

Electronic
CarEFuel
Mobile Efficient

LEIBNIZ UNIVERSITÄT HANNOVER

Nommensen, Nils
Kriegel, Josef
Wallat, Jonas
Gritz, Wolfgang

21. Januar 2018

Inhaltsverzeichnis

1	Motivation und Einleitung	1
2	Umsetzung	2
2.1	Carefuel Basic	2
2.2	Carefuel Extended	2
2.3	Dynamic-Path Algorithmus	4
2.4	Datenbankschema	5
2.5	Preisvorhersage	6
2.5.1	Einleitung	7
2.5.2	Architektur	8
2.5.3	Training	10
3	Evaluation	11
3.1	Fixed Path Algorithmus	11
3.2	Preisvorhersage	11
4	Future Work und Bekannte Fehler	12
5	Benutzerhandbuch	14
5.1	Voraussetzungen	14
5.2	Installation	14
5.3	Ausführung	15
5.3.1	Carefuel Basic	15
5.3.2	Carefuel Extended	16
5.4	Verwendung	17
5.5	FAQ	18
	Abbildungen	19
	Literatur	19
6	Literaturverzeichnis	19

1 Motivation und Einleitung

Ziel dieses Projekts ist es zum einen, die Benzinpreise von Tankstellen auf Grundlage von historischen Daten vorherzusagen und zum anderen eine möglichst günstige Route von einem Startpunkt zu einem Endpunkt zu finden, auf der man an beliebigen Tankstellen seinen Tank auffüllen kann. Die Grundaufgabenstellung sieht vor, dass man den Fixed-Path-Algorithmus implementiert und dazu eine Methode findet, um die Benzinpreise vorherzusagen. Des Weiteren sollen die Ergebnisse der Grundaufgabenstellung um weitere kreative und sinnvolle Ideen ergänzt werden.

Wir haben unser Projekt in drei separate Unterprojekte unterteilt:

- Carefuel Basic: Im Rahmen dieses Teil-Projekts haben wir die Grundaufgabenstellung gelöst. Siehe Kapitel 2.1.
- Carefuel Extended: In diesem Teil des Projekts haben wir einen Algorithmus implementiert, um das allgemeine Gasstation-Problem zu lösen. Dazu haben wir einen Webserver erstellt, um dynamisch neue Routen in Echtzeit auswählen und berechnen zu können. Siehe Kapitel 2.4.
- Preisvorhersage: Der dritte Teil des Projekts befasste sich mit der Suche nach einer Methode, die es ermöglicht, Preise, zu einer Tankstelle, für bis zu einen Monat in die Zukunft, vorherzusagen. Zur Lösung dieses Problems haben wir ein Recurrent Neural Network (RNN) verwendet. Siehe Kapitel 2.5.

Dieses Projekt hat es uns ermöglicht, die während des Studiums doch zumeist nur theoretischen Konzepte praktisch anzuwenden. Wir konnten unser Wissen in verschiedensten Bereichen der Informatik, wie zum Beispiel Web-Technologien, Data-Science und Machine Learning anwenden und erweitern. Neben der gewonnenen Erfahrung ist natürlich auch die Aussicht auf ein Preisgeld und der Kontakt mit potenziellen zukünftigen Arbeitgebern eine hohe Motivation für das Projekt gewesen.

2 Umsetzung

2.1 Carefuel Basic

Das Teilprojekt Carefuel Basic erfüllt die Grundanforderungen des InformatiCups. Dazu gehören die Vorhersage von Benzinpreisen, die Berechnung der günstigsten Route, sowie die Ausgabe beider in Form von Plaintext-Dateien. Das Ziel dieses Teilprojektes war es, diese Anforderungen mit möglichst geringem Aufwand zu erreichen, um, noch im gegebenen Zeitrahmen, mit dem anderen Teilprojekt Carefuel Extended eine interessantere Bearbeitung des Anwendungsgebietes zu schaffen.

Für Carefuel Basic wurde deshalb eine minimalistische Java-Applikation entwickelt, die sich in drei Teile unterteilt: Erstens, dem CSV-Parser, der die Routen bzw. Tankstellen-Listen parst und die Informationen für das restliche Tool zur Verfügung stellt. Zweitens, dem PricePredictor, der die Informationen zu Tankstellen um den vorhergesagten Preis erweitert (Informationen dazu wie die Preise vorhergesagt werden findet sich im Kapitel Preisvorhersage). Zuletzt der Pathfinder, der aus den gegebenen Informationen eine Route und die optimale Tankstrategie berechnet. Die Tankstrategie ergibt sich aus dem Paper "To fill or not to fill" [3]. Ebenfalls wird die berechnete Tankstrategie einer naviven Strategie gegenübergestellt. Mehr Informationen dazu in dem Evaluations-Kapitel 3.1.

Für Preis- und Tankstellen-Informationen greift das Basic Projekt auf die im Git-Repository des InformatiCups gegebenen CSV-Dateien zurück. Die notwendige Interaktion mit dem Anwender geschieht auf der Konsole (siehe Kapitel Ausführung).

2.2 Carefuel Extended

Unser Ziel im Extended Projekt war es, eine Anwendung zu entwickeln, die es ermöglicht möglichst kostengünstige Routen in Echtzeit zu berechnen und über eine Webseite zur Verfügung stellen. Wir haben uns dafür die Software Architektur in Abbildung 13 überlegt.

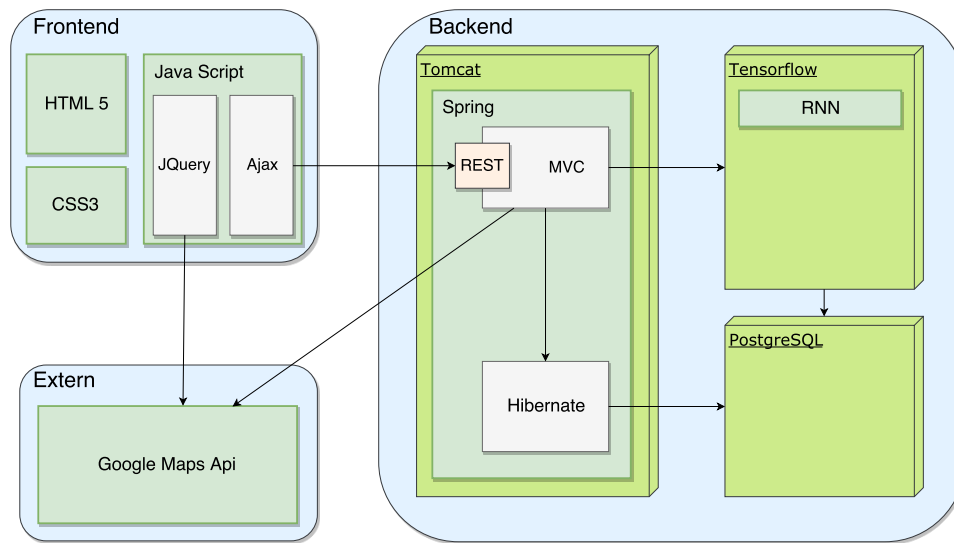


Abbildung 1: Architektur

Im Frontend soll eine responsive Website entstehen. Dadurch ist das Programm plattformunabhängig und ohne Installation von Zusatzsoftware ausführbar. Da sich das ganze Projekt um die Suche nach einer optimierten Route dreht, wird die Google Maps API benutzt, um die Route auf einer Karte mit echten Wegen darzustellen. Zur Kommunikation mit dem Backend werden Ajax-Requests an die REST-Schnittstelle unseres Servers geschickt, wodurch sich das Frontend sehr gut vom Backend abkapseln ließ.

Das Backend ist dann selber auch nochmal unterteilt.

Das RNN wurde vollständig in Python mithilfe der Open Source Bibliothek Tensorflow implementiert. Weitere Informationen dazu gibt es separat in Kapitel 2.5.

Zur persistenten Datenhaltung eignet sich eine relationale PostgreSQL-Datenbank. Weitere Informationen dazu gibt es separat in Kapitel 2.4.

Den Kern des Programms bildet eine Java-Applikation, die mithilfe des Spring MVC-Frameworks eine REST-Schnittstelle bietet. Diese stellt den Kommunikationskanal zwischen Front- und Backend her. Außerdem ist hier der Algorithmus implementiert, siehe 2.3, der für die Routenberechnung vorgenommen wird.

Um auf das RNN zuzugreifen, wird die Java-API der Tensorflow Bibliothek verwendet.

Zur Kommunikation der Java-Applikation mit der Datenbank wird die Java Persistence API und Hibernate eingesetzt.

Um Abhängigkeiten zu externen Open Source Libraries verwalten und die Auslieferung des gesamten Programms weitestgehend automatisiert zu realisieren, wird Maven eingesetzt. So muss das gebaute Release lediglich auf den Server kopiert und gestartet werden.

2.3 Dynamic-Path Algorithmus

Wie in Abschnitt Planung 2.1 bereits erläutert, haben wir entschieden, dass für die Wegfindung direkt Vorhersagen von Preisen mit einbezogen werden sollen. Um in einem solch großen Graphen mit über 15.000 Knoten eine effiziente Laufzeit zu erreichen, wird eine erweiterte Implementierung des A* Algorithmus verwendet. Der sogenannte “Explorative A*” (EA) Algorithmus.

Die Idee dazu beruht mit auf dem grundlegenden Paper zum Gas Station Problem. Mit den Einschränkungen, dass man für jede Tankstelle die Nachbarschaft über die maximale Reichweite gegeben durch die Tank-Kapazität und den durchschnittlichen Verbrauch berechnen kann, lässt sich ein Graph erzeugen, welcher deutlich weniger Kanten aufweist, als ein vollständiger Graph. Um realistische Preise für die angefahrenen Tankstellen zu berechnen, muss also eine durchschnittliche Fahrtdauer zwischen der aktuellen und der nächsten Tankstelle berechnet werden. Um den Algorithmus nicht nur den günstigsten oder den kürzesten Weg finden zu lassen, wird außerdem ein Faktor x mit eingebracht. Sollte also eine Strecke von Tankstelle A nach B beispielsweise 50 km lang sein und der Benzinpreis bei B 1,21 €/l betragen, so kann über folgende Kostenberechnung ein ausgewogener Pfad berechnet werden:

$$cost(edge_{A \rightarrow B}) = dist_{edge_{A \rightarrow B}} * (1 + (cost_{gas}(B) - 1) * x))$$

Die Kosten einer Kante zwischen zwei Tankstellen geht mit der Distanz multipliziert mit einem Wert zwischen dem eigentlichen Benzinpreis und 1 in den EA ein. Das bedeutet, dass bei $x=0$ der kürzeste Weg und bei $x=1$ der günstigste Weg gefunden wird. Aber wie auch aktuell in vielen Navigationssystemen vertreten bietet sich oftmals eine Strecke zwischen dem günstigsten und kürzesten Weg an. In vielen Fällen lässt sich statt 100% Kosten- zu 0% Streckeneffizienz und andersherum ein Verhältnis von 75% Kosten- und 75% Streckeneffizienz finden.

Algorithmus 1 : explorativeAStar(start, end, start_time, MAX_RANGE, AVG_SPEED, x)

```

1 PriorityQueue open = new PriorityQueue()
2 List closed = new List()
3 open.add(< start, start_time >)
4 while open not empty do
5     PriorityQueueElement p = open.firstElement()
6     Node current = p.Node
7     Date cur_time = p.Time
8     if current = end then
9         | return FoundPath
10    end
11    foreach node in neighbours of current do
12        | if node in closed then
13            | | continue
14        | end
15        | gCost = current.gCost + cost(Edge(current, node), x)
16        | if node in open and gCost > node.gCost then
17            | | continue
18        | end
19        | node.predecessor = current
20        | node.gCost = gCost
21        | node.hCost = node.gCost + dist(node, end)
22        | if node not in open then
23            | | open.add(node)
24        | end
25    end
26 end

```

Abbildung 2: Explorative A*“(EA) Algorithmus

Nach diesem Algorithmus lässt sich der Pfad über die Predecessor von End-Tankstelle bis Start-Tankstelle ermitteln, außer, es gibt keinen Weg. Letzteres ist natürlich möglich, wenn man beispielsweise eine Kapazität von 0 Litern übergibt.

2.4 Datenbankschema

Für die persistente Datenhaltung wird eine Postgres Datenbank verwendet. Das bietet den Vorteil, dass die historischen Preisdaten, die Tankerkönig als PostgreSQL Dump-Files bereitstellt, direkt vom System eingelesen werden können. Neue Daten können so auch in einem nächtlichen Cronjob auf dem System eingelesen werden. Dazu wird das neue Dump-File von Tankerkönig

bezogen und ein kurzes Python-Skript löscht alle Einträge vor dem letzten Import-Datum. So werden wirklich nur neue Daten eingelesen.

Das Design der Datenbank ist in 3 dargestellt.

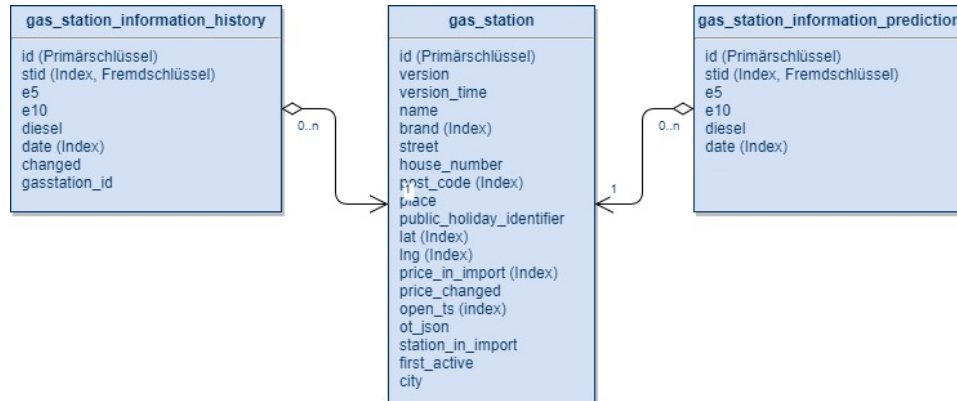


Abbildung 3: Architektur

Das Design, das Tankerkönig mit den Dump-Files vornimmt, wird vollständig erhalten und lediglich ein Index auf *gas_station_information_history.date* angelegt. Zusätzlich wird eine Tabelle für die Vorhersagen namens *gas_station_information_prediction*, analog zu den historischen Preisen angelegt. Diese dient zum Zwischenspeichern der Ergebnisse des RNNs.

2.5 Preisvorhersage

Die Benzinpreise einer Tankstelle zu einem gegebenen Zeitpunkt möglichst korrekt vorherzusagen ist nicht trivial und erfordert viel Aufwand. Wir haben uns für die Verwendung eines Recurrent Neural Networks (RNNs) entschieden, das sogenannte Long-Short-Term-Memory (LSTM) Zellen enthält. Diese Entscheidung beruht auf zwei Gründen. Erstens wurde in den letzten Jahren in mehreren Forschungsergebnissen herausgefunden, dass Recurrent Neural Networks äußerst gute Ergebnisse bei der Zeitreihenanalyse liefern können, siehe dafür zum Beispiel [1] und [4]. Desweiteren haben Neuronale Netze in den letzten Jahren einen enormen Aufschwung erlebt und auch in vielen anderen Bereichen, wie zum Beispiel Computer Vision und Language Processing, für rekordverdächtige Durchbrüche gesorgt. Da zu erwarten ist, dass sich der Trend auch in Zukunft fortsetzt und es generell ein äußerst spannendes Thema ist, haben wir dieses Projekt genutzt, um Erfahrung mit dieser Thematik zu sammeln.

2.5.1 Einleitung

In den einfachsten neuronalen Netzen die heutzutage verwendet werden, sogenannten Feed-Forward-Netzen, wird der Input des Netzes über eine oder mehrere hidden Layer zum Output transformiert. Sämtliche Neuronen der aktuellen Layer x werden mit den Neuronen der nächsten Layer y über die Beziehung $y_i = \sum_{i=1}^N w_i * x_i$ verknüpft, wobei w_i die antrainierten Gewichte sind. Man spricht auch von Fully Connected Layer, siehe Abbildung 4.

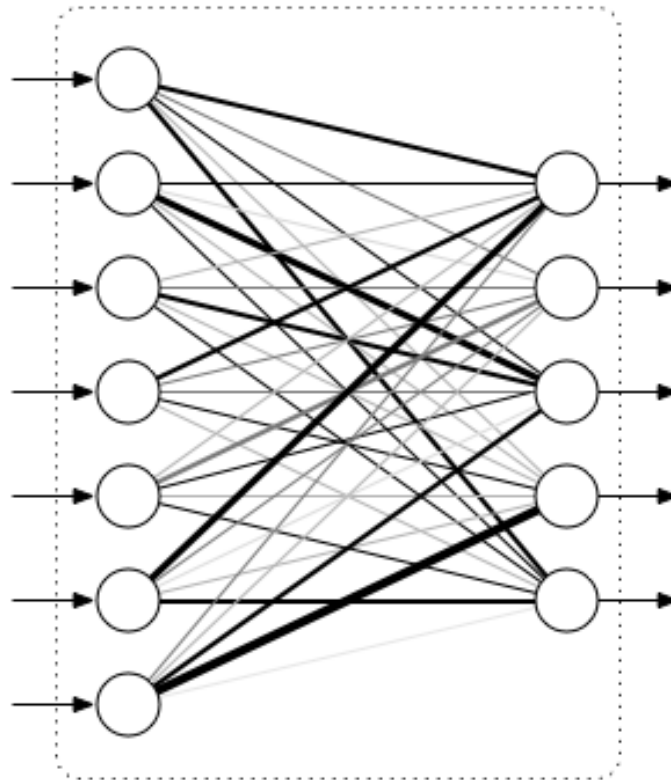


Abbildung 4: Architektur (Quelle: <https://brilliant.org/wiki/feedforward-neural-networks>)

Das Problem solcher Netze ist, dass die Anzahl der einzelnen Neuronen pro Layer festgesetzt ist und nach dem Training nicht mehr geändert werden kann. Eine Möglichkeit mit diesem Problem umzugehen ist die Verwendung von Recurrent Neural Networks. Diese erlauben es Netzen, grob gesprochen, sich Daten, die sie in vorherigen Aufrufen schon einmal gesehen haben, zu merken. Sie haben also eine Art internen Zustand. In Abbildung 5 ist schematisch der Ablauf eines RNNs dargestellt. Der Ausgang h_t eines Netzes A hängt nicht nur von dem direkten Eingang x_t ab, sondern auch von den vorherigen Eingängen x_{t-1} , deren Informationen sich das Netzwerk über einen

internen Zustand merkt.

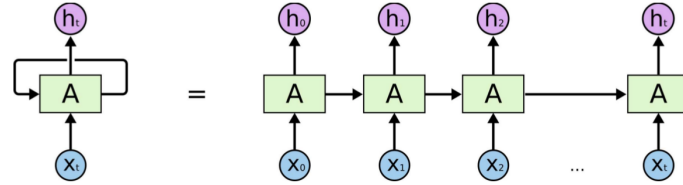


Abbildung 5: Funktionsweise von Recurrent Neural Networks (Quelle: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>)

In den letzten Jahren gab es viele Bemühungen RNNs umzusetzen und es wurden viele Architekturen vorgeschlagen. Zu den Vorreitern gehört die oben erwähnte Long-Short-Term-Memory (LSTM) Architektur von [2], die in Abbildung 6 dargestellt ist.

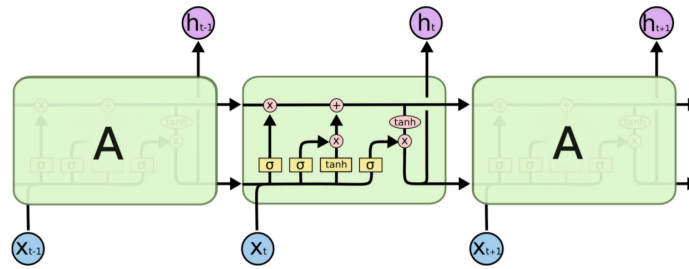


Abbildung 6: Long Short Term Memory Architektur (Quelle: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>)

Durch die Verwendung von speziellen Gates wird in jedem Schritt entschieden, welche Informationen beibehalten werden, welche neu hinzugefügt werden und wie letztendlich der Ausgang aussieht. Für eine ausführliche Erklärung der Funktionsweise von LSTMs ist für den interessierten Leser die Webseite <http://colah.github.io/posts/2015-08-Understanding-LSTMs/> sehr zu empfehlen.

2.5.2 Architektur

Um die historischen Preisdaten beliebiger Tankstellen zu verwenden, muss unser Netzwerk in der Lage sein Daten verschiedenen Umfangs zu verarbeiten. Wie bereits oben erwähnt, sind normale neuronale Netze dazu nicht in der Lage, weswegen wir uns die in 8 dargestellte Architektur überlegt haben, die auf der Verwendung von LSTM Zellen beruht. Zur Verwendung müssen die historischen Preisdaten in Intervalle x_i fester Größe aufgeteilt werden, wie in Abbildung 7 angedeutet.

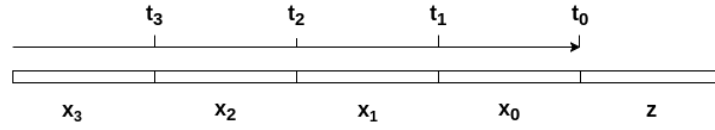


Abbildung 7: Einteilung der historischen Preisdaten in feste Intervalle

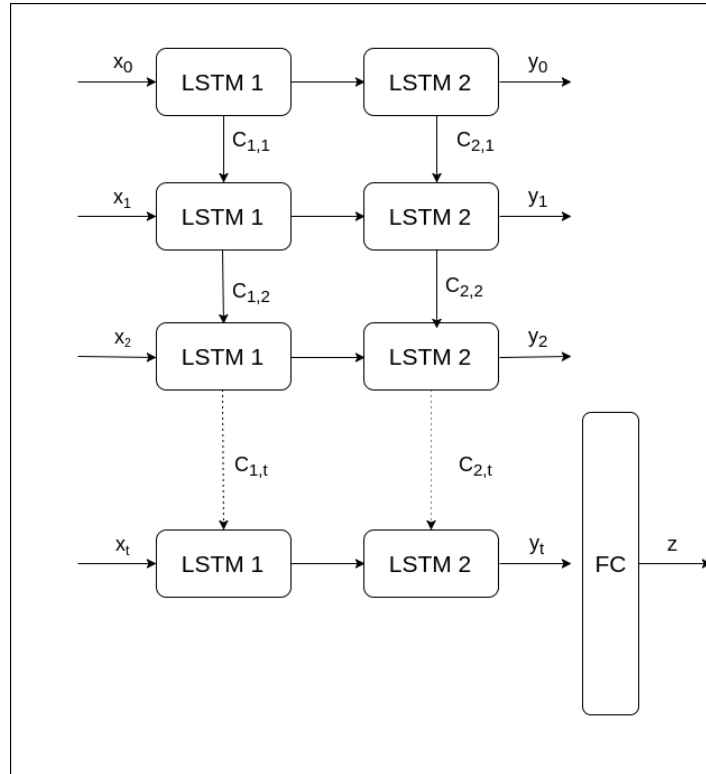


Abbildung 8: Netzwerk Architektur

Das Ziel ist es, die Preise z des nächsten Monats ab dem Zeitpunkt t_0 vorherzusagen. Dazu werden die historischen Preise in feste Intervalle x_i von t_i bis t_{i+1} eingeteilt. Zur Verwendung ist es dann essentiell, dass für jeden Monat gleich viele Preisdaten vorhanden sind. Da die Intervalle der Einträge jedoch bei allen Tankstellen noch variieren und auch nicht immer gleich viele Einträge gemacht werden, werden die Daten vorher noch mit einem linearem Spline interpoliert und an äquidistanten Zeitpunkten ausgewertet. Da, wie in Abbildung 9 zu sehen ist, die durchschnittliche Anzahl an Preismeldungen aller Tankstellen nicht größer als 12 ist, haben wir uns dafür entschieden, die Daten alle zwei Stunden zu interpolieren. Dadurch kommt man dann für jeden Monat x_i auf $31 * 12 = 372$ Einträge.

Ein weiterer Vorverarbeitungsschritt ist noch die Normalisierung aller

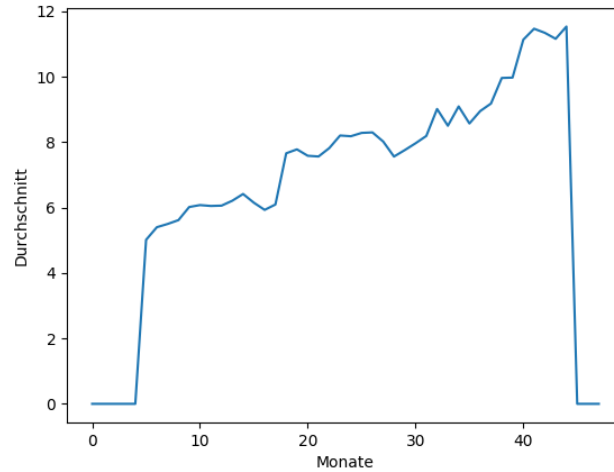


Abbildung 9: Durchschnittliche Anzahl von gemeldeten Preisen pro Tag über die letzten Monate

Preise auf einen Bereich zwischen 0 und 1, mit dem die einzelnen Elemente des Netzwerks besser arbeiten können. Der grundlegende Ablauf ist dann folgender: Alle Monate x_i werden nacheinander in das Netzwerk, bestehend aus zwei LSTM Zellen, gegeben. In jedem Schritt erhalten diese Zellen dann einen neuen Zustand, der die erhaltenen Informationen speichert und erweitert. Nachdem dann der letzte Monat ins Netzwerk gegeben ist, wird der letzte Output y_t noch einmal durch einen Fully-Connected Layer gegeben, in der die erhaltenen Informationen ein letztes mal miteinander verknüpft und zu den letztendlichen Preisen z berechnet werden, die nur noch wieder zurück skaliert werden müssen.

2.5.3 Training

Zum Training des Netzes haben wir alle historischen Preisdaten von Tankerkönig verwendet, die unter <https://creativecommons.tankerkoenig.de/> in Form eines Datenbank Dumps bezogen werden können. Bevor wir diese Daten verwenden konnten, mussten wir Sie noch aufbereiten. Tankstellen, die weniger als einen Monat an Daten bereitgestellt haben und Einträge, die ungültige Werte beinhalten, wurden entfernt. Danach hatten wir für etwa 15000 Tankstellen durchschnittlich etwa 10000 Einträge pro Kraftstoff zur Verfügung. Diese haben wir dann noch in Trainings-, Validierungs- und Testdaten im Verhältnis von 80, 10, 10 aufgeteilt und dann für jeden Kraftstoff ein eigenes Netz trainiert. Zur Vermeidung von Overfitting haben wir hinter jeder LSTM Zelle und Fully-Connected Layer während des Trainings noch ein Dropout Layer mit einer Dropout Wahrscheinlichkeit von 0.5 hin-

zugefügt. Als Loss-Funktion haben wir den Mean-Squared-Error verwendet, der mit dem State of the Art Adam Optimierer minimiert wird. Eine Übersicht über alle Hyperparameter liefert die Tabelle in Abbildung 1.

Name	Wert
Einträge pro Monat	372
Batch Größe	25
Anzahl Epochen	1000
Lern-Rate	0.0001

Tabelle 1: Übersicht der genutzten Hyperparameter pro Netz

3 Evaluation

3.1 Fixed Path Algorithmus

Im Teilprojekt Carefuel Basic wird mit Hilfe des Fixed Path Algorithmus ([3]) die optimale Tankstrategie berechnet, um am günstigsten vom Start entlang des Pfades zum Ziel zu kommen. Dabei wird mit einem leeren Tank gestartet und es soll mit einem leeren Tank am Ziel angekommen werden. Zur Evaluation der Tankstrategie berechnen wir zusätzlich zu der von uns empfohlenen eine naive Tankstrategie und geben den Preis zum Vergleich an. Die von uns gewählte naive Strategie beinhaltet es, an jeder Tankstelle nur so viel zu tanken, wie für den Weg zur nächsten nötig ist. Die Ausgabe ist in Abbildung 10 zu sehen.

```
Price of the route is: 162,39 Eur
Naive Price (if you fill just enough gas to get to next gas station): 162,72 Eur
```

Abbildung 10: Preise der Tank-Strategien

Der relativ geringe Unterschied je nach Tankstrategie ist wohl darin begründet, dass das neuronale Netz sehr gut vorhersagen kann, ob die Preise sinken oder steigen allerdings in der Prognose wie stark das passieren wird etwas zu konservativ schätzt (siehe Kapitel 3.2). Aktuell schwanken die Preise der meisten Tankstellen nur um ca. 0.6 Cent. Dementsprechend klein ist der die Preis-Differenz wenn wir an den günstigsten Tankstellen tanken oder nicht. Da der Preisunterschied bei vielen Tankstellen in der Nähe meist gering ist, bietet sich Carefuel vermutlich für längere Strecken eher an.

3.2 Preisvorhersage

Zur Evaluation der Preisvorhersage des neuronalen Netzes haben wir die historischen Preisdaten verwendet. Für jede Tankstelle wird der letzte Monat

der bekannten Preise abgeschnitten und als Vergleichsdaten verwendet, die das Netzwerk vorhersagen muss. Insgesamt haben sich so auf den Testdaten ein durchschnittlicher Fehler von 3.8 Cent für unsere Implementation des Netzes ergeben. In Abbildung 11 ist einmal der Preisverlauf einer Tankstelle über einen Tag gegenüber der Vorhersage unseres Netzes dargestellt.

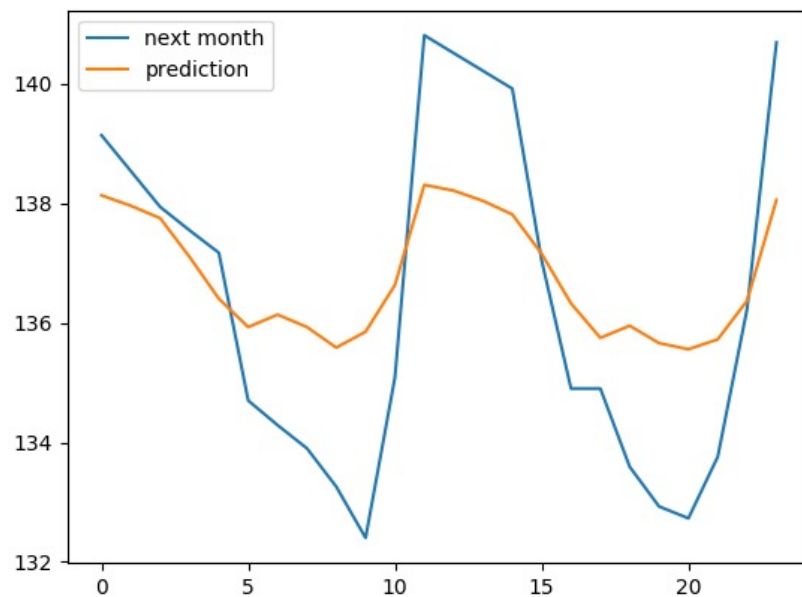


Abbildung 11: Preisverlauf einer beispielhaften Tankstelle über einen Tag

Man sieht hier relativ deutlich, wie das Netz in der Lage ist die Tendenz der Preisentwicklung nachzuvollziehen und es dem tatsächlichen Preis über die Zeit relativ genau folgt. Auch wenn das Netz nicht die genauen Preisausschläge vorhersagen kann geht es doch schon in die richtige Richtung.

4 Future Work und Bekannte Fehler

Future Work

In der Folge werden Punkte aufgelistet, die vom Team in der Projektzeit nicht umgesetzt werden konnten, aber als sinnvolle Erweiterungen aufgefasst wurden.

- Der verwendete Server, den die Universität Hannover im Rahmen dieser Veranstaltung bereitgestellt hat, bietet leider nicht die notwendigen Ressourcen, die für einen produktiven Einsatz erforderlich wären.

Selbst zur Entwicklungszeit mussten häufige Workarounds gefunden werden, wie z.B. das Einlesen der Preis-Vorhersagen in die Datenbank. Während dies lokal auf aktuellen Laptops in unter zwei Stunden möglich war, hätte der Server nach Berechnung etwa 270 Stunden benötigt. Auch die Abfragen benötigen längere Abfragezeiten, da Parallelisierungen auf dem Single-Core-System nicht ausgenutzt werden können. Der damit wichtigste Punkt wäre eine Portierung auf eine leistungsfähigere Umgebung.

- Benutzereingabe zulassen, wie viel Kraftstoff am Zielort vorhanden sein soll.
- Distanzen sollen über die Google Maps API bezogen werden, anstatt dass die Luftlinie verwendet wird.
- Usability verbessern
 - Profile für Benutzer anbieten, um zum Beispiel vergangene Routen zu speichern.
 - Wegbeschreibung verfeinern (eventuell als Link zum Handy senden für GPS/Navi)
- Die Öffnungszeiten der Tankstellen sollen mit in den Algorithmus eingebunden werden. Aktuell werden diese noch ignoriert.
- Bei der Suche nach dem günstigsten Pfad werden momentan Tankstellen, die sich in unmittelbarer Nähe zueinander befinden, nicht geclustert. Das führt dazu, dass der Algorithmus jede als günstigsten Weg in Betracht zieht, was die Anzahl an Rechenschritten drastisch erhöht und so eine hohe Abfragedauer zur Folge hat.
- Aktuell werden neue hinzukommende Tankstellen bis sie für 30 Tage Daten besitzen ignoriert. Um eine Fehlerquelle bei den Predictions auszuschließen, werden diese GasStations aus der Datenbank entfernt. Wenn das update Script für die DB ausgeführt wird, kann er die GasStations in der DB nicht aktualisieren, sodass die neueren Tankstellen nie mit eingebaut werden können.

Bekannte Fehler

In der Folge werden bekannte Fehler aufgelistet, die zur Projektzeit nicht mehr gelöst werden konnten und erst in zukünftigen Releases behoben werden würden.

- Der Benutzer muss bei der Eingabe von Start- und Zielort die Orte direkt auswählen und es werden dort keine fehlerhaften Eingaben abgefangen.

- Die Start- und Zieltankstelle werden radial vom Start- und Zielort über die Tankerkönig-API ermittelt. Der Datenbestand von Tankerkönig muss aber nicht mit unserer Datenbank übereinstimmen, da wir die Daten bereinigen. Es kann deshalb vorkommen, dass keine passende Tankstelle in der Datenbank zur Anfrage gefunden werden kann. Dieser Fehler liefert auch keine eigene Fehlermeldung, sondern fällt lediglich in der Browser-Konsole auf.
- Bei Eingabe konkreter Adressen über das Frontend kommt es teilweise zu leichten Verschiebungen. Gibt man beispielsweise eine Hausnummer 14 an ist es möglich das in den Routen-Informationen die Route bei Hausnummer 10 etc. startet. Ein Grund könnte sein, dass die Lon/Lat-Werte die Google-Maps für die Eingabe des Nutzers liefert in der Datenbank oder vom Programm nicht mit notwendiger Präzision gespeichert werden und es so zu minimalen Abweichungen kommt.

5 Benutzerhandbuch

5.1 Voraussetzungen

Um die Software zu starten werden folgende minimale Voraussetzungen benötigt:

- Debian 8.9 oder aktueller
- >25GB freier SSD-Festplattenspeicher
- >16GB Arbeitsspeicher
- PostgreSQL 9.4 oder aktueller
- Java 1.8.0_101 oder neuer
- aktueller Google Chrome Browser (<https://www.google.de/chrome/browser/desktop/index.html>) oder aktueller Mozilla Firefox (<https://www.mozilla.org/de/firefox/>) Browser

5.2 Installation

Die Installation wurde bereits vom Team vorgenommen. Die Zugangsdaten zum Server werden der Jury bereitgestellt. **Die Benutzerdaten entnehmen Sie bitte der E-Mail, welche unsere Abgabe enthält.**

Die Anleitung für eine manuelle Installation ist in Kapitel 5.3.2 zu finden.

5.3 Ausführung

5.3.1 Carefuel Basic

Zur Verwendung des Basic Projektes navigieren Sie bitte in der Eingabeaufforderung zu

InformatiCup/carefuel.basic

und geben Sie folgende Zeile ein:

```
java -cp target/carefuel_basic.jar carefuel.app.App
```

Daraufhin wird das Carefuel.Basic Projekt in der Komandozeile ausgeführt und es werden 3 Optionen angeboten:

- Wahl einer der angezeigten Routen
- Vorhersage von Benzinpreisen
- Beenden des Programms

Sie können zwischen allen Routen wählen, die im Verzeichnis InformatiCup/carefuel.basic/resource/routes/ vorhanden sind. Soll eine andere Route getestet werden, muss die Datei (im korrekten Format) in das obige Verzeichnis kopiert werden.

```
PS C:\Users\jwal\git\develop\InformatiCup\carefuel_basic> java -cp .\target\carefuel_basic_1.0-SNAPSHOT.jar carefuel.app.App
Startup of CareFuel_Basic at Thu Dec 28 15:12:01 CET 2017

***** WELCOME TO CAREFUEL *****

Select the route:
[1] Bertha Benz Memorial Route 2017.csv
[2] Bertha Benz Memorial Route.csv
[3] Emsland Route.csv
[4] Husum Route.csv
[5] Kiel-Augsburg.csv
[6] Leipzig-Stuttgart.csv
[7] Lüneburg Route.csv
[8] Peine-Ost Route.csv
[9] Ruhrpott-Tour.csv
[10] Rund-um-Berlin.csv
----- More options -----
[11] Predict gasoline prices
[12] Exit
```

Abbildung 12: Liste der Routen

Über die Eingabe der zugehörigen Nummer kann eine Route oder Option ausgewählt werden. Nach Wahl einer Route wird die beste Tankstrategie aus den hervorgesagten Benzinpreisen berechnet und auf dem Bildschirm ausgegeben, sowie unter


InformatiCup/carefuel.basic/out/routes/

im geforderten Format abgespeichert. Wird die Benzinpreisvorhersage ausgewählt, werden automatisch alle im Verzeichnis

InformatiCup/carefuel.basic/resource/pricePrediction/

vorhandenen Dateien angezeigt. Wird eine der Dateien ausgewählt, beginnt die Preisvorhersage und es werden auf der Konsole die vorhergesagten Preise ausgegeben, sowie im Verzeichnis

InformatiCup/carefuel.basic/out/pricePrediction/
im vorgegebenen Format abgespeichert. Soll eine noch nicht im Verzeichnis
angezeigte Datei verwendet werden, muss diese ins Verzeichnis
InformatiCup/carefuel.basic/resource/pricePrediction/
kopiert werden.



```
Select the file to predict prices for:
[1] gasStationsToPredict
[2] Exit
```

Abbildung 13: Auswahl der vorherzusagenden Benzinpreise

Das Wählen der Beenden-Option, beendet das Programm.

5.3.2 Carefuel Extended

Zur Installation der Carefuel Extended Anwendung müssen die unter Kapitel 5.1 genannten Voraussetzungen erfüllt sein.

Im Terminal müssen die Ports für den Server geöffnet werden:

```
sudo iptables -A INPUT -p tcp --dport http -j ACCEPT
sudo iptables -A INPUT -p tcp --dport https -j ACCEPT
```

Für den Server muss nun ein Ordner mit dem Namen carefuel im root-Verzeichnis erstellt werden und darauf hin der Server mit dem folgenden Befehl gestartet werden:

```
sudo java -jar -Dserver.port=80 carefuel_server-0.1.0.jar
```

Als nächstes wird die Datenbank aufgesetzt. Dazu muss die Datei /etc/apt/sources.list.d/pgdg.list mit dem Inhalt
'deb http://apt.postgresql.org/pub/repos/apt/ jessie-pgdg main' erzeugt werden und die folgenden Befehle ausgeführt werden:

```
sudo wget -quiet -O - https://www.postgresql.org/media/
keys/ACCC4CF8.asc — sudo apt-key add -
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib
```

Dann wird sich mit dem folgenden Befehlen in die Datenbank eingeloggt und die Daten eingelesen:

```
sudo -u postgres psql postgres
sudo su - postgres
psql carefuel < history.dump
psql carefuel < history.dump.1
```

Zuletzt muss das Skript für das Aktualisieren der Tankstellen-Daten ausgeführt werden (/carefuel/update_prices.sh). Das Skript ist im InformatiCup-Ordner unter sarefuel_extended/skripts zu finden. Sollen jede Nacht neue Daten heruntergeladen und eingebunden werden muss ein entsprechender Cronjob erstellt werden, der das Skript jede Nacht aufruft.

5.4 Verwendung

Es muss lediglich im Google Chrome oder Mozilla Firefox Browser zu <https://carefuel.ddns.net/> navigiert werden. Dort erscheint dann unsere Website, wie in Abbildung 14 dargestellt. Alternativ kann auch die leichtgewichtige Android-Applikation installiert werden (die APK befindet sich im Hauptverzeichnis unter carefuel_extended). Diese stellt allerdings auch lediglich unsere Website dar, wodurch sich keine Unterschiede in der Bedienung ergeben.

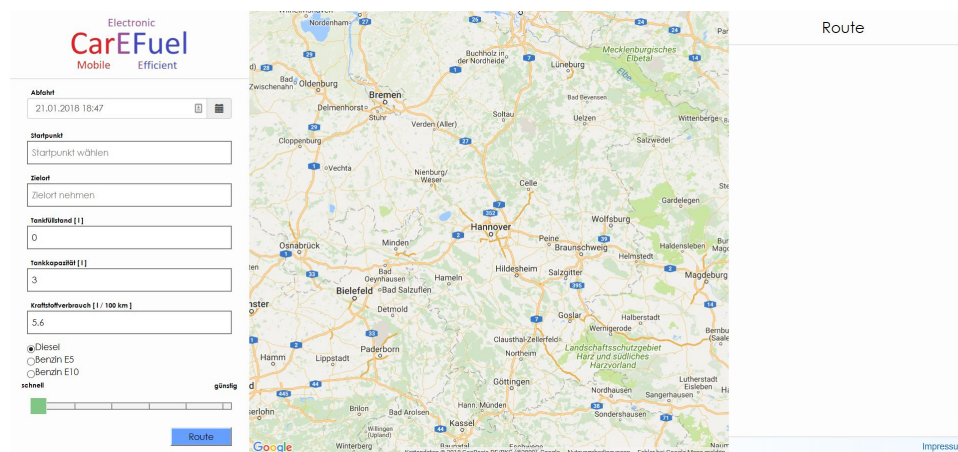


Abbildung 14: erster Aufruf der Website

Dort müssen nun ein beliebiger Start- und Zielort, sowie Angaben zum Tank und Kraftstoff und des gewünschten Fokusses (lieber eine schnelle oder eine günstige Route) ausgewählt werden. Durch einen Klick auf *Route* wird der Algorithmus gestartet und ein Ladebildschirm erscheint. Dies kann nun je nach Distanz und Konfiguration einige Zeit dauern. Sobald die Route berechnet ist, verschwindet der Ladebildschirm und die Route wird auf der Karte angezeigt. Sollte man sich in der Desktop- und nicht die mobilen Version der Website befinden, wird nun rechts auch zusätzlich noch eine textuelle Zusammenfassung der Route angezeigt, inklusive Distanz und Menge an Kraftstoff, die während des Stops getankt werden soll. Dies ist beispielhaft in Abbildung 15 dargestellt.

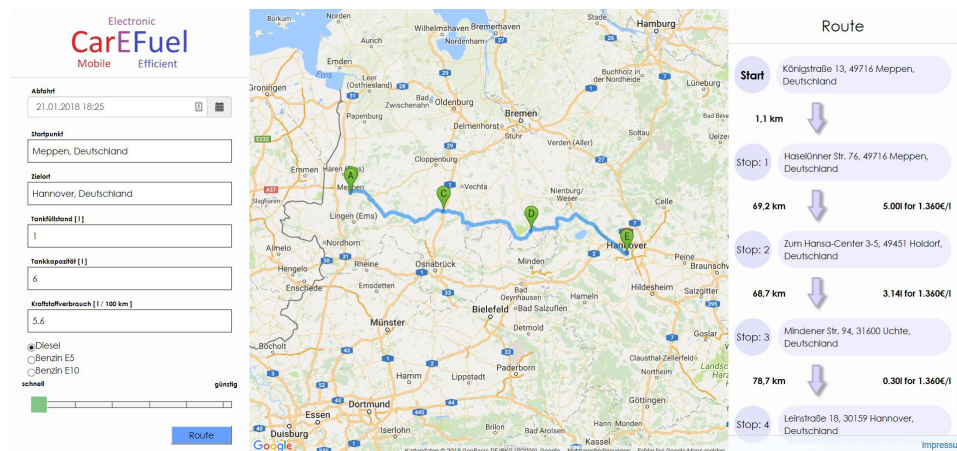


Abbildung 15: Route wird auf der Website angezeigt

5.5 FAQ

Es wurden andere Studierende und Freunde aus verschiedensten Kontexten gebeten unser Programm auszuprobieren. In der Folge sind häufiger gestellte Fragen mit Antworten dargestellt.

- **Ich habe mich vertippt, wie kann ich den Ladebildschirm schließen?** Dieses Verhalten ist nicht berücksichtigt. Sobald eine Route angefragt wird, muss diese vom Programm berechnet und zurückgegeben werden.
- **Ich frage eine weite Strecke an, aber selbst nach 30 Sekunden erscheint keine Route. Was mache ich falsch?** Möglicherweise gar nichts. Die Berechnung der Route kann je nach Distanz und Konfiguration auch länger als 30 Sekunden dauern.
- **Wenn ich den Slider von Schnell bis Günstig ganz links (schnell) habe**
- **Gibt es eine Möglichkeit Routen zu speichern oder muss ich jedes mal alles neu eintippen?** Zum aktuellen Zeitpunkt gibt es leider keine Möglichkeit Routen zu speichern. Es ist allerdings für die Zukunft der Einsatz von Profilen geplant, der das Speichern von Routen ermöglichen würde.
- **Ich tippe Start- und Zielort ein, aber trotzdem wird keine Route berechnet. Warum nicht?** Start- und Zielort müssen leider aus der Liste, die beim Tippen erscheint, ausgewählt und angeklickt werden. In Zukunft soll aber das bloße Eintippen reichen.

Abbildungsverzeichnis

1	Architektur	3
2	Explorative A*(EA) Algorithmus	5
3	Architektur	6
4	Architektur (Quelle: https://brilliant.org/wiki/feedforward-neural-networks)	7
5	Funktionsweise von Recurrent Neural Networks (Quelle: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)	8
6	Long Short Term Memory Architektur (Quelle: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)	8
7	Einteilung der historischen Preisdaten in feste Intervalle . . .	9
8	Netzwerk Architektur	9
9	Durchschnittliche Anzahl von gemeldeten Preisen pro Tag über die letzten Monate	10
10	Preise der Tank-Strategien	11
11	Preisverlauf einer beispielhaften Tankstelle über einen Tag . .	12
12	Liste der Routen	15
13	Auswahl der vorherzusagenden Benzinpreise	16
14	erster Aufruf der Website	17
15	Route wird auf der Website angezeigt	18

6 Literaturverzeichnis

- [1] “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting”. In: (2015).
- [2] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: Neural Comput. 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: <http://dx.doi.org/10.1162/neco.1997.9.8.1735>.
- [3] Samir Khuller, Azarakhsh Malekian, and Julián Mestre. “To Fill or Not to Fill: The Gas Station Problem”. In: (2011).
- [4] Xiaolei Maa Zhimin Tao Yinhai Wang Haiyang Yua Yunpeng Wang. “Long short-term memory neural network for traffic speed prediction using remote microwave sensor data”. In: (2015).