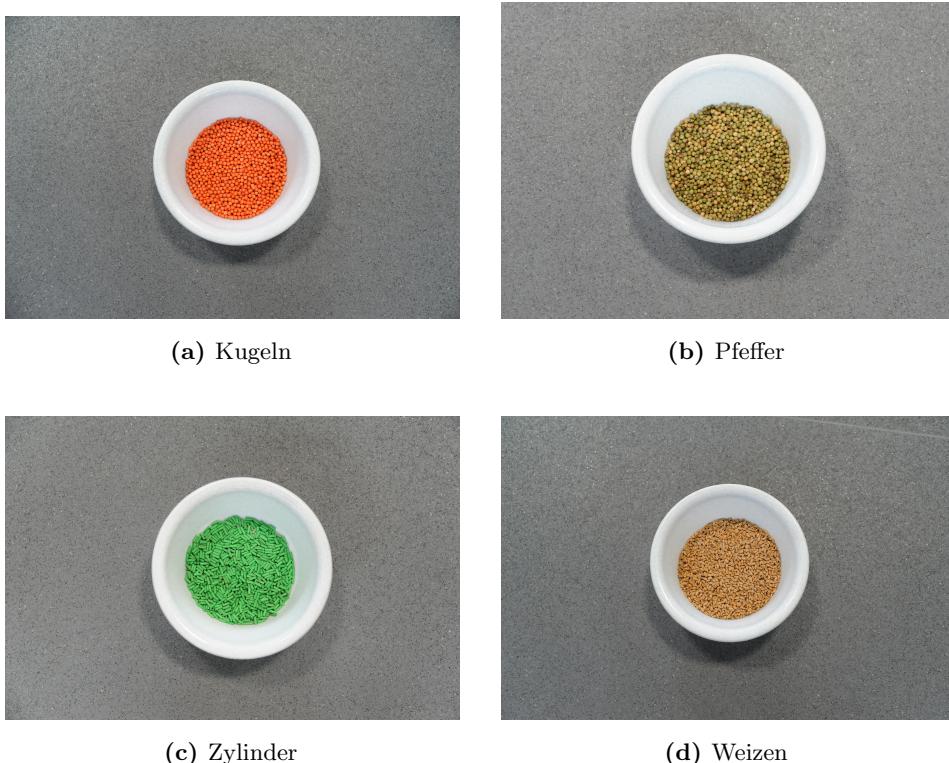
Mehr details:  
Parameters...

TODO: Verifi-  
zieren

Auf diesen kann dann eine Segmentierung vorgenommen werden. Hierzu wurde erneut die Computer Vision Library OpenCV benutzt. Das Ergebnis des Segmentierungsscripts [Reference segment.py] ist ein CSV File für jedes Batch. Eine Zeile repräsentiert 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 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 3.1 zu sehen.

### 3.4 Simulierte Daten

Die DEM Daten, wo sie herkommen, was der unterschied ist zu den selbstaufgenommenen Daten. Vorteile und Nachteile...



**Abbildung 3.4:** Verschiedene Schüttgüter auf dem Förderband

### 3.4.1 Menge

Insgesamt wurden 177954 Bilder aufgenommen.

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

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.

outdated: neue  
Zählung ist  
notwendig

Menge in gecleaneten Daten:

7343 Kugeln 6824 Pfefferkörner 15760 Zylinder 8426 Weizenkörner

## 3.5 Daten Postprocessing

[FilterTracksByAngle] [FilterByVectorLengthChange]

[Data Augmentation: Spiegeln]

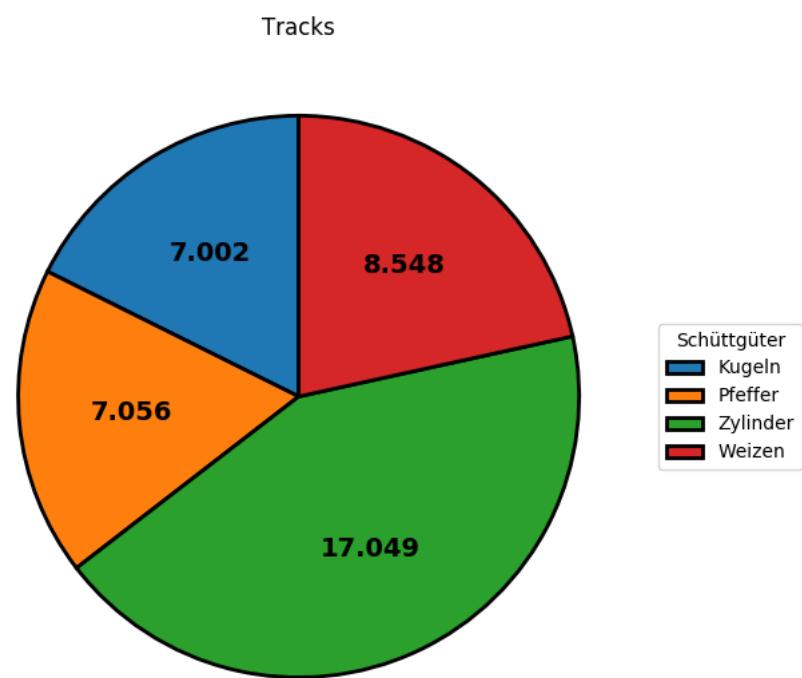


Abbildung 3.5: Verteilung Schüttgut Elemente nach Sorte

**Tabelle 3.1:** Beispielhafter Ausschnitt aus einem CSV File

TrackID_1_X	TrackID_1_Y	TrackID_2_X	TrackID_2_Y	TrackID_3_X	TrackID_3_Y
1169	1357	465	1294	1450	364
1168	1449	461	1383	1448	457
1167	1540	457	1471	1446	549
1166	1633	453	1559	1445	642
NaN	NaN	449	1647	1443	734
NaN	NaN	NaN	NaN	1441	826
NaN	NaN	NaN	NaN	1439	919
NaN	NaN	NaN	NaN	1437	1012

Als Data Augmentation bezeichnet man Verfahren, die das eigene Set an Daten erweitern ohne zusätzliche Daten aufzunehmen in dem man aus den bestehenden Daten synthetische Daten generiert. Ausreichend viele Datenpunkte zu haben ist notwendig um mit neuronalen Netzen eine gute Performance zu erzielen. Die synthetischen Beispiele müssen jedoch plausibel sein, da sie sonst die performance verschlechtern

nur Trainings-set  
performance verschlechtern

- 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.

## 3.6 Trainingsbeispiele

Train - Test - Validation - Split.

Features sind für NextStep und Separator gleich. (Unclear yet:Separator müssen insgesamt die Trainingsbeispiele gefiltert werden? Zumindest für die Evaluation? TODO!)

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

für separator slightly more complicated:



## KAPITEL 4

# Umsetzung und Implementierung

### 4.1 Software

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

### 4.2 Code Struktur

### 4.3 Hyperparameter

```
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/
simulated/SpheresDownsampled",
10         "modelBasePath": "/home/hornberger/MasterarbeitTobias/
models/simulated/",
11         "imagePath": "/home/hornberger/MasterarbeitTobias/images/",
12         "separator": 0,
13         "separatorPosition": 1550,
14         "thresholdPoint": 1200,
```

```

15         "predictionCutOff": 1300
16
17     },
18     "train": {
19         "batch_size": 1000,
20         "epochs": 500,
21         "steps_per_epoch": 200,
22         "learning_rate": 0.01,
23         "optimizer": "Adam"
24     },
25     "data": {
26         "numberFakeLines": 500,
27         "testSize": 0.1,
28         "augmentMidpoint": 1123,
29         "augmentRange": 1000,
30         "direction": "x",
31         "unitLoc": "px",
32         "unitTime": "1/100 Frames",
33         "limits": [0.388, 0.788, 0.0, 0.18]
34     }
35 }
```

**Listing 4.1:** Beispiel eines Hyperparameter Files in JSON

Example Hyperparameter File

- Architektur:
  - Dropout (wird eigentlich nicht benutzt weil es regression kaputt macht), relikt vergangener Zeiten
  - 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 ( = $\downarrow$  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
- Train:

verify

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

Suchen auf Simulierten Daten, weil da mehr Trainingbeispiele sind und Ergebnis auf realen Daten verifizieren.

Aktueller Stand: NextStep: kein Overfitting gefunden =<sub>c</sub> L1 und L2 Regularisierung haben keinen positiven Effekt Adam Optimizer ist am besten Learning Rate Decay ist eine gute Idee.

Separator: TODO

#### 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 4.1:** Architektur NN NextStep [TODO Quelle]

## 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 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 Next Step

Netz Variante 1: den nächsten Schritt vorhersagen

definitions vergleichsmodelle:

Notation aus Florian's Diss. Zustandsvector für CV.

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

Zustandsvector für CA:

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

### Constant Velocity Modell

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

### Contant Acceleration Modell

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

- CV, CA - Ergebnis Netz - Ergebnis Lineare Regression
- Zeit benötigt für evaluation?

### 5.3 Separator

- CV, CA, (AA?), IA - Ergebnis Netz - Ergebnis Lineare Regression - Ergebnis ausgleichsgerade? Maybe

## KAPITEL 6

# Fazit und Ausblick

## 6.1 Fazit

Mehr daten!

einführungstext  
in Fazit und  
Ausblick

blick darauf  
wie es gelaufen  
ist...

## 6.2 Ausblick

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

was man noch  
so machen  
könnte...



# Literatur

- [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.
- [SKP15] F. Schroff, D. Kalenichenko und J. Philbin, „Facenet: A Unified Embedding for Face Recognition and Clustering“, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, S. 815–823.
- [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+16a] 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.
- [WZY17] L. Wang, L. Zhang und Z. Yi, „Trajectory Predictor by Using Recurrent Neural Networks in Visual Tracking“, *IEEE Transactions on Cybernetics*, Jg. 47, Nr. 10, S. 3172–3183, Okt. 2017, ISSN: 2168-2267. DOI: [10.1109/TCYB.2017.2705345](https://doi.org/10.1109/TCYB.2017.2705345).
- [TSB17] K. Thormann, F. Sigges und M. Baum, „Learning an Object Tracker with a Random Forest and Simulated Measurements“, in *Proceedings of the 20th International Conference on Information Fusion (FUSION 2017)*, Xi'an, P.R. China, Juli 2017.
- [Pfa+16b] 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.
- [Fra] Fraunhofer IOSB, *TableSort Schüttgutsortierer*.

- [All14] Allied Vision Technologies GmbH, *User Manual for Bonito CL-400 200 Fps High Speed Camera*. Nov. 2014, Rev. I.