End-to-End Learning für autonomes Fahren

Alexander Hoffmann

alexander.hoffmann3@haw-hamburg.de

Zusammenfassung. Der End-to-End Learning Ansatz zieht immer mehr Interesse auf sich, da er die Möglichkeit bereitstellt ein Problem mit einem übersichtlichen System zu lösen. Dieses System besteht aus Eingabe- und Ausgabedaten und einem neuronalen Netz dazwischen, der anhand der gewünschten Ausgabedaten passend zu der Eingabe trainiert werden muss. Auf das autonome Fahren bezogen könnte es sich bei der Eingabe um Bilder der Straße handeln und bei der Ausgabe um Geschwindigkeit und Lenkwinkel. In dieser Ausarbeitung werden verschiedene Arbeiten in dem Bereich des End-to-End Learning für autonomes Fahren vorgestellt, dabei werden einige davon grob beschrieben und die anderen näher erläutert.

Schlüsselwörter: Autonomes Fahren · End-to-End Learning · Reinforcement Learning · Imitation Learning · Machine Learning · Deep Learning

1 Einleitung

Die Vorstellung, dass ein Auto einen Menschen von Punk A zum Punk B bringt, ohne dass jemand am Steuer sitzt ist schon lange keine Zukunftsfantasie mehr. Schon jetzt gibt Fahrsysteme, die in der Lage sind das Auto auf der Straße zu navigieren und sogar einzuparken. Diese traditionelle autonome Fahrsysteme bestehen meistens aus Wahrnehmung, Lokalisierung und Kartierung, Pfadplanung, Entscheidungsfindung und Fahrzeugsteuerung. Wenn ein End-to-End-System für autonomes Fahren betrachtet wird, handelt es sich um eine Blackbox, wo Daten, die den Zustand des Autos und seiner Umgebung beschreiben, hereingehen und die Fahrzeugbefehle herauskommen. Dadurch ist ein End-to-End-System viel übersichtlicher.

2 Grundlagen

Da die Arbeiten in dieser Ausarbeitung sich auf den Markow-Entscheidungsproblem (MEP) oder auf Englisch Markov Decision Process (MDP) [PM17] beziehen muss er kurz erläutert werden. Für autonomes Fahren stellt MDP ein Model der Umgebung dar und ist meistens eine 4-Tupel (S, A, P, R). S ist eine Menge an Zuständen, und A eine Menge an Aktionen. Eine Transitionswahrscheinlichkeit P gibt an mit welche Wahrscheinlich ein System vom Zustand aus S mit einer Aktion aus A in den Zustand wiederum aus S überführt wird $P : S \times A \times S \rightarrow [0, 1]$.

Zum Schluss bekommt man bei jeder dieser Überführung von einem Zustand zum anderen eine Belohnung R mit $R : S \times A \times S \to \mathbb{R}$. Das Ziel ist es eine *policy* π zu finden, bei der die Summe aller Belohnung am höchsten ist. Bei einer *policy* werden Zustande aus S auf Aktionen aus A abgebildet. Reinforcement Learning (RL) ist eine Möglichkeit den MDP zu lösen. RL arbeiten anhand des Versuch-und-Irrtum-Prinzips und bekommt dementsprechend eine negative oder eine positive Belohnung. Somit sucht RL einen Weg bei der die Gesamtbelohnung am höchsten ist.

3 Traditionelles autonomes Fahrsystem

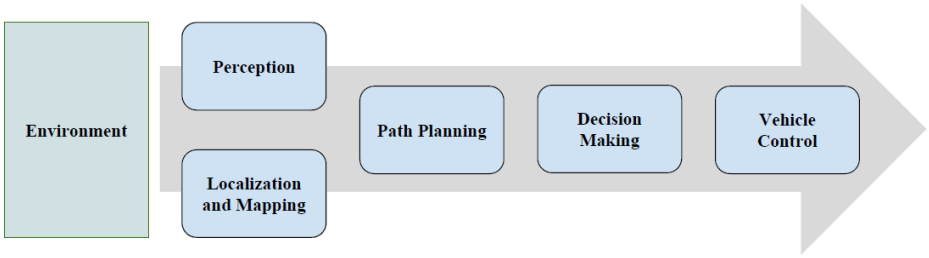

Abb. 1. Überblick über ein traditionelles autonomes Fahrsystem [Zha19]

Wie in der Abbildung 1 mit englischen Begriffen dargestellt ist, besteht das traditionelle autonome Fahrsystem meistens aus fünf Komponenten: Wahrnehmung, Lokalisierung und Kartierung, Pfadplanung, Entscheidungsfindung und Fahrzeugsteuerung [Che11]. Für alle diese Komponenten gibt es individuelle Herausforderungen. Die größten Probleme liegen in der Wahrnehmung und Lokalisierung und Kartierung. Bei der Wahrnehmung werden die Features von Menschen entworfen. Diese Features sind zum Beispiel Verkehrszeichen, Ampeln, Fahrspurmarkierungen und Begrenzungsrahmen für Fahr relevante Objekte. Dadurch müssen große Mengen an Daten manuell gelabelt werden, was auf aufwendig ist. Bei der Lokalisierung und Kartierung werden hochauflösende Karten erstellt, die detailliert, statisch und sehr genau sind. Diese Karten sind aufwendig Up-to-Date zu halten, wenn sie die Straßenlage verändert und zum Beispiel neuer Straßen dazu kommen oder verschwinden. Auch sind die Karten beschränkt, wenn es um kurzfristige Veränderungen geht wie zum Beispiel eine Baustelle. Auch dieser Aspekt würde bei End-to-End-System wegfallen, da in diesem System keine Karten erstellt werden.

4 End-to-End-System für autonomes Fahren

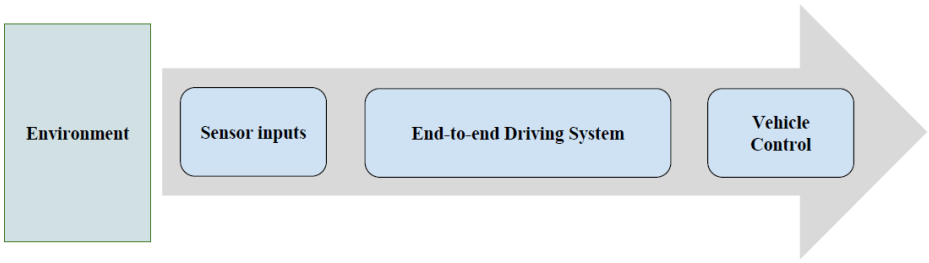

Abb. 2. Überblick über ein End-to-End-System für autonomes Fahren [Zha19]

Bei dem End-to-End-System handelt es sich um eine Blackbox. Durch Sensoren werden Daten aus der Umgebung gesammelt. Diese Daten können LiDAR oder Telemetrie Daten sein oder auch einfache Frames von einer oder mehreren Kameras. Die Daten werden einem neuronalen Netzwerk übergeben und am anderen Ende liefert dieses Netz die Befehle für das Fahrzeug. Ein gängiges Beispiel für die Befehle sind die nötige Geschwindigkeit und der nötige Lenkwinkel, um das Fahrzeug auf der Spur zu halten und gegen keine Hindernisse zu stoßen. Dieser Netzwerk wird vorher trainiert, um die richtigen Befehle zu erhalten.

5 DAVE 2

DAVE 2 ist eine End-to-End Systeme für autonomes Fahren [BDTD+16]. In diesem System wurde ein Convolutional Neural Network (CNN), zu Deutsch faltendes neuronales Netzwerk, verwendet, um das autonome Fahren zu realisieren. Der CNN wurde anhand einfachen RGB Bildern trainiert, wobei drei Kameras verwendet wurden mit jeweils 10 FPS.

5.1 DAVE (Rückblick)

Die Grundlagen für DAVE 2 lieferte das gleichnamige von 2004 entwickelte DAVE Projekt [LCB+04]. DAVE war ein kleines ferngesteuertes Fahrzeug, dessen CNN darauf trainiert wurde auf einem Schrottplatz Hindernissen auszuweichen.

5.2 Herausforderungen

Die Herausforderungen die DAVE 2 bewerkstelligen sollte wurden in drei Bereiche aufgeteilt. Der erste Bereich ist die fehlende Übereinstimmung der Daten zwischen Fahrer und dem gelehrtem Model, da dass Model in Zustände kommen kann, die vom Fahrer nicht vorgegeben wurden, wie zum Beispiel zu weit

in die Seite abzudriften. Der Zweite Bereich ist das unsichere Fahrverhalten des Fahrers, da jeder Fahrer anders fährt. Und der letzte Bereich ist da Testen und Bewerten des Models.

5.3 Data Augmentation

Um die fehlende Übereinstimmung der Daten zwischen Fahrer dem gelehrten Model entgegenzuwirken, müssen die gesammelten Daten erweitert werden. Die beiden äußeren Kameras zeigen jeweils schräg nach links und nach rechts, sodass sie die Situationen zeigen, was die Kamera aufzeichnen würde, wenn das Auto nach links oder rechts fahren würde. Das Netzwerk bekommt Bilder aus vielen möglichen Transformationen zwischen den drei Kameras und der Lenkwinkel entspricht dem Winkel, der das Fahrzeug zurück auf die Bahn führen würden. In Abbildung 3 ist das System aufgezeigt, wie es für das Training verwendet wurde. Das Fahren an sich geschieht nur anhand der Kamera in der Mitte.

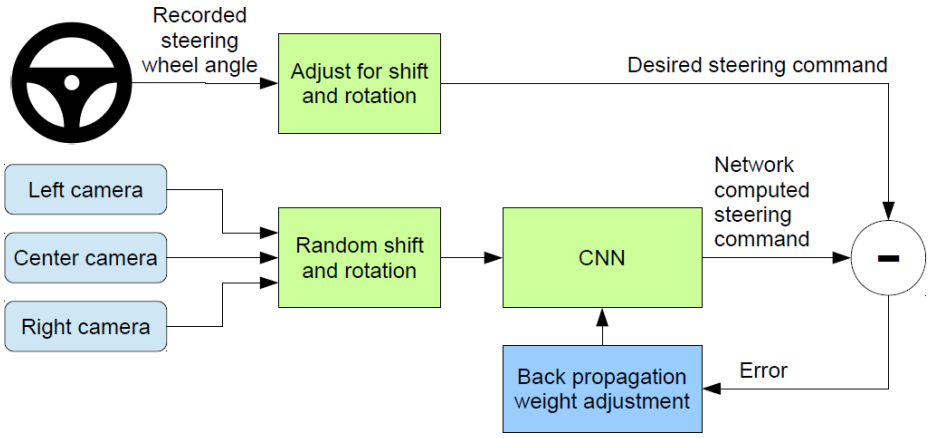

Abb. 3. Blockdiagramm für das Training des neuronale Netzwerkes [Zha19]

5.4 Videosimulator

Trainiert und getestet wurde das Netzwerk in einem Videosimulator. Dazu wurden die Fahrten auf verschiedenen Straßen mit dem dazugehörigen Lenkwinkel aufgezeichnet. Beim Testen wurden aufgezeichneten Bildern entsprechend dem Lenkwinkel transformiert. Überstieg der Abstand des Fahrzeugs zu der Mitte der Bahn ein Meter, wurde das Fahrzeug zurück auf die Mitte gesetzt. Die Abbildung 4 zeigt den Videosimulator.

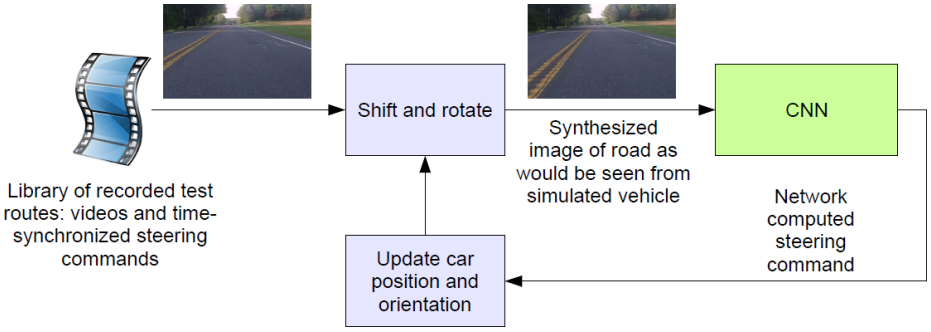

Abb. 4. Videosimulator [Zha19]

5.5 Fazit

Das System war in der Lage mit einer kleinen Menge an Trainingsdaten und weniger als 100 Stunden Training robust zu fahren. Das Fahre wurde auf Autobahnen, Landstraßen und Wohngebieten realisiert unter verschiedenen Wetterbedingungen: sonnig, bewölkt und regnerisch.

6 Learning to Drive in a Day

In dieser Arbeit wurde das autonome Fahren mit Reinforcement Learning realisiert [KHJ+19]. Die Parameter wurden am Anfang zufällig initialisiert. Es wurde ein kontinuierliches, modellfreies Deep Reinforcement Learning Algorithmus verwendet.

6.1 Architektur

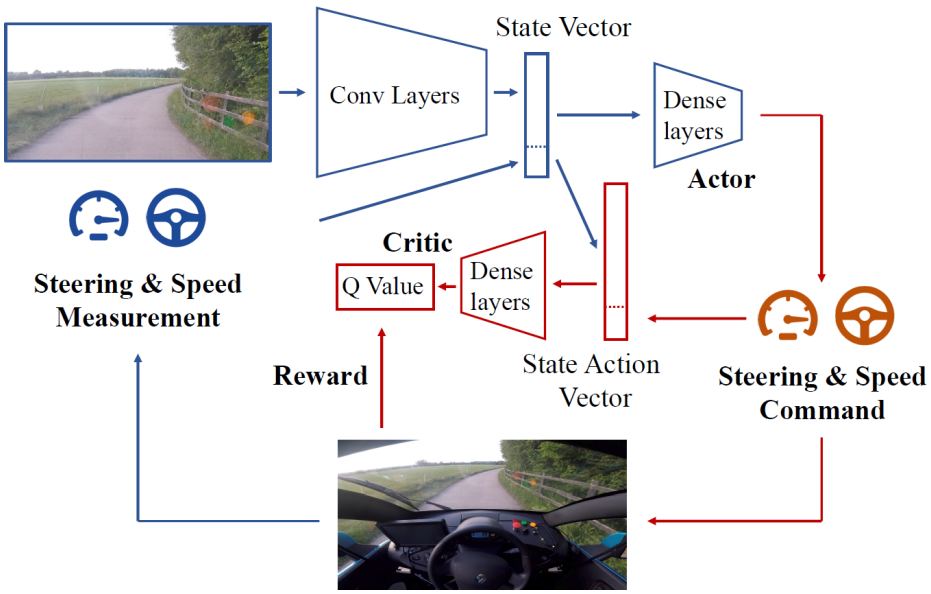

Abb. 5. Darstellung des Actor-Critic Algorithmus [KHJ+19]

Die Abbildung 5 beschreibt den verwendeten Algorithmus. Auf den MDP bezogen umfasst der Zustandsraum sowohl das Bild als auch die Geschwindigkeit und Lenkwinkel. Der Aktionsraum sind die Befehle für die Geschwindigkeit und Lenkwinkel. Für die Belohnungsfunktion wurde die zurückgelegte Strecke verwendet. Je weiter das Fahrzeug kommt, ohne abzudriften, desto höher die Belohnung. Bei dem DRL Algorithmus handelt es sich um Deep Deterministic Policy Gradients (DDPG). Der DDPG besteht auf zwei Funktionen. Die erste Funktion ist die *Critic* Funktion $Q : S \times A \rightarrow \mathbb{R}$, eine Bewertungsfunktion, die der Aktion aus A in Zustand aus S einen Wert zuordnet und somit die Elemente des kartesischen Produktes aus S und A auf eine reelle Zahl abbildet. Der sogenannte Q-Wert wird mit der Bellman Gleichung berechnet

$$Q(s_t, a_t) = r_{t+1} + \gamma(1 - d_t)Q(s_{t+1}, \pi(s_{t+1}))$$

anhand der *policy* $\pi : S \rightarrow A$, die wiederum die zweite Funktion des DDPG darstellt mit

$$\pi(s) = argmax_a Q(s, a).$$

Die Variable r_{t+1} ist die Belohnung für diese Aktion in diesem Zustand und γ gibt an wie stark die zukünftigen Q-Werte berücksichtigt werden. Die Variable d_t ist ein Abbruchkriterium, da wenn $d_t = 1$ ist der nächste Q-Wert nicht mehr

berücksichtigt wird. Bei $\pi(s)$ wird immer die Aktion gewählt mit dem größten Q-Wert. In dem Exploration-Teil des Trainings wurde zu der Aktion aus der *policy* Ornstein-Uhlenbeck-Prozess Rauschen hinzugefügt

$$x_{t+1} = x_t + \theta(\mu - x_t) + \sigma\epsilon_t$$

θ, μ und σ sind Hyperparameter und ϵ_t eine Zufallszahl aus der Normalverteilung $N(0,1)$. Die Parameter müssen sorgfältig gewählt werden, da kleines Rauschen zwar übersichtlicher ist, großes Rauschen dafür besser den Zustands- und Aktionsraum abdeckt.

6.2 Training

Das Training des Netzwerkes wurde in vier Aufgaben aufgeteilt: *train*, *test*, *undo*, *done*. Die Aufgabe *train* unterscheidet sich durch das hinzugefügte Rauschen von der Aufgabe *test*. Die Aufgabe *undo* setzt die vorherige Aufgabe *test* oder *train* zurück. Das Beenden des Experimentes wird durch *done* realisiert.

6.3 Probleme

Die Probleme, die in dieser Arbeit aufgeführt werden und Probleme der RL allgemein sind unter anderem das Sicherheitsrisiko in einer realen Umgebung. Da in RL Fehlverhalten erlaubt sind, kann es zu Schäden in der Umgebung oder anderen Beteiligten kommen. Zudem sind viele Misserfolge in der realen Welt nicht erschwinglich und stellen ein weiteres Problem dar. Dazu kommt noch, dass Experimente in der realen Welt teuer und zeitraubend sind. Um diese Probleme zuvorzukommen, wird das Training meistens in Simulationen durchgeführt, wobei ein Simulation nie die reale Welt gänzliche abbilden kann.

6.4 Fazit

Das Training wurde sowohl in einer Simulation, als auch in der realen Umgebung auf einer 250 m langen privaten Straße durchgeführt und es war möglich ein stabiles Fahrverhalten in weniger als 30 Minuten in der realen Umgebung zu realisieren.

7 DRL Framework

In dieser Arbeit wurde ein Framework vorgeschlagen und vorgestellt, das als Basis DRL hat [SAPY17]. Das Framework nutzt Recurrent Neural Networks (RNN) und Attention Model und wurde in The Open Racing Car Simulator (TORCS) verwendet. Es war in der Lage komplex Straßenkrümmungen und einfache Interaktion mit anderen Autos zu bewerkstelligen.

7.1 Überblick

Das Framework wurde in drei Kategorien aufgeteilt: Erkennung, Voraussage und Planung. Als Input dienten die Zustände der Umgebung und ihre Aggregation über die Zeit und als Output lieferte das Framework die Kommandos für die Fahraktionen.

7.2 Framework

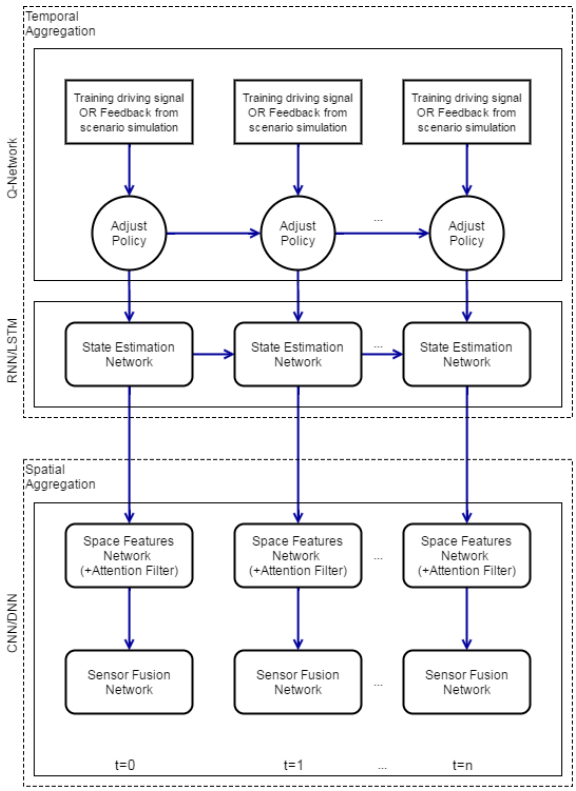

Abb. 6. Deep Reinforcement Learning Frameworks [SAPY17]

In Abbildung 6 ist das Framework dargestellt. Die Erkennung wurde von einem CNN erledigt, wobei ein Attention Filter dafür benutzt wurde, um nur fahrrelevante Objekte zu berücksichtigen. Das CNN erhielt die Sensorfusion aus dem Zustand des Autos und seiner Umgebung. Für die Voraussage wurde ein Long-Short Term Memory (LSTM) Netzwerk verwendet. Für die Planung wurde ein Deep Q-Network (DQN) Deep Deterministic Actor Critic (DDAC) genutzt. DDAC ermöglicht einen kontinuierlichen Aktionsraum.

7.3 Fazit

Mit diesem Framework war es möglich ein erfolgreiches Spurfolgen in TORCS zu erreichen mit komplexen Straßenkrümmungen und einfacher Interaktion mit anderen Autos. Ohne es explizit zu trainieren wurde beobachtet, dass das virtuelle Auto bei Straßenkrümmungen vom Gas ging.

8 SafeDAgger

Eine Lösung für das Problem der fehlenden Übereinstimmung der Daten zwischen dem Fahrer und gelehrtem Model ohne Data Augmentation ist *SafeDAgger* [ZC17]. In *SafeDAgger* wird das gelehrte Model laufen gelassen und die besuchten Zustände an den Experten übergeben, da es möglich ist in die Zustände zu kommen, die von dem Experten nicht vorgegeben wurden und das Model nicht weiß welche Aktionen er daraufhin ausführen soll. Der Experte liefert dann die dazugehörigen Aktionen. Anhand der Zustände und der Aktionen wird das Model weiter trainiert. Der Experte kann ein echter oder simulierter Fahrer sein. In dieser Arbeit wurde TORCS verwendet und wann der Experte gefragt wird bestimmt der *safety classi*

fier.

8.1 Imitation Learning

In *SafeDAgger* wird Imitation Learning (IL) verwendet. In IL gibt es zwei *policys*: *primary policy* π und *reference policy* π^*. Die *primary policy* wird anhand der *reference policy* trainiert und ist somit das gelehrte Model. Die *reference policy* ist in der Regel ein Experte, in diesem Fall ein simulierter Fahrer.

8.2 DAgger

SafeDAgger ist eine Erweiterung des *DAgger* Algorithmus [RGB11].

Initialize $D \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in $\mathit{\Pi}$.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
  Sample T-step trajectories using π_i.
  Get dataset $D_i = \{(s, \pi^*(s))\}$ of visited states by π_i and actions given by expert.
  Aggregate datasets: $D \leftarrow D \cup D_i$.
  Train classifier $\hat{\pi}_{i+1}$ on D.
end for
return best $\hat{\pi}_i$ on validation.

Algorithm 1: DAGGER Algorithmus [RGB11]

Der Algorithmus 1 beschreibt den *DAgger*. Nach der Initialisierung wird eine *policy* π_i ausgewählt, wobei β_i entscheidet wie viel von der *primary policy* $\hat{\pi}_i$ und wie viel aus der *reference policy* π^* genommen wird. Dann wird π_i ausgeführt und die besuchten Zustände gesammelt. Diese Zustände werden dem Experten übergeben und die resultierenden Aktionen mit den Zuständen in einem Datensatz gesammelt $D_i = \{(s, \pi^*(s))\}$. Dann findet die Aggregation statt $D \leftarrow D \cup D_i$ und anhand dieser Daten wird $\hat{\pi}_{i+1}$ trainiert.

8.3 Safety Classifier

Im Gegensatz zu *DAgger*, wo β_i darüber entscheidet, welche *policy* mit welchem Gewicht ausgewählt wird ohne irgendwelche Bedingungen, gibt es bei *SafeDAgger* den *Safety Classifier*. Der *Safety Classifier* beschreibt die Abweichung der *primary policy* von der *reference policy*.

$$\epsilon(\pi, \pi^*, \phi(s)) = ||\pi(\phi(s)) - \pi^*(\phi(s))||^2$$

Nur wenn die Differenz zwischen den beiden *policys* einen bestimmten Wert übersteigt, wird die *reference policy*, also der Experte, gefragt, ansonsten läuft die *primary policy*.

$$c^*_{safe}(\pi, \phi(s)) = \begin{cases} 0, & \text{if } \epsilon(\pi, \pi^*, \phi(s)) > \tau \\ 1, & \text{sonst} \end{cases}$$

τ ist der Schwellwert und $\phi(s)$ der beobachteter Zustand. Wenn zum Beispiel s alle nötigen Information von der Umgebung zusammenfasst, wie Markierungen, Schilder, andere Autos, usw., entspricht $\phi(s)$ einem Bild, aufgenommen von einer einzelnen Kamera.

8.4 Fazit

Durch den *Safety Classifier* gibt es viel weniger Anfragen an *reference policy*, was *SafeDAgger* effizienter macht. Zusätzlich wird dadurch gefährlichen Zuständen vermieden, da wenn die *primary policy* zu stark von der *reference policy* abweicht und dadurch eigentlich in einem gefährlichen Zustand kommen könnte, auf die *reference policy* zurückgegriffen wird. Sowohl *SafeDAgger* als auch *DAgger* sind auf die *reference policy* beschränkt und brauch somit einen Experten.

9 Weitere Arbeiten

9.1 PilotNet

Bei *PilotNet* wurde ein einfaches Bild als Input genommen und als Output lieferte er den Lenkwinkel [BYC+17]. Das CNN wurde unter verschiedenen Wetterbedingungen, mit und ohne Straßenmarkierungen trainiert, unter Verwendung von Data Augmentation. Das Ziel war es herauszufinden, was das Netzwerk lernt,

indem in die einzelnen Schichte des CNN hineingeschaut wurde. Es wurde festgestellt, dass *PilotNet* in der Lage war, für den Verkehr irrelevante Strukturen zu ignorieren. Besonders auffallend war, dass *PilotNet* Strukturen gelehrt hat, die zuvor gar nicht expliziert vorgegeben waren, somit zu den Fahrspurmarkierungen, Straßenränder und andere Autos auch Büsche, die den Straßenrand säumen gelehrt.

Abb. 7. Beispiele für hervorstechende Objekte bei verschiedene Bildaufnahmen [BYC+17]

In Abbildung 7 werden diese gelernten Strukturen bei drei verschiedenen Bildaufnahmen aufgezeigt

9.2 CARMA: A Deep Reinforcement Learning Approach to Autonomous Driving

Bei *CARMA* ging es um das autonome Fahren mit DQN in TORCS [VN16]. Es wurde ein Hybrid von CNN und RNN verwendet, wobei CNN das Bild I_t und RNN die Geschwindigkeit v_t, Abstand zur Straßenmitte d_t und Winkel zwischen Straße und Auto ϕ_t verarbeitet sollte. Für den Zustandsraum wurde somit ein 4-Tupel definiert mit dem Zustand $s_t = (I_t, v_t, d_t, \phi_t)$. In Abbildung 8 wird diese Architektur veranschaulicht.

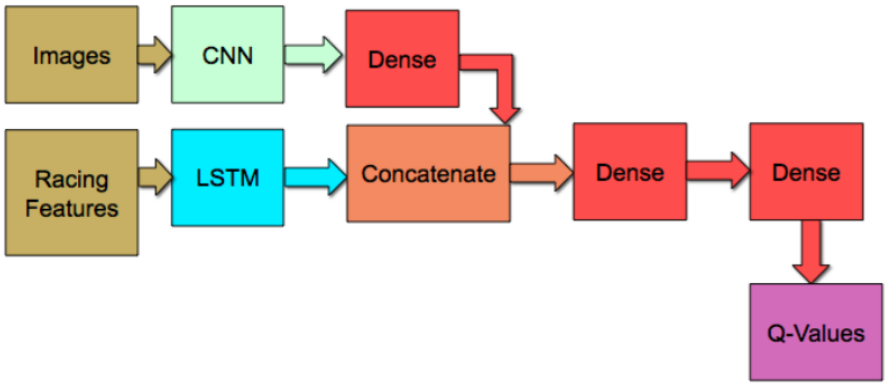

Abb. 8. Architektur des Hybrides aus CNN und RNN [VN16]

Für die Aktionen wurden drei Befehle für die Beschleunigung $acc_t = (Brake, DoNothing, Accelerate)$ und drei Befehle für die Lenkung $steer_t = (TurnLeft, DoNothing, TurnRight)$ definiert. Das ergibt 9 möglichen Kombination für die Aktion $a_t = (acc_t, steer_t)$.

---

function REWARD(s_t, a_t)
  $v_{min} \leftarrow 5$ ▷ Defined in meters/second.
  $v_{brake} \leftarrow 20$ ▷ Defined in meters/second.
  $v_{max} \leftarrow 18$ ▷ Defined in meters/second.
  $\phi_{threshold} \leftarrow 0.05$ ▷ Defined in radians.
  $reward \leftarrow 0$
  if $v_t > v_{brake}$ **and** $acc_t = Brake$ **then**
    $reward \leftarrow reward + 1$
  end if
  if $v_t > v_{brake}$ **and** $acc_t = Accelerate$ **then**
    $reward \leftarrow reward - 1$
  end if
  if $v_t < v_{max}$ **and** $acc_t = Accelerate$ **then**
    $reward \leftarrow reward + 1$
  end if
  if $v_t < v_{max}$ **and** $acc_t = Brake$ **then**
    $reward \leftarrow reward - 1$
  end if
  if $\phi_t < -\phi_{threshold}$ **and** $steer_t = TurnRight$ **then**
    $reward \leftarrow reward + 1$
  end if

22: **if** $\phi_t < -\phi_{threshold}$ **and** $steer_t = TurnLeft$ **then**
23: $reward \leftarrow reward - 1$
24: **end if**
25: **if** $\phi_t > \phi_{threshold}$ **and** $steer_t = TurnLeft$ **then**
26: $reward \leftarrow reward + 1$
27: **end if**
28: **if** $\phi_t > \phi_{threshold}$ **and** $steer_t = TurnRight$ **then**
29: $reward \leftarrow reward - 1$
30: **end if**
31: **if** $v_t < v_{min}$ **and** acc_t" $\neq Accelerate$ **then** ▷ Encourage forward motion at low speeds.
32: $reward \leftarrow reward + 1$
33: **end if**
34: **return** $reward$
35: **end function**

Algorithm 2: Belohnungsfunktion [VN16]

Der Algorithmus 2 zeigt die Belohnungsfunktion. Wenn das System sich nicht so verhält, wie es sollte zum Beispiel bei der Überschreitung einer vorgegebenen Geschwindigkeit zu bremsen, wird die Belohnung dekrementiert, ansonsten wird sie inkrementiert. Die Belohnungsfunktion sorgt dafür, dass *CARMA* lernt auf der Spur zu bleiben, was sie in der simulierten Umgebung auch in der Lage war.

9.3 Adaptive Behavior Generation for Autonomous Drivingusing DRL with Compact Semantic States

In Vergleich zu den zuvor vorgestellten Arbeiten, wo die Geschwindigkeit und oder Lenkung durch das neuronale Netzwerk, anhand von Sensorinput, realisiert wurde, wird in dieser Arbeit das Netzwerk darauf trainiert die Spur zu wechseln [WKW+18]. Die Steuerung und die Sensorverarbeitung übernehmen andere Komponente. In zwei Szenarien: Auf der Autobahn fahren und sich in den Verkehr der Autobahn einfädeln entscheidet ein DQN, ob das Auto die Spur wechseln soll oder nicht. Als Input für den DQN dient ein kompakter semantischer Zustand, wie er in Abbildung 9 dargestellt ist.

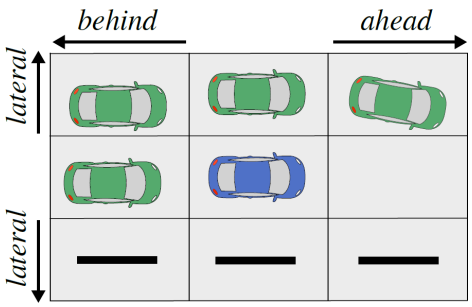

Abb. 9. Beispiel für einen kompakten semantischen Zustand [WKW+18]

Der Aktionsraum umfasst das Beschleunigen, Abbremsen und der Spurwechsel nach links oder rechts. Für die Belohnungsfunktion wird unter drei verschiedenen Prioritäten unterschieden. Die Kollisionsvermeidung hat die höchste, das Einhalten der Regeln die mittlere und der Fahrstil die niedere Priorität.

10 Zusammenfassung

In dieser Arbeit wurde zuerst das Markow-Entscheidungsproblem erläutert, da Reinforcement Learning und Imitation Leraning sich darauf beziehen, dann wurde der Unterschied zwischen dem traditionelle autonomen Fahrsystem und dem End-to-End-System für das autonome Fahren beschrieben und zum Schluss einige der Arbeiten, die sich mit dem End-to-End Learning beschäftigen vorgestellt. Diese Arbeiten bezogen sich immer auf das autonome Fahren. Mit End-to-End Learning ist es möglich das autonome Fahren mit wenig Aufwand zu realisieren, doch das Trainieren bei RL sollte in einer Simulation stattfinden, da in der realen Umgebung es kostspielig und zeitraubend ist und eine Gefahr für die Umgebung und andere Verkehrsteilnehmer darstellt. Das System ist leicht erweiterbar, da es nur unter anderen Bedingungen trainiert werden muss, ohne in das System einzugreifen und zu verändern. Da es sich bei End-to-End Learning um eine Blackbox handelt, ist das System übersichtlicher als ein System mit vielen Komponenten wie bei dem traditionellen autonomen Fahren. Doch genau diese Blackbox stellt auch eine Schwierigkeit dar. Es ist schwer nachzuvollziehen, wieso das System sich so entschieden hat und nicht anders. Werden die Methoden das System zu beschreiben und zu erklären auch weiterhin entwickelt und verbessert, stellt das End-to-End Learning eine effiziente und einfache Methode für das autonome Fahren.

Literatur

[BDTD+16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, and Jiakai Zhang. End to end learning for self-driving cars. 2016.

[BYC+17] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. 2017.

[Che11] Hong Cheng. *Autonomous intelligent vehicles: theory, algorithms, and implementation*. Springer Science & Business Media, 2011.

[KHJ+19] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254, 2019.

[LCB+04] Y. Lecun, E. Cosatto, J. Ben, U. Muller, and B. Flepp. DAVE: Autonomous off-road vehicle control using end-to-end learning, 2004.

[PM17] David L. Poole and Alan K. Mackworth. Artificial intelligence: Foundations of computational agents. In *Artificial Intelligence: Foundations of Computational Agents*, pages 458–462. Cambridge University Press, 2 edition, 2017.

[RGB11] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[SAPY17] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. 2017(19):70–76, 2017.

[VN16] Matt Vitelli and Aran Nayebi. Carma: A deep reinforcement learning approach to autonomous driving, 2016.

[WKW+18] Peter Wolf, Karl Kurzer, Tobias Wingert, Florian Kuhnt, and J. Marius Zollner. Adaptive behavior generation for autonomous driving using deep reinforcement learning with compact semantic states. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 993–1000. IEEE, 2018.

[ZC17] Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end simulated driving. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[Zha19] Jiakai Zhang. End-to-end learning for autonomous driving, 2019.