Formula One Race Prediction

J. Greil[1], F. Köhler[2], H. Weber[3], and M. Zeh[4]

[1] Siemens AG, Lyoner Straße 27, D-60528 Frankfurt am Main
e-mail: julian.greil@siemens.com
[2] Siemens AG, Lyoner Straße 27, D-60528 Frankfurt am Main
e-mail: koehler.florian@siemens.com
[3] Siemens AG, Lyoner Straße 27, D-60528 Frankfurt am Main
e-mail: hannah.weber@siemens.com
[4] Siemens AG, Carl-Benz-Straße 22, D-60386 Frankfurt am Main
e-mail: manuel.zeh@siemens.com

July 17, 2020

ABSTRACT

Aims. Es soll ein Algorithmus entwickelt werden, welcher nach der Hälfte eines Formel-1 Rennens die ersten 10 Endplatzierungen ausgibt.
Methods. Mithilfe von Featureengineering auf den vorliegenden Daten wird ein Neuronales Netz trainiert und die Modellparameter optimiert, sodass am Ende durch Crossvalidation ein finales Modell entsteht, welches verlässliche Vorhersagen liefert.
Results. Durch die Unvorhersehbarkeit der Rennen ist eine perfekte Voraussage des Rennausgangs schwer möglich. Jedoch erzielt das finale Modell insbesondere in den vorderen Platzierungen vielversprechende Ergebnisse.

Key words. Formula One – Race Prediction – Machine Learning – Deep Neural Networks – Hyperparametertuning

1. Einführung

Die Formel 1 ist mit 1,827 Milliarden US-Dollar 2018 eine der umsatzstärksten Sportarten überhaupt. Kumuliert sahen im Jahr 2019 knapp 2 Milliarden Zuschauer die Rennen rund um den Globus. Durch die verwendeten Technologien fallen pro Rennen viele verschiedene Daten an, von Wetterdaten über die Rundenzeiten bis zu den Endplatzierungen. Ziel des Projektes ist es, die anfallenden Daten und Datensätze aus vergangenen Formel-1-Saisons auszuwerten und daraus eine Vorhersage für künftige Rennen zu erstellen. Da in der Formel 1 Bruchteile von Sekunden über Sieg oder Niederlage entscheiden können, ist es eine besondere Herausforderung, präzise Vorhersagen zu treffen. Die hier vorausgesagten Ergebnisse können in verschiedenen Formen verwendet werden. Neben einfachen Anwendungsgebieten wie Sportwetten können die Vorhersagen als TV-Overlay für die Zuschauer in das Rennen eingebunden werden, um so auf Basis vergangener Jahre den Rennausgang zu prognostizieren. Am Ende soll es das Ziel sein, die Livedaten aus einem Rennen nach der Hälfte der absolvierten Runden einem Modell zu übergeben. Dieses soll das Endergebnis des Rennens ausgeben.

2. Verwandte Arbeiten

Das Thema, Rennergebnisse vorherzusagen, wurde bereits mehrfach aufgefasst und bearbeitet. So behandelt Eloy Stoppels in seiner Masterarbeit die Thematik, Rennergebnisse mit künstlichen neuronalen Netzen vorherzusagen (Vgl. Reference (1)). Der Autor erklärt hier, warum deep neural networks verwendet werden. Auch Mehmet Gulum behandelt die Rennvorhersage in einer Arbeit, jedoch widmet er sich dem Pferderennen. Da diese Sportarten sich jedoch relativ ähnlich sind, was den Verlauf eines Rennens und die Abhängigkeit von diversen Parametern angeht, ist hier die Relevanz der Arbeit zu unserem Thema gegeben. Gulum verwendet ebenfalls neuronale Netze, sowie "graph-based features", also gerichtete Graphen, um den Ausgang der Rennen zu bestimmen (Vgl. Reference (2)). Rory Bunker und Fadi Thabtah weiten die Verwendung von neuronalen Netzwerken auf die Voraussage von Sportergebnissen im Allgemeinen aus. Hier steht besonders das Trainieren des Algorithmus sowie die Unterteilung der Daten im Vordergrund (Vgl. Reference (3)).

3. Auswahl relevanter Features des Datensatzes

3.1. Grundsätzliches

Da nach Occams Razor nur das verwendet werden soll, was auch wirklich benötigt wird, muss eine Vorauswahl der relevanten Features getroffen und weitere Aggregationen vorgenommen werden. Das Ziel des Projektes ist es, nach 50% eines Rennens die ersten zehn Plätze des aktuellen Rennens vorherzusagen. Daher muss ein einzelnes Rennen und jeder Fahrer eindeutig mithilfe einer ID identifiziert werden können. Zum Trainieren des Modells muss das Endergebnis bekannt sein, welche als Zielvariable Modellen übergeben wird. Außerdem können - bis auf die finale Platzierung - alle Informationen, die erst später als nach 50% des Rennens vorliegen, entfernt werden. Der Fortschritt eines einzelnen Rennens kann mithilfe der aktuellen Runde geteilt durch die gesamten Rundenanzahl berechnet werden. Es werden zum Trainieren der Modelle nur Daten von Rennen ab 2011 betrachtet, da es ab dieser Saison gravierende Regeländerungen gab. Rennen vor 2011 würden also in Betracht auf den Usecase nur geringe Relevanz haben und darauf aufbauende Modelle schlechtestenfalls noch manipulieren, da keine Vergleichbarkeit zwischen Rennen von vor und ab 2011 gegeben ist.

3.2. Relevante Features vor Rennbeginn

Der Konstrukteur, für den ein Fahrer fährt, spielt eine tragende Rolle für das Endergebnis. Beispielsweise gewann das Team Mercedes-Benz AMG von 2014 bis 2019 jedes Jahr die sogenannte Konstrukteurs-Weltmeisterschaft. Daher ist anzunehmen, dass Fahrer, welche für bessere Konstrukteure wie Mercedes-Benz AMG fahren, eine bessere Chance auf den Rennsieg haben, als Fahrer, die in schwächeren Teams, wie zum Beispiel Haas fahren - dieses Team hat in der Geschichte der Formel 1 noch kein einziges Rennen gewonnen. Teams werden im vorliegenden Datensatz durch eindeutige IDs gekennzeichnet. Bei diesen IDs handelt es sich um ein kategorisches Feature, welches mit Hilfe von Dummy-Variablen nach dem One-Hot-Encoding Prinzip encoded werden muss. Um eine Übergabe in die Modelle zu erleichtern, wird diese Transformation einmal zu Beginn durchgeführt, bevor der Datensatz in einzelne Rennen aufgeteilt wird. Bei diesem Vorgehen wird für jedes Rennen ein zusätzliches Attribut für jeden Konstrukteur von 2011 bis 2017 erzeugt, selbst wenn nicht alle Konstrukteure an allen Rennen in diesem Zeitraum teilnehmen. Das Attribut würde in diesem Fall dann nur einen Nullvektor enthalten. Dieses Vorgehen wird auf alle nominalen Attribute ausgeweitet, welche in der Datenbasis vorliegen. So wird sichergestellt, dass jedes Rennen in der gleichen Dimension vorliegt. Des weiteren ist die Startposition eines Fahrers ebenso wie die Rennstrecke, entscheidend für Vorhersagen. So gibt es Hochgeschwindigkeitsstrecken, bei denen vor allem die Top-Speed eines Autos entscheidend ist, oder auch Strecken, bei denen das Kurvenverhalten und die Aerodynamik eines Autos eine große Relevanz besitzt. Außerdem spielen hierbei oftmals auch äußere Faktoren wie Luftdruck und Streckentemperatur eine Rolle. Die verschiedenen Autos reagieren unterschiedlich auf diese Einflussfaktoren. Um einen Zusammenhang zwischen Teams und Rennstrecken herzustellen, wird die Strecke ebenfalls als Feature aufgenommen und diese mit Hilfe von Dummy-Variablen darzustellen. Die Informationen über die Fahrer sollen nicht für Vorhersagen des Modells direkt genutzt werden. Dies hat den Grund, dass ein Modell sonst lernen könnte, dass ein gewisser Fahrer immer auf eine der vorderen Positionen fährt. Diese Positionierung ist jedoch teamabhängig. So wurde Sebastian Vettel 2008 als Fahrer für Toro-Rosso-Ferrari nur Achter in der Weltmeisterschaftsgesamtwertung, gewann aber in den Jahren 2010 bis 2013 vier Jahre in Folge mit dem Team RedBull Racing die Fahrer-Weltmeisterschaft. Die Information über das Abschneiden der verschiedenen Fahrer soll jedoch nicht verloren gehen. Deshalb wird innerhalb einer Saison mit Hilfe der vorherigen drei Rennergebnisse die aktuelle Form eines Fahrers berechnet. Diese setzt sich aus der Summe der Platzierungen der letzten drei Rennen geteilt durch drei zusammen und wird mit null initialisiert. Scheidet ein Fahrer aus, wird ihm die Startposition plus drei Strafplätze angerechnet. Ein Fahrer ist oftmals nicht allein für sein Ausscheiden verantwortlich, aus diesem Grund wird die Startposition genutzt, um die Form eines Spitzenfahrers durch einen Ausfall nicht extrem zu verschlechtern. Mit dieser Methodik soll eine realistische Position gefunden werden, die dem Können des Fahrers entspricht. Somit kann einem ausgeschiedenen Fahrer nie ein Wert zur Form hinzugefügt werden, der aus einer Podiumsplatzierung resultieren würde. Die Form zeigt sich als vielversprechendes Feature, da sie einen Korrelationswert von 0,58 mit der Endposition hat.

3.3. Daten aus dem Rennverlauf

Informationen, die während des Rennverlaufes entstehen, spielen bei der Vorhersage ebenfalls eine Rolle. So ist die aktuelle Position eines Fahrers ebenso von Bedeutung, wie seine Rundenzeiten. Diese einzelnen Rundenzeiten werden kumuliert, damit Modelle die Informationen über den genauen zeitlichen Unterschied zwischen den Fahrern in der letzten zu betrachtenden Runde bewerten können. Des Weiteren ist die Anzahl der bereits absolvierten Boxenstopps relevant, da diese einen direkten Einfluss auf die gefahrene Rennzeit haben. Wenn es ein Fahrer nicht schafft, 50% des Rennens zu absolvieren, werden alle Gründe für das Ausscheiden des Fahrers zusammengefasst und binär dargestellt. Für die Vorhersage spielt der Grund des Ausscheidens keine Rolle. Weiter angereichert wird der Datensatz mit der Information, ob es sich bei dem Rennen um ein Regenrennen handelt, da dieser den Verlauf eines Rennens stark variieren kann. Beim Vergleich von Trocken- mit Regenrennen lässt sich beobachten, dass die Performance der verschiedenen Autos voneinander abweicht und auch eine Erhöhung der Rundenzeiten vorliegt.

4. Gegenüberstellung verschiedener Modelle

Grundlage der folgenden Modelle ist das Aufsplitten der vorliegenden Daten. Nachdem diese als Teil der Datenaufbereitung und des Feature Engineerings in einer einheitlichen Form aufbereitet und bereitgestellt wurden, müssen diese nun Trainings-, Development- und Testdaten unterteilt werden. Der Trainingsdatensatz wird für das Training der Modelle bereitgestellt, die Developmentdaten für das Finden der besten Modellparameter und die Testdaten schließlich als finale Validierung und Vergleich der einzelnen Modelle. Hierfür werden die vorliegenden Rennen im DataFrame-Format nach dem 70/20/10-Prinzip aufgeteilt (vgl. Reference (4)). Da seit dem Jahr 2011 137 Rennen vorliegen, werden 96 Rennen als Trainingsdaten, 27 Rennen als Developmentdaten und 14 Rennen als Testdaten verwendet, um eine Aussage über die Güte von Modellen treffen zu können. Als Vergleichsmaß wird hier stets der Mean-Absolute-Error (MAE) verwendet, da dieser einfach zu implementieren ist. Um zu verhindern, dass Modelle die Reihenfolge der Rennen erlernen, werden die Rennen vor der Aufteilung gemischt. Da die Daten in aggregierter Form vorliegen, ergibt sich für jedes Rennen ein DataFrame mit den Dimensionen 22, 52. 22 ist die Anzahl der Fahrer pro Rennen, über welche die im bisherigen Rennverlauf gemachten Boxenstopps, die gefahrenen Minuten und weitere Informationen bekannt sind. 52 ist die Anzahl der Attribute, die von einem Trainingsbeispiel betrachtet werden sollen. Das erste Modell ist ein lineares Regressionsmodell, basierend auf dem Ordinary Least Squares Prinzip. Es bietet sich an, die gefahrenen Minuten als Zielvariable zu wählen, da sowohl die Podiumspositionen, als auch die insgesamt gefahrenen Zeiten der Fahrer in Minuten vorliegen. Zudem handelt es sich bei der gewählten Regression um eine Funktion, die anhand von gegebenen Datenbeispielen geschätzt wird. Somit ergibt es Sinn, ein Vorhersage- und nicht als Klassifizierungsproblem zu betrachten. Verwendet wird das von sklearn bereitgestellte Lineare Regressionsmodell LinearRegression(). Dem Modell kann mit Hilfe der fit() Methode ein Datensatz übergeben werden (vgl. Reference (5)). In diesem Beispiel handelt es sich um ein DataFrame, das kumuliert alle Zeilen der 96 zuvor als Trainingsdatensätze deklarierten DataFrames, also 96 * 22 Einträge enthält. Die Koeffizienten der einzelnen Attribute können dann mit Hilfe des

coef_- Attributes, welches Teil des linearen Regressionsmodells ist, ausgegeben werden. Mit Hilfe der predict()-Funktion wird dann eine Vorhersage gemacht, welche die gefahrenen Zeiten der Fahrer in Minuten angeben soll. Da es das Ziel des Projektes ist, das Ergebnis eines Rennens vorherzusagen, und es sich somit um ein Klassifizierungsproblem handelt - auch wenn es nicht als solches behandelt wird - reicht es nicht aus, die Zeiten allein vorherzusagen und mit den tatsächlichen Zeiten zu vergleichen. Stattdessen müssen die vorhergesagten Zeiten in die Rennergebnisse umgewandelt werden. Dies geschieht, indem die vorhergesagten Zeiten aufsteigend sortiert werden, da die Annahme getroffen wird, dass der Fahrer, der am schnellsten im Ziel ist, auch den ersten Platz belegt. Vor allem für die kategorischen Attribute ConstructorID und CircuitID sind als Koeffizienten besonders große negative Werte gelernt worden:

```
---coefficients for linear regression---
status_binary: 25.7776161126595
lap_position: -0.07816185219144312
race_completion: 56.25453557044228
grid: 0.11851603033653113
form: -0.10816392912479955
rain: -0.3272464112866711
bias: -165808869450.44742
circuitId_1.0: -104497541597.23811
circuitId_2.0: -104497541588.28265
circuitId_3.0: -104497541591.12566
circuitId_4.0: -104497541588.39464
circuitId_5.0: -104497541591.25186
```

Fig. 1. Koeffizienten für lineare Regression (I)

```
circuitId_69.0: -104497541589.89507
circuitId_70.0: -104497541595.5703
circuitId_71.0: -104497541592.15247
circuitId_73.0: -104497541593.96843
constructorId_1.0: -57158273417.47223
constructorId_3.0: -57158273417.712265
constructorId_4.0: -57158273418.07646
constructorId_5.0: -57158273418.57208
constructorId_6.0: -57158273418.14148
constructorId_9.0: -57158273418.85057
constructorId_10.0: -57158273417.45752
```

Fig. 2. Koeffizienten für lineare Regression (II)

Um dem entgegenzuwirken, werden Lasso- (L1) und Ridge-Regularisierung (L2) verwendet, die für manche Koeffizienten Werte nahe Null oder gleich Null im L1 Fall (vgl. Reference (7)) lernt. Zusätzlich soll sowohl für die Lasso- (L1) als auch für die Ridge (L2) Regularisierung ein optimaler Regularisierungsfaktor Alpha gefunden werden. Dieses Alpha kann bei der Erzeugung der linearen, regulierten Modelle übergeben werden. Für diese wird auch wieder auf die von sklearn bereitgestellten Modelle Ridge() und Lasso() zurückgegriffen. Um die Vorhersagen der regulierten Regressionen zu verbessern, wird mit Hilfe einer Grid Search in Zehntelschritten das Alpha zwischen 0 und 1 gesucht, für welches der MAE des Modells am geringsten ist. Wie die Abbildungen 3 bis 6 zeigen, werden die betroffenen Koeffizienten für das lineare Regressionsmodell mit beiden Varianten deutlich besser.

```
---coefficients for lasso regression---
status_binary: 25.77768140974023
lap_position: -0.07818390308267434
race_completion: 56.25436791323797
grid: 0.11851076755304284
form: -0.1081589348843398
rain: -0.32725287860854135
bias: 0.0
circuitId_1.0: -5.482558881654912
circuitId_2.0: 3.4727754653549847
circuitId_3.0: 0.6297907872423777
circuitId_4.0: 3.360818495203222
circuitId_5.0: 0.5037801505275916
circuitId_6.0: 6.779086923398938
circuitId_7.0: 0.3303854683370079
circuitId_9.0: -2.952967367328203
```

Fig. 3. Koeffizienten für Lasso-Regression (I)

```
circuitId_70.0: -3.814851891217119
circuitId_71.0: -0.39691238196062456
circuitId_73.0: -2.2128585495510364
constructorId_1.0: -0.05922857483720014
constructorId_3.0: -0.2992401446696627
constructorId_4.0: -0.6634786319929803
constructorId_5.0: -1.1589709679851123
constructorId_6.0: -0.7283659551876966
```

Fig. 4. Koeffizienten für Lasso-Regression (II)

```
---coefficients for ridge regression---
status_binary: 25.84418791318039
lap_position: -0.08070665415893563
race_completion: 55.29255529220573
grid: 0.11972183232996694
form: -0.10754535194783757
rain: -0.4035462409767298
bias: 0.0
circuitId_1.0: -5.8680400579602505
circuitId_2.0: 3.036376376093535
circuitId_3.0: 0.2371259536224382
circuitId_4.0: 2.957059999039457
circuitId_5.0: 0.11927430513456647
circuitId_6.0: 6.382798225588283
circuitId_7.0: -0.04838399268522368
circuitId_9.0: -3.3418562937564524
circuitId_10.0: -2.0269272779050076
```

Fig. 5. Koeffizienten für Ridge-Regression (I)

```
circuitId_70.0: -4.1911647084348065
circuitId_71.0: -0.7798495086246784
circuitId_73.0: -2.6440082651183725
constructorId_1.0: -0.6119367663228833
constructorId_3.0: -0.8513337041974998
constructorId_4.0: -1.2227830939574518
constructorId_5.0: -1.7123660006673371
constructorId_6.0: -1.2863921637385465
```

Fig. 6. Koeffizienten für Ridge-Regression (II)

Die Modelle sind aber nicht dafür geeignet, Vorhersagen für den Ausgang des Rennens zu machen. Das ist an der folgenden, beispielhaften Vorhersage in den Abbildungen 7 bis 9 erkennbar. So wurden mit allen drei Methoden die eigentlich Letztplatzierten auf die ersten drei Plätze vorhergesagt:

944.0_linreg:

	podium_position	predicted_position	total_minutes	prediction
0	1.0	5	91.151500	91.664612
1	2.0	7	91.280767	91.874817
2	3.0	8	91.388900	92.402100
3	4.0	4	91.943883	91.305542
4	5.0	11	92.488150	92.872925
5	6.0	10	92.763512	92.773560
6	7.0	6	92.816035	91.854431
7	8.0	13	93.103196	93.259216
8	9.0	9	93.285328	92.614502
9	10.0	14	93.314945	93.421021
10	11.0	12	93.340995	93.092773
11	12.0	17	93.475405	94.004517
12	13.0	15	93.500356	93.469482
13	14.0	18	93.509874	94.146973
14	15.0	19	93.667274	94.393066
15	16.0	16	93.987193	93.527832
16	17.0	2	96.842304	71.478760
17	18.0	3	97.028459	71.506836
18	20.0	1	91.593900	67.204834

Fig. 7. Beispielhafte Vorhersage mit linearer Regression

944.0_lasso:

	podium_position	predicted_position	total_minutes	prediction
0	1.0	5	91.151500	91.665017
1	2.0	7	91.280767	91.875099
2	3.0	8	91.388900	92.402437
3	4.0	4	91.943883	91.305963
4	5.0	11	92.488150	92.873174
5	6.0	10	92.763512	92.773734
6	7.0	6	92.816035	91.854628
7	8.0	13	93.103196	93.259215
8	9.0	9	93.285328	92.614613
9	10.0	14	93.314945	93.420988
10	11.0	12	93.340995	93.092689
11	12.0	17	93.475405	94.004499
12	13.0	15	93.500356	93.469466
13	14.0	18	93.509874	94.146891
14	15.0	19	93.667274	94.392887
15	16.0	16	93.987193	93.527585
16	17.0	2	96.842304	71.478518
17	18.0	3	97.028459	71.506542
18	20.0	1	91.593900	67.204954

Fig. 8. Beispielhafte Vorhersage mit Lasso-Regression

944.0_ridge:

	podium_position	predicted_position	total_minutes	prediction
0	1.0	5	91.151500	91.608891
1	2.0	7	91.280767	91.817218
2	3.0	8	91.388900	92.341323
3	4.0	4	91.943883	91.237665
4	5.0	11	92.488150	92.806976
5	6.0	10	92.763512	92.711540
6	7.0	6	92.816035	91.777698
7	8.0	13	93.103196	93.202855
8	9.0	9	93.285328	92.550871
9	10.0	14	93.314945	93.343054
10	11.0	12	93.340995	93.032110
11	12.0	17	93.475405	93.936286
12	13.0	15	93.500356	93.396236
13	14.0	18	93.509874	94.087263
14	15.0	19	93.667274	94.337221
15	16.0	16	93.987193	93.456700
16	17.0	2	96.842304	71.337190
17	18.0	3	97.028459	71.360733
18	20.0	1	91.593900	67.069737

Fig. 9. Beispielhafte Vorhersage mit Ridge-Regression

Aus diesem Grund werden die Zusammenhänge zwischen der Zielvariablen und den gewählten numerischen Attributen genauer betrachtet.

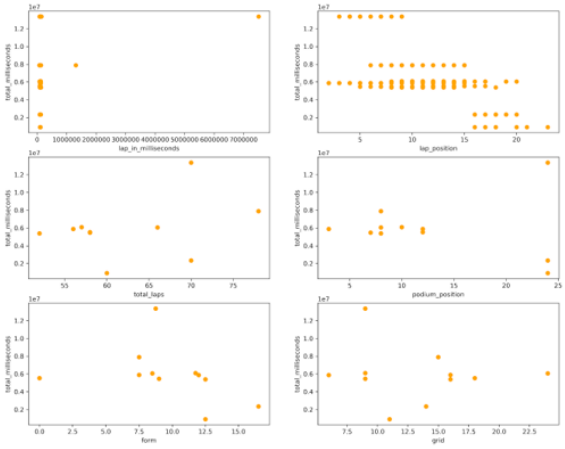

Fig. 10. Korrelation zwischen der Zielvariable und numerischen Werten

Es lässt sich anhand der geplotteten Graphen feststellen, dass es keinen direkten linearen Zusammenhang zwischen der Zielvariablen und den ausgewählten numerischen Attributen gibt. Ein lineares Regressionsmodell kann also nur eingeschränkt gute Vorhersagen auf dem vorliegenden Datensatz machen, da es primär dafür vorgesehen ist, lineare Zusammenhänge zwischen Attributen zu lernen, welche hier nicht vorliegen. Daher wird ein Modell bevorzugt, welches auch nichtlineare Zusammenhänge zwischen Attribute lernen kann.

4.1. Neuronales Netz

Beim zweiten Modell handelt es sich um ein Deep Neural Network. Dieses wird ausgewählt, da es hier neben der Nichtlinearität weitere Parameter gibt, welche zur Verbesserung der Ergebnisse angepasst werden können. Dazu gehören unter anderem die Anzahl der Layer, die Lernrate, die Anzahl der Neuronen und weitere Hyperparameter. Zu Beginn wird ein Netz gesucht, welches gute Ergebnisse auf den vorliegenden Daten liefert und an dem sich weiter orientiert werden kann. Dafür wird ein solches gebaut, welches aus Linear-Layers besteht, jedoch durch die Aktivierungsfunktion ReLU auch nichtlineare Zusammenhänge lernen kann. Um eine grobe Idee für den Aufbau eines Neuronalen Netzes in diesem Zusammenhang zu bekommen wurde (1) zur Orientierung verwendet. Als letztendliches Basis-Netzwerk wird ein Netzwerk aus acht linearen Layern, einer Dropout Layer, einem Adam Optimizer und einer Lernrate von 0.0001 mit Pytorch aufgebaut. Diese Parameter werden durch händisches Testen grob optimiert, da mit diesem Netzwerk auch mit mehreren Durchläufen konstante Ergebnisse erzielt werden. Ein weiterer Teil, welcher während diesem Prozess geändert wird, ist die Veränderung der Zielvariable. Wie bereits erwähnt, ist das eigentliche Ziel, die finale Rennzeit des jeweiligen Fahrers vorherzusagen. Allerdings werden hier auch keine wünschenswerten Zeiten vorhergesagt, da diese zu weit von den realistischen Zeiten abweichen. Deshalb wird die Zielvariable ausgetauscht, indem ein kontinuierlicher Wert für die finale Platzierung vorhergesagt wird. So kann das Netz einschätzen, wo welcher Fahrer das Rennen beenden wird und es lässt zu, dass jedem Fahrer nur eine Position zugeschrieben wird, indem man den Output ordnet. Mit dieser Methode wurden von Anfang an bessere Ergebnisse erzielt, die nun auch

dank eines niedrigeren MSE besser vergleichbar waren. Weitere Möglichkeiten für die Optimierungsfunktionen werden ebenfalls in Betracht gezogen und ausprobiert, unter anderem tanh und Sigmoid. Diese müssen auch in verschiedenen Kombinationen untersucht werden. Das wäre allerdings zu aufwändig, um es jedes Mal per Hand abzuändern und die Ergebnisse zu vergleichen. Deshalb wurde das oben beschriebene Netzwerk als Vergleichswert für andere, dynamisch aufgebaute Netzwerke verwendet.

5. Hyperparameteroptimierung des Neuronalen Netzes

Die Grundidee einer Hyperparameteroptimierung ist es, dass nicht nur ein gutes Modell für den jeweiligen Use Case ausgewählt wird, sondern auch die bestmöglichen Parameter in Zusammenhang mit diesem Modell gefunden werden. Im Folgenden wird eine solche Hyperparameteroptimierung jedoch nur unter gewissen Einschränkungen durchgeführt, um das Vorgehen und die Auswertung überschaubarer zu halten. Bei einem Neuronalen Netz lassen sich für den vorliegenden Use Case die folgenden Parameter als optimierbar identifizieren: Die Anzahl der zu trainierenden Epochen - im Folgenden Trainingsepochen - die Lernrate für den Adam Optimizer, die Anzahl der zu verwendenden Layer, die Neuronen und die Aktivierungsfunktionen für die Layer eines Netzes. Weiter eingeschränkt wird die Suche nach optimalen Parametern für das Neuronale Netz dadurch, dass grundlegende Annahmen für das Netz selbst getroffen werden. Dazu gehört, dass dieses nur aus den Layertypen Linear Layer besteht, während als Aktivierungsfunktionen ReLu, Sigmoid und Tanh verwendet werden. Um Overfitting entgegen zu wirken wird an einer zufälligen Stelle im Netz ein Dropout Layer platziert. Zusätzlich wird davon ausgegangen, dass die besten Ergebnisse erzielt werden, wenn die Anzahl an Neuronen bis zum mittleren Layer des Netzes zunehmen und ab da wieder abnehmen. Für die Backpropagation wird der Adam Optimizer verwendet. Die oben aufgeführten Parameter werden entkoppelt voneinander betrachtet. So werden die Lernrate und die Anzahl der Trainingsepochen gemeinsam optimiert, genauso wie die Anzahl an Layern und die Aktivierungsfunktionen, da die maximale Laufzeit eines Jupyter Notebooks in Google Colab auf zwölf Stunden beschränkt ist und eine gemeinsame Optimierung somit nicht ausführbar wäre (vgl. Reference (6)). Um die Layeranzahl und Aktivierungsfunktionen zu optimieren, wird eine Klasse deklariert, welche anhand von verschiedenen Einschränkungen dynamisch Neuronale Netze erzeugt, trainiert und wie zuvor auf den übergebenen Developmentdaten optimiert. Hierbei müssen die Trainingsepochenanzahl und die Lernrate für die Modelle festgelegt werden. Anhand von Erfahrungswerten und eingeschränkter Rechenleistung wurde sich für 3 und 0.0001 entschieden. Die Klasse geht wie folgt vor: Es werden Neuronale Netze mit verschiedenen Layeranzahlen, welche anhand eines übergebenen Intervalls vorgegeben sind, und Neuronenzahlen pro Layer erzeugt. Die Neuronenanzahl variiert pro Hidden Layer zwischen 30 und 200 nimmt bis zum mittleren Layer zu, und dann zum Outputlayer hin wieder ab (s.o.). Für jedes Layer wird zufällig entschieden ob es sich um ein Dropout-Layer oder ein Linear-Layer handelt. Da die Anzahl von möglichen Dropout-Layern auf eins limitiert ist, werden nach diesem nur noch Linear-Layer erzeugt. So kommt sowohl durch die zufällige Anzahl Neuronen, als auch durch die zufällige Stelle eines Dropout-Layers im Netz ein Zufallsfaktor in in die Netze. Aus Gründen der Durchführbarkeit wird von jeder Layeranzahl nur ein Netz erzeugt, theoretisch könnte aber mit der Klasse eine Reihe von Variationen von Netzen erzeugt werden, sodass mehrere Netze einer Layeranzahl auch untereinander verglichen werden können. Um eine Aussage über die Güte von Aktivierungsfunktionen zu treffen, werden die bereits erzeugten Netze kopiert und die zuvor festgelegte Aktivierungsfunktion (ReLU) gegen andere Aktivierungsfunktionen ausgetauscht. Auch hier wird das beste Neuronale Netz anhand des kleinsten MAE bestimmt und von der Optimizer-Klasse zurückgegeben.

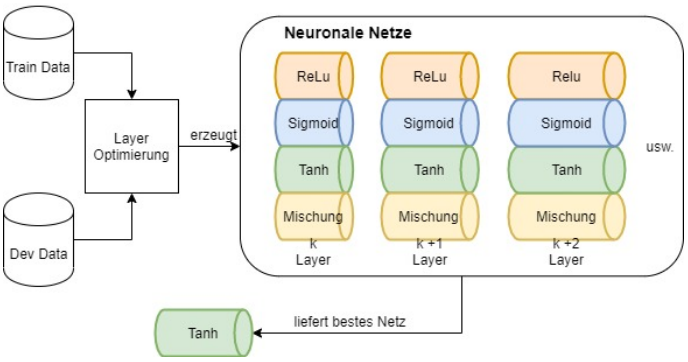

Fig. 11. Aufbau der Layer-Aktivierungsfunktionen-Optimierung

Um eine vergleichbare Menge von als gut eingestuften Neuronalen Netzen zur Verfügung zu haben, wird die Klasse zur Layer Optimierung 30 mal aufgerufen. Die Parameter der Klasse werden so übergeben, dass Netze mit 7 bis 11 Layern erzeugt werden. So werden pro Klassenaufruf 20 Netze erzeugt, da für jede Layeranzahl ein Netz mit ReLu, ein Netz mit Tanh, ein Netz mit Sigmoid und ein Netz mit einer Mischung dieser drei Aktivierungsfunktionen erzeugt wird. Aus diesen 20 Netzen wird das beste Netz von der Klasse zurückgegeben. Dadurch, dass die Klasse 30 mal aufgerufen wird, werden insgesamt 30 Netze zurückgegeben. Somit trainiert und testet der Algorithmus insgesamt 600 Neuronale Netze. Um die Lernrate und die Trainingsepochen dieser 30 Netze zu optimieren, wird eine Klasse verwendet, welche sich als Basis einer Grid Search bedient. Hierbei werden Parameter in festen Intervallen gesucht und gegeneinander getestet. Zu diesem Zweck wird der Klasse eine Range übergeben, in der sie nach der optimalen Zahl von Trainingsepochen sucht, ebenso wie eine Range, in welcher nach der bestmöglichen Lernrate für den Adam-Optimizer gesucht wird. Anhand dieser Angaben werden die Neuronalen Netze erneut auf dem Trainingsdatensatz trainiert, und auf dem Developmentdatensatz getestet. Als Gütemaß wird wieder der Mean Absolute Error gewählt. So kann die beste Lernraten- Trainingsepochen-Kombination aus den vorgegebenen Intervallen ausgewählt werden. Inwieweit der MAE der Netze anhand einer passenden Kombination aus Lernrate und Epoche verbessert wird, zeigt Abbildung 12:

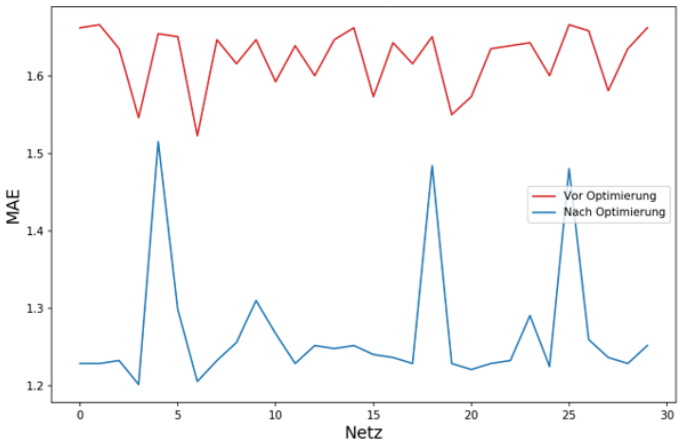

Fig. 12. Verbesserung des MAE nach Optimierung von Lernrate und Epochenzahl

Anhand dieser 30 Netze zeigt sich, dass das Dropout Layer häufiger im zweiten oder dritten Layer eines guten Netzes gefunden wird, als später im Netz: Zuletzt wird mit Hilfe von Cross-

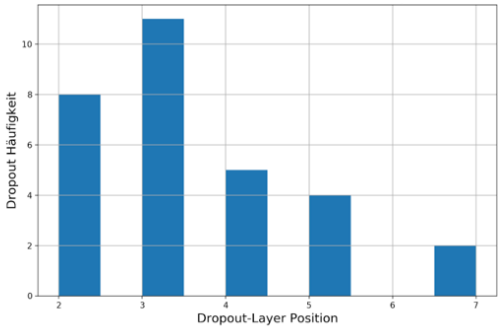

Fig. 13. Übersicht über die Positionen des Dropout-Layers

validation aus den 30 Netzen das beste Netz ausgewählt und auf dem kompletten Datensatz ohne das letzte Rennen der letzten Saison trainiert. Als bestes Netz wird ein Netz mit 7 Layern, dem Dropout Layer an 5. Stelle und nur Tanh Aktivierungsfunktionen ausgewählt. Es liefert auf dem Testrennen das folgenden Ergebnis:

Target	Prediction	Pred_Name	Target_Name
1	1	Valtteri Bottas	Valtteri Bottas
2	2	Lewis Hamilton	Lewis Hamilton
3	3	Sebastian Vettel	Sebastian Vettel
4	4	Kimi Räikkönen	Kimi Räikkönen
5	5	Max Verstappen	Max Verstappen
6	8	Esteban Ocon	Nico Hülkenberg
7	7	Sergio Pérez	Sergio Pérez
8	6	Nico Hülkenberg	Esteban Ocon
9	9	Fernando Alonso	Fernando Alonso
10	10	Felipe Massa	Felipe Massa

Fig. 14. Ergebnis der Vorhersage

Dieses Modell wird als finales Modell definiert. Für den Produktiveinsatz in der zukünftigen Saison, würde das Modell auf allen vorliegenden Daten trainiert werden.

6. Kritische Auseinandersetzung

Das Ziel des Projekts ist es, die ersten zehn Platzierungen am Ende eines Rennens vorherzusagen, jedoch allein von der gefahrenen Zeit eines Fahrers abhängig. Da zusätzliche Informationen wie Strafzeiten, die auf die gefahrene Zeit aufaddiert werden, einen Einfluss auf den schlussendlichen Ausgang eines Rennens haben, jedoch nicht vorliegen, ist es für das Modell nicht möglich, diese zu erlernen und in den Entscheidungsprozess mit aufzunehmen. Hiermit wird die den Vorhersagen zugrunde liegende Annahme, dass die Geschwindigkeit allein aussagekräftig für die finale Podiumsposition ist, widerlegt. Auch ist es nicht möglich, Strategien der Teams bei der Vorhersage zu berücksichtigen. Dies kann nur indirekt geschehen, in dem die bis zum bekannten Zeitpunkt des Rennens durchgeführten Boxenstopps übergeben werden. Die eigentliche Boxenstrategie der Teams lässt sich daraus jedoch nicht ermitteln, da mit Boxenstopps immer Verzögerungen in der jeweiligen Runde verbunden sind. Fahrer, die also in der ersten Rennhälfte weniger Boxenstopps gemacht haben, wirken insgesamt schneller, werden jedoch im noch unbekannten Rennteil mehr Boxenstops machen müssen. Durch die Wahl eines Neuronalen Netzes als finales Modell lässt sich keine Aussage darüber treffen, inwiefern Boxenstopps eine Auswirkung auf die Vorhersage des Netzes haben. Kritisch zu bewerten ist außerdem, dass die i.i.d. Vermutung der verwendeten Daten, welche jedem Modell zugrunde liegen sollte, durch die Form verletzt wird. Die Form basiert auf den Ergebnissen der direkt vorhergegangenen Rennen einer Saison und sorgt somit dafür, dass die Unabhängigkeit zwischen den Daten nicht mehr gegeben ist. Dieser Regelbruch der i.i.d. Vermutung kann jedoch akzeptiert werden, da so eine Dimensionsreduktion realisiert werden kann, die Vorhersagen des Modells verbessert werden und sich dieser Bruch mit dem Usecase vereinbaren lässt. Außerdem ist es kritisch zu betrachten, inwiefern Formel-1-Rennergebnisse vom Zufall geprägt sind und perfekt vorhergesagt werden können.

References

[1] Stoppels, E., 2017, Predicting Race Results using Artificial Neural Networks, University of Twente, https://essay.utwente.nl/74765/1/FinalThesisEloyStoppelsNoCompany.pdf, letzter Zugriff am 13.07.2020

[2] Gulum, M. A., 2018, HORSE RACING PREDICTION USING GRAPH-BASED FEATURES, University of Louisville, https://ir.library.louisville.edu/cgi/viewcontent.cgi?article=4083&context=etd, letzter Zugriff am 13.07.2020

[3] Bunker, R. P., Thabtah, F., 2019, A machine learning framework for sport result prediction, in: Applied Computing and Informatics, https://doi.org/10.1016/j.aci.2017.09.005, letzter Zugriff am 13.07.2020

[4] Kumar, S., 2019, Everything You Need To Know About Train/Dev/Test Split — What, How and Why; https://medium.com/@snji.khjuria/everything-you-need-to-know-about-train-dev-test-split-what-how-and-why-6ca17ea6f35, letzter Zugriff 29.06.2020

[5] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html, letzter Zugriff 11.07.2020

[6] https://research.google.com/colaboratory/faq.html#:~:text=How\%20long\%20can\%20notebooks\%20run,or\%20based\%20on\%20your\%20usage, letzter Zugriff am 13.07.2020

[7] Müller, A., Guido, S., 2016, Introduction to Machine Learning with Python: A Guide for Data Scientists, erste Ausgabe erschienen in O'Reilly UK, ISBN 9781449369415