Duale Hochschule Baden-Württemberg Mannheim

DonkeyCar Simulator - Projektbericht Reinforcement Learning

Studiengang Wirtschaftsinformatik

Studienrichtung Wirtschaftsinformatik (Data Science)

Verfasser(in):	Olena Lavrikova, Philipp Dingfelder
Matrikelnummer:	5436924, 8687786
Kurs:	WWI20DSB
Studiengangsleiter:	Prof. Dr. Bernhard Drabant
Wissenschaftliche(r) Betreuer(in):	Janina Patzer
Eingereicht:	30.08.2023

Ehrenwörtliche Erklärung

Wir versichern hiermit, dass wir die vorliegende Arbeit mit dem Titel "*DonkeyCar Simulator - Projektbericht Reinforcement Learning*" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Ort, Datum

Mannheim, 30.07.2023

Olena Lavrikova, Philipp Dingfelder

Inhaltsverzeichnis

Abbildungsverzeichnis iii

Abkürzungsverzeichnis iv

Kurzfassung (Abstract) v

1 Einleitung 1

2 Entwicklung des RL-Agenten 3
- 2.1 Double Deep Q-Network . . . 3
- 2.2 Soft Actor-Critic . . . 4
  - 2.2.1 Autoencoder . . . 4
  - 2.2.2 Soft Actor-Critic (SAC)-Modell . . . 5

3 Bewertung und Visualisierung 7

Literaturverzeichnis 9

Abbildungsverzeichnis

1.1 DonkeyCar Oberservation Space 2

2.1 Architektur Autoencoder 4
2.2 Darstellung Autoencoder 5

3.1 Training und Evaluierungsreward 7

Abkürzungsverzeichnis

DDQN	Double Deep Q Networks
DHBW	Duale Hochschule Baden-Württemberg
SAC	Soft Actor-Critic

Kurzfassung (Abstract)

Der vorliegende Projektbericht handelt vom Training eines Reiforcement Algorithmus auf der DonkeyCar Simulation.
Das Trainieren der Modelle auf den ursprünglichen Bilddaten, die 120x160x3 Pixel im Falle des Donkeycars darstellen, wird als ineffizient und zeitaufwändig angesehen. Deshalb wurden neben den Modellen auch verschiedene Bildvorverarbeitungsmethoden verwendet. Hierfür wurde zuerst ein Double Deep Q Networks (DDQN)-Modell trainiert, das mit Lienendetection vorverarbeitete Bilder der Donkeycar-Umgebung als Eingabe bekommt und eine Aktion zurückgibt. Da dieser Ansatz nicht den gewünschten Trainingserfolg brachte, erfolgte im Anschluss das Trainieren eines SAC-Agenten, der anstelle der urpsrünglichen Bilder durch einen Autoencoder vorverarbeitete Bilder als Eingabe erhält. Hierbei zeigt sich, dass der Agent trainiert. Dennoch sind weitere Optimierungen notwendig, wie ein längeres Trainieren, das Tuning von Hyperparametern oder das Trainieren des Autoencoders auf unterschiedlichen Umgebungen, um den Agenten ebenfalls auf diesen anwenden zu können. Der Code zum Projekt und ein Video zum Trainingsverlauf ist in https://github.com/Len8/RL zu finden.

1 Einleitung

Das autonome Fahren weckt aufgrund seines Potenzials, die Mobilität zu revolutionieren sehr viel Interesse. Deswegen finden sich hierzu auch verschiedene Simulationsumgebungen und Projekte, die als Vorstufe zum autonomen Fahren dessen Herausforderungen imitieren. Eine dieser Umgebungen ist das Open-Source-Projekt DonkeyCar, das es ermöglicht das Funktionsprinzip autonomer Fahrzeuge zu verstehen sowie einen Einblick in den aktuellen Stand der Technik zu gewinnen. Für das DonkeyCar-Framework gibt es eine Simulationsumgebung, die installiert werden kann, sowie physische Modellautos. Die Idee dahinter ist, dass das Auto zunächst in einem virtuellen Simulator mithilfe von Reinforcement Learning traniert werden soll, um dann die erlernte Policy in die reale Welt übertragen zu können. Das Fahrzeug muss dabei lernen, auf der Strecke zu bleiben, Kurven zu bewältigen, Hindernissen auszuweichen sowie optimale Fahrstrategien zu entwickeln, um die Runde schnell und sicher zu absolvieren. Der Simulator stellt ein exaktes Abbild des DonkeyCar-Frameworks in zehn verschiedenen Strecken zur Verfügung.

Bei der vorhandenen Simulation gab es zu Beginn einen High-Fidelity-Simulator für DonkeyCar in Unity, jedoch fehlte die Unterstützung für Reinforcement Learning. Um den vorhandenen Simulator für Reinforcement Learning anzupassen, hat Tawn Kramer den Gym DonkeyCar zur Verfügung gestellt: *Github: Gym DonkeyCar*
Mit diesem lässt sich die Environment leicht in Python integrieren. Als Eingabedaten für mögliche Modelle (Observation-Space) liefert diese ein Bild der Dimension 120x160x3 (s. Abbildung 1.1). Damit kann im Anschluss ein Modell trainiert werden, dass im Action-Space kontinuierliche Lenk- und Gaswerte zurückgibt. Hierbei reichen die Lenkwerte von -1 bis 1 (je nachdem, ob nach links oder rechts gelenkt werden soll) und die Gaswerte von 0 bis 1 (je nachdem, wie schnell das Auto fahren soll).

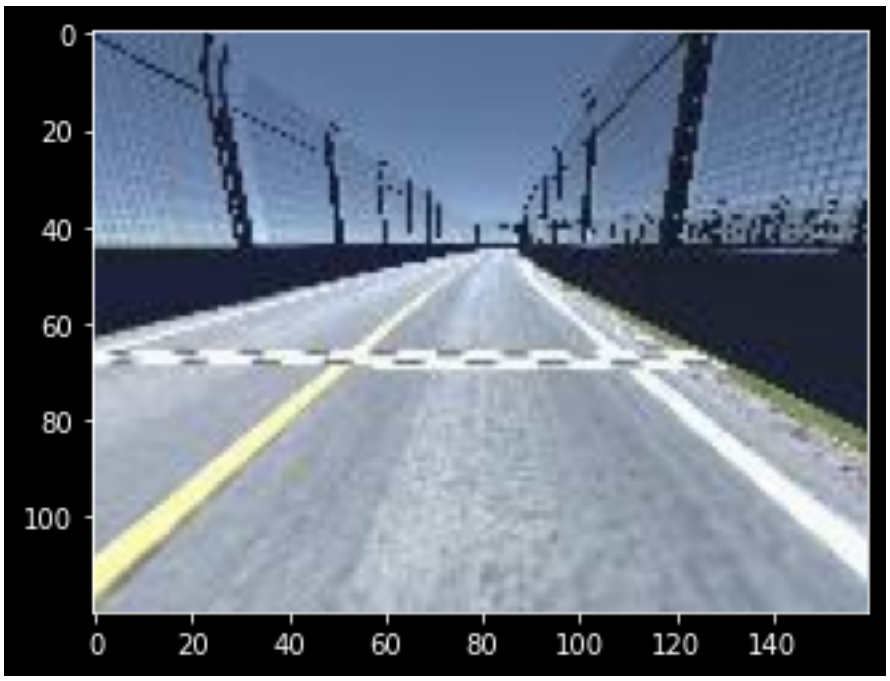

Abbildung 1.1: Darstellung einer möglichen Observation des DonkeyCar Simulators- Quelle: eigene Abbildung

Autonomes Fahren ist also eine zentrale Herausforderung der aktuellen Forschung insbesondere für die Automobilindustrie. Um in diesen interessanten und zukunftsrelevanten Bereich erste Einblicke zu bekommen, beschäftigt sich diese Arbeit mit der Entwicklung eines Reinforcement-Learning Algorithmus für die Simulationsumgebung DonkeyCar. Hierfür wurden verschiedenen Reinforcement Learning-Algorithmen, darunter Double Deep Q-Network und Soft Actor-Critic, darauf trainiert, eine simulierte Strecke selbstständig zu fahren. Genauere Details zu den Modellen und zur Umsetzung werden im nachfolgenden Kapitel beschrieben.

2 Entwicklung des RL-Agenten

Für das Training und die Optimierung des Modells wurden verschiedene Ansätze getestet. Allgemein besteht das Problem bei der DonkeyCar Umgebung, dass das Lernen aus reinen Bildpixeldaten äußerst ineffizient ist. Deswegen gliedern sich alle Beschreibungen in zwei Bestandteile auf: zum einen wird eine Vorverarbeitung angewendet, um die Daten in einen niedrigerdimensionalen Raum zu konvertieren, zum anderen findet auf diesen Vorverarbeiteten Beobachtungen das Training unterschiedlicher Reinforcement-Learning Algorithmen statt.

2.1 Double Deep Q-Network

Zuerst wurde ein Double Deep Q-Network (DDQN) verwendet, welches das Problem der Überbewertung von Q-Werten in der Q-Learning-Algorithmusfamilie löst. DDQN verbessert die Stabilität des Modells und ermöglicht es dem selbstfahrenden Auto, aus seiner Erfahrungen zu lernen und sich kontinuierlich zu verbessern. [vgl. 4, Kap. 12]

Dabei wurde versucht, dass der Agent bei der Lenkung sich nur auf die Position und Ausrichtung der Fahrspurlinien konzentiert. Damit sollte vermieden werden, dass es zu einer Überanpassung an die Hintergrundmuster kommt. Aus diesem Grund sollte der Hintergrund vernachlässigt werden.

Hierfür wurde eine Pipeline implementiert, diese beinhaltet:

- Canny Edge Detector, um die Kanten zu erkennen und zu extrahieren
- Hough-Line-Transformation, um die geraden Linien zu identifizieren
- Unterteilung der Linien in positiv und negativ neigende Linien
- Aussortierung von Linien, welche nicht zur Spur gehören

Diese Pipeline dient dazu, die Fahrspurlinien aus den Rohpixelbildern zu segmentieren, bevor sie in das CNN eingespeist werden. [vgl. 7]

Obwohl DDQN ein vielversprechender Algorithmus ist, hat sich die Kombination von DDQN mit der Erkennung der Fahrspurlinien bei der Implementierung als nicht optimal erwiesen.

Die Fahrspurlinien brachten in diesem Fall keine erkennbare Verbesserung des Agenten beim Training im Vergleich zum Trainieren auf Pixeldaten. Deswegen wurde ein weiterer Ansatz getestet, der im Folgenden vorgestellt wird.

2.2 Soft Actor-Critic

Auch beim zweiten Modell musste eine Strategie erstellt werden, wie das Modell aus den Bilddaten die richtigen Aktionen lernen kann. Zur Konvertierung der Daten in einen niedrigerdimensionalen Raum wurde zuerst ein Autoencoder trainiert und verwendet. Im Anschluss daran findet mit dieser neuen Repräsentation des Observations-Space das Training eines SAC-Modelles statt. Beide Bestandteile und deren Training werden im Folgenden erklärt.

2.2.1 Autoencoder

Der Autoencoder stellt eine Konvertierungsmöglichkeit hochdimensionaler Daten in niedrigere Dimensionen mithilfe eines neuronalen Netzes dar. Diese besteht aus dem Encoder, der die Eingabedaten über Hidden-Layers in eine geringe Dimension überführt, einem Bottleneck, dass die Stelle mit der geringsten Dimensionalität aufweist, und einem Decoder, der meist den Encoder spiegelt und für die Rekonstruktion der ursprünglichen Bilder aus der niedrigdimensionalen Latent-Space-Repräsentation zuständig ist (vgl. Abbildung 2.1).

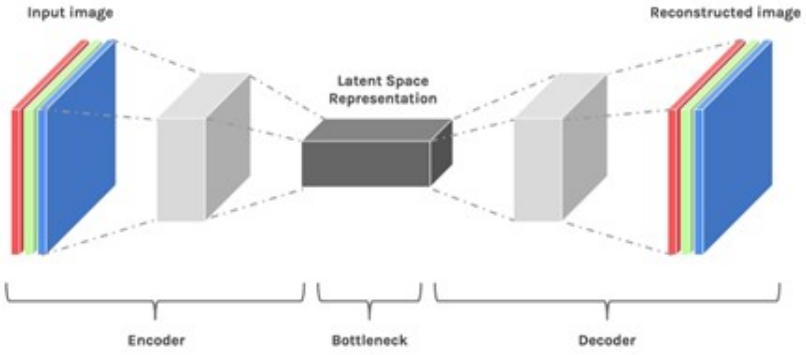

Abbildung 2.1: Architektur Autoencoder - Quelle: [8]

Das Training erfolgt durch die Minimierung der Unterschiede der rekonstruierten Daten im Vergleich zu den Originaldaten, was der sogenannten Rekonstruktion-Loss angibt. Es

werden also die Eingabedaten als Label verwendet, weswegen der Trainingsansatz auch als self-supervised Training bezeichnet wird. [2]

Für unseren Anwendungsfall wurde ein augmented Autoencoder trainiert, der auf der Trainingsfunktion von [1] basiert. Hierfür werden zuerst eigene Videodaten erstellt, während selbständig in der Umgebung gefahren wird. Im Anschluss daran werden diese Daten in Bilder konvertiert, die auf den relevanten Fahrbereich reduziert werden und anschließend als Trainingsdaten für das Convolutional Auto-Encoder Modell dienen. Dieses erhält eine Latent-Space von 32, das heißt, dass die ursprünglichen 57.600 Datenpunkte in 32 Datenpunkten abgebildet werden, was eine deutliche Dimensionsreduktion darstellt. Mithilfe des Decoders können die ursprünglichen Bilder aus den Daten des Latent-Space im Anschluss wieder rekonstruiert werden, um beispielsweise das Modell zu evaluieren. Eine Abbildung hiervon ist in Abbildung 2.2 gegeben. Dabei zeigt sich, dass das rekonstruierte Bild nur den relevanten Teil des ursprünglichen enthält, da nur die Strecke ohne Hintergrund dargestellt wurde. Auf der Rekonstruktion des Bildes sind die Fahrlinien zu erkennen, das Training des Autoencoders war also erfolgreich.

(a) Ursprüngliches Bild der Simulationsumgebung DonkeyCar aus den Trainingsdaten des Autoencoders

(b) Rekonstruiertes Bild mithilfe des Latent-Spaces und des Decoders

Abbildung 2.2: Visualisierung der Bilddaten des Autoencoders

2.2.2 SAC-Modell

SAC ist ein Off-Policy-Algorithmus, welcher eine stochastische Richtlinie verwendet. Der Algorithmus basiert auf dem Konzept der Entropie, welche die Unsicherheit oder Unvorhersehbarkeit der Zufallsvariablen aussagt. Die Richtlinie gibt vor, welche Aktion in einem Zustand ausgeführt werden soll. Im Falle einer hohen Entropie der Richtlinie, wird diese

unterschiedliche Aktionen ausführen. Bei einer niedrigen Entropie, führt die Richtlinie jedes Mal die selbe Aktion aus. Die Erhöhung der Entropie der Richtlinie fördert die Exploration, während die Verringerung zu weniger Exploration führt. Der Akteur zielt darauf ab, die erwartete Belohnung zu maximieren und gleichzeitig die Entropie zu maximieren. Dabei soll er seine Aufgabe erfolgreich meistern und so willkürlich wie möglich agieren. Die Zielfunktion besteht aus zwei Termen: einer für die Belohnung und ein anderer für die Entropie der Richtlinie. Anstatt nur die Belohnung zu maximieren, maximiert der SAC folglich auch die Entropie der Richtlinie. [vgl. 6, Kap. 12]

Mit der Kombination von Off-Policy-Aktualisierungen mit einer stabilen stochastischen Akteur-Kritiker-Formulierung erreicht der Algorithmus eine hochmoderne Leistung bei einer Reihe kontinuierlicher Aufgaben und übertrifft frühere On-Policy- und Off-Policy-Methoden. Darüber hinaus ist der SAC im Gegensatz zu anderen Off-Policy-Algorithmen sehr stabil und kann über verschiedene Zufallsstartwerte hinweg eine sehr ähnliche Leistung erzielen. [vgl. 3]

Der SAC eignet sich aus mehreren Gründe besonders gut für die DonkeyCar Umgebung:

- Dieser wurde speziell für die Steuerung in kontinuierlichen Aktionsräumen entwickelt
- Entropie gewährleistet mehr Exploration, dies ermöglicht verschiedenen Situationen zu erlernen
- Mithilfe von Off-Policy-Updates, kann der Agent aus den Erfahrungen in der Vergangenheit lernen

Das Training des SAC-Modells orientierte sich an den in der Vorlesung kennengelernten Codebeispielen, insbesondere an Kapitel 12. Das finale Modell wurde auf der Mountain Car Environment trainiert, auf der auch der Autoencoder optimiert ist. Als Eingabedaten dient das durch diesen vorverarbeitete Bild mit 32 Datenpunkten. Die maximale Trainingszeit wurde auf zwei Stunden festgelegt. Die finalen Ergebnisse des Modelltrainings werden im folgenden Kapitel erläutert.

3 Bewertung und Visualisierung

Während des Trainings aber insbesondere während der Evaluierung zeigt sich ab ca. Epoche 200 eine langsame aber kontinuierliche Verbesserung des gleitenden durchschnittlichen Rewards Abbildung 3.1. Daraus lässt sich schließen, dass das Modell trainiert. Dennoch ist weiteres und längeres Training notwendig, wie auch die Evaluierungsvideos in der Readme des GitHub Repositorys zeigen. Im Vergleich zu aktuellen state-of-the-art Modellen, wie sie beispielsweise in [5] zu finden sind, ist die Performance des trainierten Modells deutlich geringer. Beispielweise schafft der trainierte Agent aktuell noch keine vollständige Runde, während bei anderen Ansätzen eine möglichst schnelle Rundenzeit das Ziel ist. Diese wurden jedoch auch mit optimierten Parametern und mit deutlich längerem Training und leistungsfähigerer Hardware trainiert. Auch tendiert der Agent dazu, stark nach links zu lenken zu Beginn, was damit zu begründen ist, dass der Agent rechts startet und der Reward höher ist, je mittiger der Agent im Bild fährt.

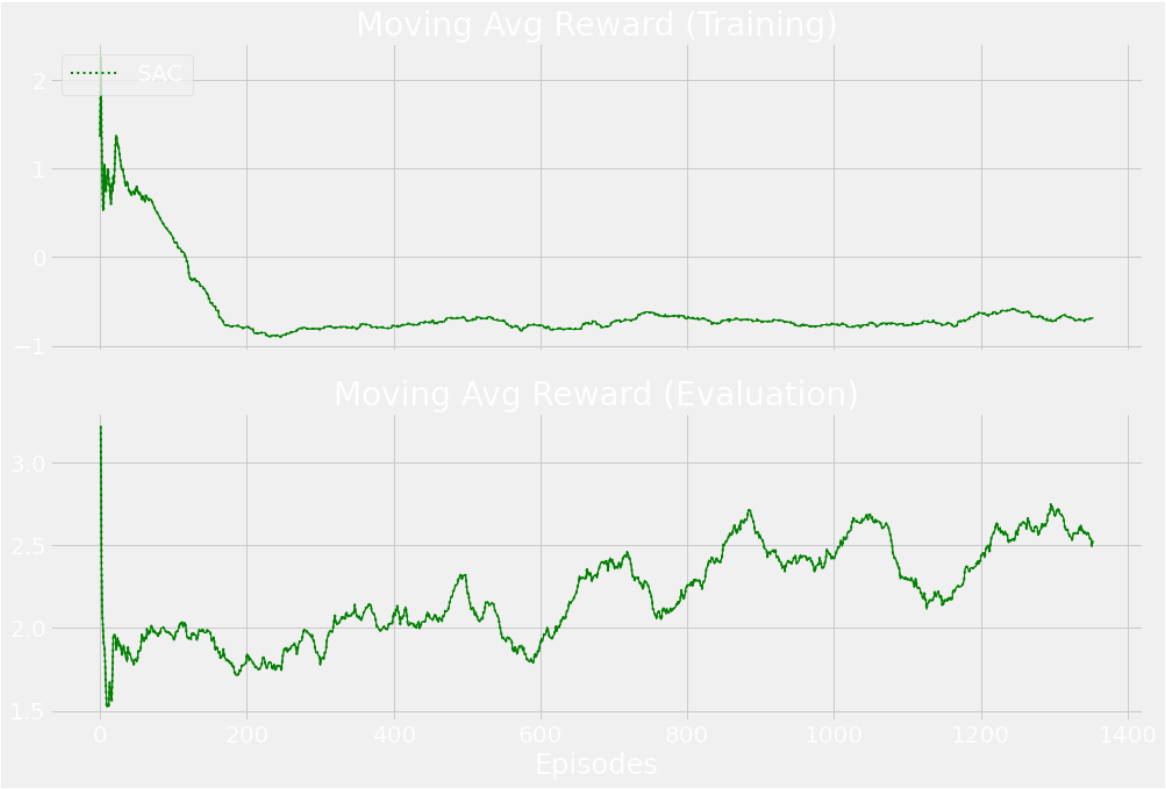

Abbildung 3.1: Reward des Trainings- und Evaluierungsprozesses über die Epochen - Quelle: eigene Abbildung

Als Optimierung wurde die Einführung verschiedener Modelle und Vorverarbeitungstechniken durchgeführt. Darüber hinaus bestehen etliche weitere Optimierungspotentiale, die nicht im Rahmen der Arbeit durchgeführt werden konnten. Hierzu zählen primär ein längeres Training. Auch Hyperparameter-Tuning kann durchgeführt werden, um das Modell weiter zu optimieren. Die Verwendung weiterer Informationen, wie der die Geschwindigkeit des Modells, oder das Stacking verschiedener aufeinanderfolgender Bilder für ein Training sind ebenfalls sinnvolle weitere Änderungen im Trainingsablauf. Darüber hinaus wurde festgestellt, dass sich das Modell und insbesondere der Autoencoder, der nur auf einer DonkeyCar Strecke trainiert wurde, nur bedingt für weitere Strecken eignet. Vor allem bei Strecken ohne Bande an der Seite neigt das Modell dazu zu erlernen, stark in eine Richtung zu fahren, und durch diesen Kreis die Ziellinie zu überschreiten. Hierdurch gibt die Umgebung dem Modell sogar eine Rundenzeit aus. Dies könnte dadurch verhindert werden, dass das Fahren in die falsche Richtung stark bestraft wird. Alternativ kann ebenfalls der Autoencoder auf weiteren Strecken nachtrainiert werden. Auch Weiterentwicklungen des Autoencoders wie der Variational Autoencoder sind denkbar.

Als Herausforderungen während des Projektes stellte sich zuerst die Wahl eines geeigneten Algorithmus dar. Hierbei wurden verschiedene RL-Modelle getestet, wobei häufig kein Training ersichtlich war, insbesondere wenn das Modell auf den ursprünglichen Bilddaten trainiert wurde. Auch wurden eine Vielzahl an unterschiedlichen vorhandenen Quellen verwendet, die das Modelltraining beschreiben. Hierbei stellte sich jedoch häufig heraus, dass die Modelle respektive Repositorys nicht direkt ausführbar sind (häufig aufgrund von veralteter Versionen). Deswegen wurde dazu übergegangen, das Modelltraining von Grund auf aus den Vorlesungsbeispielen zu übernehmen und zu bearbeiten. Auch die Umgebung DonkeyCar stellte eine Herausforderung dar, vor allem aufgrund der benötigten Vorverarbeitungen der Bilddaten und der langen Trainingsdauer, die unter anderem daraus resultiert, dass die Umgebung dauerhaft dargestellt werden muss, um mit dieser zu interagieren.

Literaturverzeichnis

[1] Antonin Raffin. *Augmented Auto-Encoder Training Code*. 5.04.2022. URL: https://github.com/araffin/aae-train-donkeycar/tree/feat/live-twitch-2.

[2] G. E. Hinton* und R. R. Salakhutdinov. „Reducing the Dimensionality of Data with Neural Networks“. In: *SCIENCE* 2006.313 (), S. 504–507.

[3] Tuomas Haarnoja. *Soft Actor-Critic: Off-Policy maximum entropy deep reinforcement learning with a stochastic actor*. Jan. 2018. URL: https://arxiv.org/abs/1801.01290.

[4] Zeqing Jin et al. „Machine learning for advanced additive manufacturing“. In: *Matter* 3.5 (Nov. 2020), S. 1541–1556. DOI: 10.1016/j.matt.2020.08.023. URL: https://doi.org/10.1016/j.matt.2020.08.023.

[5] Antonin Raffin. *RL Baselines3 Zoo*. https://github.com/DLR-RM/rl-baselines3-zoo. 2020.

[6] Sudharsan Ravichandiran. *Deep Reinforcement Learning with Python - second edition*. URL: https://learning.oreilly.com/library/view/deep-reinforcement-learning/9781839210686/Text/Chapter_12.xhtml#_idParaDest-331.

[7] *Train Donkey car in unity Simulator with reinforcement learning | Felix Yu*. Sep. 2018. URL: https://flyyufelix.github.io/2018/09/11/donkey-rl-simulation.html.

[8] Vindula Jayawardana. *Autoencoders — Bits and Bytes of Deep Learning: What's an Autoencoder?* Hrsg. von Towards Data Science. 4.08.2017. URL: https://towardsdatascience.com/autoencoders-bits-and-bytes-of-deep-learning-eaba376f23ad.