Julian Garske, Anna-Lena Grimberg, Moritz Klein und David Rabek

Matrikel Nr: 6728241, 3982838, 7755141 und 5563215

Dozent: Herr Benedikt Sondermann

Klimasteuerung der Zukunft

Inhaltsverzeichnis

[Abbildungsverzeichnis II](#_Toc5794809)

[Einleitung 1](#_Toc5794810)

[Motivation und Zielsetzung 1](#_Toc5794811)

[Abgrenzung der Arbeit 2](#_Toc5794812)

[Technologie und Werkzeuge 3](#_Toc5794813)

[Technlogien 3](#_Toc5794814)

[Werkzeuge 3](#_Toc5794815)

[Komponenten und Architektur 6](#_Toc5794816)

[Technische Bedienungsanleitung 7](#_Toc5794817)

[Probleme bei der Entwicklung 9](#_Toc5794818)

[Fazit 10](#_Toc5794819)

[Quellenverzeichnis 11](#_Toc5794820)

# Abbildungsverzeichnis

[Abbildung 1: Rahmen der Entwicklung 6](#_Toc5796681)

[Abbildung 2: Codeausschnitt "Skalieren der Daten" 6](#_Toc5796682)

[Abbildung 3: Codeausschnitt „Rückskalierung der Daten“ 7](#_Toc5796683)

[Abbildung 4: Codeausschnitt "Umrechnung in Uhrzeitformat" 7](#_Toc5796684)

[Abbildung 5: Eingabefenster 9](#_Toc5796685)

[Abbildung 6: Ausgabe des Ergebnis 10](#_Toc5796686)

# Einleitung

## Motivation und Zielsetzung

Im Automobilbereich ist die Digitalisierung mittlerweile nicht nur angekommen, sonders bereits weit fortgeschritten. Technologien wie Künstliche Intelligenz ermöglichen Verkehrsteilnehmern bereits teilautonomes Fahren und erhöhen somit die Sicherheit im Straßenverkehr. Genannte Technologien können jedoch auch über die Grenzen der Sicherheit hinaus Basis für Systeme sein, welche allen Fahrzeuginsassen einen Zusatz an Komfort bieten.

Statistiken belegen, dass immer mehr Fahrzeuge eine Standheizung bzw. Standlüftung besitzen. Gerade die Zunahme an Neuzulassungen von Elektroautos stützt diese These, da diese Fahrzeuge meist über eine solche Ausstattung verfügen. Elektrofahrzeuge, werden beim Parken meist geladen, weshalb das Einschalten einer Standheizung oder -lüftung nicht die Reichweite verkürzen würde, da die benötigte Energie direkt der örtlichen Stromquelle entnommen wird.

Fahrzeuge regeln bereits die Menge und Position der Luftzufuhr automatisch bei Eingabe der gewünschten Temperatur. Hierbei wird die Klimaanlage oder die Heizung, je nachdem ob gekühlt oder geheizt werden muss, eigenständig zugeschaltet. Diese Systeme werden als Klimaautomatik bezeichnet. Konventionelle Standheizungen müssen manuell eingeschaltet werden oder es muss manuell eine Zeit vorgegeben werden, wann die Standheizung einschalten soll.

Die Idee dieses Softwareprodukts, ist eine vollautomatische Steuerung der Standheizung und -lüftung eines Fahrzeugs. Hierzu soll eine Zeitreihenanalyse durchgeführt werden, um das Verhalten der Benutzer zu analysieren und ein Muster zu erkennen. Dieses Muster, stellt die Grundlage dar für die zukünftige Steuerung der Standheizung und -lüftung. Im Rahmen dieser Arbeit wird lediglich die Voraussage einer Uhrzeit zu der die Standheizung/ -lüftung eingeschaltet werden soll betrachtet. Ziel ist es, dass die Künstliche Intelligenz erkennt, wann der Benutzer die Standheizung einschaltet und dabei berücksichtigt, ob es sich um einen Werktag oder einen Tag am Wochenende handelt.

## Abgrenzung der Arbeit

Nicht im Rahmen dieser Arbeit enthalten ist die Berücksichtigung weiterer Faktoren beim Einschalten der Standheizung oder -lüftung. Die Einflussnahme solcher Faktoren, kann die Entscheidungen des Systems verbessern, lässt sich jedoch aufgrund begrenzter Ressourcen nicht realisieren. Faktoren, welche die Effizienz erhöhen würden, wären beispielsweise die Innen- und Außentemperatur des Fahrzeugs. Hierdurch wäre die Software in der Lage, die Standheizung bzw. Standlüftung auch einzuschalten, wenn die Temperaturen besonders hoch oder besonders niedrig sind. Der Fahrer und die Beifahrer würden das Fahrzeug so angenehm klimatisiert vorfinden, bei extremer Hitze im Sommer oder Kälte im Winter. Des Weiteren wäre ein automatisches Ausschalten realisierbar, wenn die Innentemperatur einen angenehmen Wert erreicht hat. Dadurch könnten Ressourcen wie Energie oder Kraftstoff geschont werden.

Ferner bietet die bereits beschriebe Lösung eine Erweiterung auf sämtliche Heiz- und Kühleinheiten eines Kraftfahrzeugs wie zum Beispiel:

* Eine Erweiterung der oben genannten Klimaautomatik. Das automatische Regulieren der Soll-Temperatur, die der Fahrer oder andere Fahrzeuginsassen konventionell selbst eingeben müssen, würde hierbei wegfallen.
* Selbstständiges ein- und ausschalten der Sitzbelüftung, Sitz-, Armlehnen-, Frontscheiben-, Heckscheiben- und Spiegelheizung.

Die dargelegten Erweiterungen bieten einen Ausblick auf die Zukunft, werden jedoch nicht im Rahmen dieses Projekts realisiert.

# Technologie und Werkzeuge

## Technlogien

Das dokumentierte Projekt basiert auf einem künstlichen neuronalen Netz. Künstliche neuronale Netze versuchen neuronale Netze von Menschen oder Tieren nachzubilden. Man versteht darunter eine Ansammlung an Neuronen, die in einer Beziehung zueinanderstehen, um eine bestimmte Funktion zu erfüllen. Die Ansammlung dieser Neuronen, Zellen oder auch Units – wie sie meist in künstlichen neuronalen Netzen bezeichnet werden – lassen die sogenannte Intelligenz zustande kommen. Um diese mit Daten anzulernen gibt es verschiedenen Ansätze.[[1]](#footnote-2)

Für dieses Projekt wurde das Long Short-Term Memory als geeignetstes Verfahren ausgewählt. Übersetzt bedeutet dies ein „langes Kurzzeitgedächtnis“. Dies ist eine Technik, um die Leistungsfähigkeit rekurrenter neuronaler Netze zu verbessern. Rekurrente Netze zeichnen sich dadurch aus, dass eine Rückkopplung von Neuronen einer Schicht zu anderen Neuronen erfolgt. Diese Netze eignen sich besonders, um zeitlich codierte Informationen in Daten zu entdecken.[[2]](#footnote-3)

Rekurrente neuronale Netze, die mit Long Short-Term Memorys arbeiten, sind in der Lage sich an Langzeit-Abhängigkeiten und frühere Erfahrungen zu erinnern.[[3]](#footnote-4) Aus diesem Grund eignet sich diese Technologie optimal, um eine Zeitreihenanalyse durchzuführen. Mithilfe dieser Analyse sollen Strukturen erkannt werden, um Voraussagen für die Zukunft treffen zu können. In diesem Projekt soll das Netz ein Muster erkennen, in welchem der Anwender die Standheizung einschaltet. Hierbei wird pro Tag eine Uhrzeit berücksichtigt.

## Werkzeuge

Die vorläufige Entwicklungsumgebung für das Projekt ist Google Colaboratory (kurz: Colab). Colab ist eine freie Jupyter Notebook Umgebung, welche mittlerweile von Google übernommen wurde. Die Daten und der Code werden in der Google Drive Cloud gespeichert und ausgeführt, wodurch der Programmcode ohne jegliche Installationen auf dem Computer getestet werden kann.

Einer Jupyter Notebook Dokument ist ein JSON Dokument in einem versionierten Schema. Notebooks sind strukturiert in mehrere Zellen, wobei in Code-Zellen und Text-Zellen zu unterscheiden ist. Text-Zellen bieten dem Entwickler die Möglichkeit direkt im Notebook zu dokumentieren, um den Code nachvollziehbarer zu gestalten. Die Code-Zellen enthalten ausführbaren Quellcode und bilden in Kombination die Softwarelösung. Die einzelnen Code-Zellen können schrittweise aufgeführt werden. Dies bietet den Vorteil, dass der Code nach und nach getestet werden kann und Fehler leichter zu erkennen und zu beheben sind. Colab bietet jedoch auch die Möglichkeit den Code in Gänze auszuführen, wenn das Notebook vollständig ist.

Da Algorithmen der Künstlichen Intelligenz und neuronaler Netze sehr umfangreich und komplex sind, würde das Entwickeln eines ausschließlich eigenen Codes den Rahmen dieses Projekts übertreffen. Tensorflow ist eine open-source Plattform für maschinelles Lernen. Es unterstützt Entwickler mit Bibliotheken, Werkzeugen und Kommunikationsplattformen. Die Machine Learning Modelle von Tensorflow können einfach in Cloud Umgebungen wie Colab eingebunden werden und eignen sich deshalb am besten für dieses Projekt. Tensorflow stellt Modelle in verschiedenen Sprachen bereit. Aufgrund der größten Bekanntheit wurde im Rahmen dieses Projekts Python ausgewählt, da hierzu am meisten Selbsthilfe im Internet verfügbar ist. Weitere Möglichkeiten sind JavaScript, Swift und mobile Plattformen wie iOS und Android.

Um den Nutzern eine offline ausführbare Applikation bereitzustellen, wurde der zuerst in Colab entwickelte Code in ein einfaches Python-Projekt übertragen und mit einer simplen GUI erweitert. Der Nachteil dabei ist, dass einige Frameworks und Bibliotheken zuvor manuell installiert werden müssen, um den Code ausführen zu können, und er nicht mehr blockweise getestet werden kann. Der Python-Code aus Colab kann ohne weitere Modifikationen lokal ausgeführt werden.

Tensorflow mit Python

Künstlich generierte Trainings- und Testdaten

# Komponenten und Architektur

Output

Neuronales Netz

Input



Abbildung 1: Rahmen der Entwicklung

Die Applikation ist so aufgebaut, dass als Input eine Datei mit Daten gefordert wird, die die Trainings- und Testdaten für das neuronale Netzwerk enthält. Dabei muss es sich um eine CSV-Datei mit Kommas als Trennzeichen handeln. Diese Datei muss eine Spalte mit der Überschrift „Startuhrzeit“ und darunter 100 Zeilen mit Zahlen zwischen 0 und 24 besitzen.

Als Output der Entwicklung folgt die Rückgabe der vom neuronalen Netzwerk berechneten Uhrzeit, zu der die Standheizung/ -lüftung eingeschaltet werden soll. Die Entscheidung, was eingeschaltet wird nicht von unserer Entwicklung getroffen.

Innerhalb des Programms erfolgt nun eine Zeitreihenanalyse. Der Programmablauf wird im Folgenden genauer erläutert.

#Skalieren der Daten, sodass sie zwischen 0 und 1 liegen

for i in range(len(data\_to\_use)):

data\_to\_use[i] = data\_to\_use[i] /24

print(data\_to\_use)

scaler = StandardScaler()

scaled\_data = data\_to\_use.reshape(-1,1)

Abbildung : Codeausschnitt "Skalieren der Daten"

In Abbildung 2 ist der Codeabschnitt zu sehen in dem die aus der CSV-Datei ausgelesenen Daten von 0 bis 1 skaliert werden. Mit dieser Skalierung kann das neuronale Netzwerk besser rechnen als mit Werten von 0 bis 24 Uhr. Zur Umrechnung werden die Werte des Arrays data\_to\_use durch 24 geteilt und so zu einem Wert zwischen 0 und 1 (z.B. bei 8.5 für 8:30 Uhr zu 0.3541667). data\_to\_use ist ein eindimensionales Array, das nun durch ein StandardScaler-Objekt scaler zu einem mehrdimensionalen Array umgewandelt wird.

#windowing the data with window\_data function

X, y = window\_data(scaled\_data, 7)

print(len(X))

print(len(y))

#X: 20 Arrays mit größe window\_size (80), die Daten enthalten, die die Tage 81, 82 etc. vorraussagen (Trainingsdaten) - mehr als 20 Arrays -> 93 Arrays

#y: Testdaten: Die 20 letzten werde, die als Vergleichswerte genutzt werden - mehr als 20 Arrays -> 93 Arrays

#print(X)

#print(y)

#we now split the data into training and test set

import numpy as np

X\_train = np.array(X[:99]) # 80: ab 80. Wert, :80 erste 80

y\_train = np.array(y[:99]) # Was ist X\_train und y\_train? bzw. X\_test und y\_test

# X\_train = 80 Arrays ; y\_train = 80 Werte

#print(X\_train)

#print (y\_train)

#PROBLEM: X und y müssen 100 lang sein, sodass die Train-Arrays 80 lang sind und die Test Arrays 20 lang. Jetzt durch die 73: umgangen.

# Die Test Arrays sind immer window\_size zu kurz (s. Funktion window\_data)

#73, damit die test-Arrays 20 lang sind

X\_test = np.array(X[92:]) # Warum sind diese Arrays leer? Wenn window\_size 7 sind sie gefüllt, window\_size ist die Anzahl der Werte, die uns komischerweise wegfällt

y\_test = np.array(y[92:])

#print(X\_test)

print("X\_train size: {}".format(X\_train.shape))

print("y\_train size: {}".format(y\_train.shape))

print("X\_test size: {}".format(X\_test.shape))

print("y\_test size: {}".format(y\_test.shape))

Abbildung : Codeausschnitt "Einteilung in Test- und Trainingsdaten"

In Abbildung 3 …

* Einteilung der Daten in Test und Trainingsdaten
* Training des neuronalen Netzwerks

#daten zurück auf 0 bis 24 formatieren

unscaled\_data = []

unscaled\_sup = []

unscaled\_test\_results = []

unscaled\_data = data\_to\_use \* 24

#die Umrechnung muss anders gemacht werden, da sup ein Tupel-Objekt ist

for i in sup:

unscaled\_sup.append(i\*24)

#Hier handelt es sich ebenfalls um ein Tupel-Objekt, das an den ersten 80 Stellen den Wert None hat. Wenn dieser vorhanden ist soll und kann nicht multipliziert werden, daher wird er einfach übernommen

for i in test\_results:

if i ==None:

unscaled\_test\_results.append(None)

else:

unscaled\_test\_results.append(i \* 24)

return(unscaled\_test\_results[99][0][0])

Abbildung : Codeausschnitt "Rückskalierung der Daten"

In Abbildung 3 ist der Code abgebildet, in dem die vom neuronalen Netzwerk berechneten Daten, die im Tupel-Objekt test\_results enthalten sind und Werte von 0 bis 1 besitzen, umgerechnet werden auf Werte von 0 bis 24, damit sie wieder unserem Anfangsformat entsprechen.

hour = int(result)

minutes\_string = str(result).split(".")

minutes = int(float( "0." + minutes\_string[1][0:2])\*60)

time = "Die Standheizung soll um " + str(hour) + ":" + str(minutes) + " Uhr gestartet werden."

popupmsg(time)

Abbildung : Codeausschnitt "Umrechnung in Uhrzeitformat"

Im Codeausschnitt in Abbildung 4 wird der Wert den das neuronale Netzwerk für den 100. Tag berechnet hat (result) in ein Uhrzeitformat formatiert. Hierzu wird in hour als Wert für die Stunde der int-Wert von result genommen. Hierbei wird nicht gerundet. result wird als String formatiert und dann an der Stelle des Kommas aufgeteilt in zwei Strings, die in Form eines String-Arrays in minutes\_string gespeichert werden. minutes\_string[1] enthält nun die Nachkommastellen des Ergebnis des neuronalen Netzwerk für den 100. Tag als String-Objekt. Dieses wird nun mit einem String „0.“ zusammengesetzt, damit es sich nicht um ganzzahlige Werte, sondern um Nachkommastellen handelt und anschließend mit 60 multipliziert. Das Ergebnis wird in minutes als int-Wert gespeichert.

Außerdem wird die Meldung zusammengebaut, die in unserem Ergebnisfeld angezeigt werden soll.

# Technische Bedienungsanleitung

Nun folgt eine Beschreibung, welche Schritte notwendig sind, damit der Code ausgeführt werden kann.

1. Die Phyton-Datei *zeitreihe.py* und die Datei *Datenbank.csv* müssen lokal abgelegt werden.
2. Phyton muss in der Version 3.4, 3.5 oder 3.6 installiert sein
3. Pip muss installiert sein
4. Die PATH-Variablen von Windows müssen angepasst werden, sodass die Befehle python und pip in der Konsole verfügbar sind
5. Ausführen der folgenden Befehle in der Konsole:
   * pip install tensorflow
   * pip install numpy
   * pip install pandas
   * pip install matplotlib
   * pip install sklearn
6. Zum Ausführen der Datei in der Konsole in das Verzeichnis navigieren in der die Phyton-Datei liegt und den Befehl python zeitreihe.pyausführen

Anschließend sollte sich ein Fenster öffnen, das aussieht wie das Fenster in Abbildung 2.

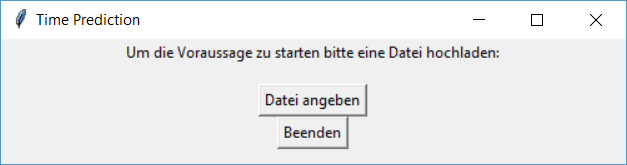


Abbildung 6: Eingabefenster

Durch betätigen des „Datei angeben“-Button öffnet sich der Dateiexplorer. Nun muss zum Verzeichnis der Datei *Datenbank.csv* navigiert werden und diese ausgewählt werden. Alternativ können andere CSV-Dateien mit dem in Kapitel „Komponenten und Architektur“ beschriebenen Format verwendet werden.

Daraufhin startet das Training des neuronalen Netzwerks gefolgt von der Berechnung der Uhrzeit. Dies kann abhängig von der Hardware des Computers eine Weile dauern, währenddessen keine Interaktion möglich ist.

Sobald die Berechnung erfolgt ist, öffnet sich ein weiteres Fenster, in dem die Uhrzeit ausgegeben wird, die berechnet wurde (siehe Abbildung 3).

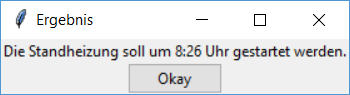


Abbildung 7: Ausgabe des Ergebnis

# Probleme bei der Entwicklung

Zunächst gab es die Idee, Tensorflow über Javascript in einer Webanwendung zu nutzen. Dafür existiert seit 2018 die Bibliothek Tensorflow.js. Nach etwas Einarbeitungszeit stellte sich jedoch heraus, dass diese Herangehensweise zu viel Aufwand darstellen würde. Keins der Gruppenmitglieder hat mit der Bibliothek Erfahrung und es wäre sehr umständlich, sich das komplette Wissen anzueignen und das gesamte neuronale Netz von Grund auf zu programmieren.

Stattdessen wurde beschlossen, das Projekt an ein bestehendes Beispiel anzulehnen, um daraus einige Teile übernehmen zu können. Dabei handelt es sich um ein Colab-Projekt, welches mittels eines neuronalen Netzes die Preisentwicklung des Bitcoins versucht zu prognostizieren. Deshalb wurde der Programmcode zunächst auch in Colab geschrieben. Das Prinzip bei dem Beispiel ist das gleiche, wie bei diesem Anwendungsfall: Auf Basis einer Zeitreihe soll die Zukunft vorhergesagt werden. Um es hierauf anzupassen musste der Code aber zunächst verstanden und anschließend noch eine Menge daran verändert werden, sodass auch dafür ein recht hoher Aufwand benötigt worden ist.

Nachdem die Applikation in Colab ausgeführt werden konnte, wurde der Code, um dem Anwendungsfall gerecht zu werden, in einer Python-Datei abgelegt. Die Nutzer der Applikation möchten in einer GUI ihre Daten eingeben und ein Ergebnis zurückerhalten, anstatt die einzelnen Codeabschnitte auf einer Webseite auszuführen. Deshalb war für diesen Prototypen die Entwicklung eines Frontends notwendig. In einer finalen Version könnte der Code sogar im Auto ausgeführt werden, indem das Auto die Daten für den Input selbst erfasst und abhängig davon die Standheizung startet oder es z. B. dem Fahrer in einer App vorschlägt, sie zu der Zeit zu starten. Die Einbindung des Programmcodes in ein simples Frontend stellte nur wenig Aufwand dar.

# Fazit

# Quellenverzeichnis

Luber, Stefan: Was ist ein Long Short-Term Memory? Verfügbar über: <https://www.bigdata-insider.de/was-ist-ein-long-short-term-memory-a-774848/> Letzter Zugriff: 22.03.2019

Neuronales Netz: Rekurrente Netze. Verfügbar über: <http://www.neuronalesnetz.de/rekurrente.html> Letzter Zugriff: 22.03.2019

Tiedemann, Michaela: Wie Künstliche Neuronale Netze die Erfolge des Maschinellen Lernen ermöglichen. Verfügbar über: <https://www.alexanderthamm.com/de/artikel/wie-kuenstliche-neuronale-netze-die-erfolge-des-maschinellen-lernen-ermoeglichen/> Letzter Zugriff: 22.03.2019

1. Tiedemann, Michaela: Wie Künstliche Neuronale Netze die Erfolge des Maschinellen Lernen ermöglichen. [↑](#footnote-ref-2)
2. Neuronales Netz: Rekurrente Netze. [↑](#footnote-ref-3)
3. Luber, Stefan: Was ist ein Long Short-Term Memory? [↑](#footnote-ref-4)