Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Projektarbeit (Informatik)

Reinforcement Learning mit einem Multi-Agenten System für die Planung von Zügen

Autoren	Dano Roost Ralph Meier
Hauptbetreuung	Andreas Weiler
Nebenbetreuung	Thilo Stadelmann

Datum	18.09.2019

Zusammenfassung

Zusammenfassung in Deutsch

Abstract

Abstract in English

(Deutschsprachiges Management Summary)

(Englischsprachiges Management Summary)

Vorwort

Stellt den persönlichen Bezug zur Arbeit dar und spricht Dank aus.

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbständige Verfassen einer Projektarbeit an der School of Engineering

Mit der Abgabe dieser Projektarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum: Unterschriften:

…… ……

……..

……

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Projektarbeiten zu Beginn der Dokumentation nach dem Abstract bzw. dem Management Summary mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Contents

1. Introduction

1.1. Baseline

- Nennt bestehende Arbeiten/Literatur zum Thema -> Literaturrecherche
- Stand der Technik: Bisherige Lösungen des Problems und deren Grenzen
- (Nennt kurz den Industriepartner und/oder weitere Kooperationspartner und dessen/deren Interesse am Thema Fragestellung)

This work explores a real world usage of deep reinforcment learning (RL) for controlling train traffic in a complex railway system. As part of the flatland challenge, a contest created by the Swiss Federal Railways (SBB AG) and the crowdsourcing platform AICrowd [1], we try to improve the performance of RL based train guidance and rescheduling. The goal of the challenge is to successfully guide all trains to their assigned target stations without getting stuck. This is challenging because a single wrong decision can cause a chain reaction that makes it impossible for many other trains to sucessfully reach their destinations. This endeavour is further complicated by trains with different speed profiles and the possibility of malfunctioning trains. In the words of SBB and AICrowd, the challenge is described as follows [1]:

> The Flatland Challenge is a competition to foster progress in multi-agent reinforcement learning for any re-scheduling problem (RSP). The challenge addresses a real-world problem faced by many transportation and logistics companies around the world (such as the Swiss Federal Railways, SBB. Different tasks related to RSP on a simplified 2D multi-agent railway simulation must be solved. Your contribution may shape the way modern traffic management systems (TMS) are implemented not only in railway but also in other areas of transportation and logistics. This will be the first of a series of challenges related to re-scheduling and complex transportation systems.

The challenge consists of two parts [1].

- Part 1 includes avoiding conflicts with multiple trains (agents) on a given environment. The difficulty thereby is, that the layout of the environment is not known upfront.
- Part 2 aims to optimize train traffic which includes trains with different speed profiles, malfunctioning trains, less switchover facilities and in general more scheduled trains in a shorter time.

This work is based on the work of Stefan Huschauer [2] and further investigates on the idea to use the asynchronous advantage actor critic algorithm (A3C) [3], a state of the art RL algorithm, to solve the task. Besides the work of S. Huschauer, this work is also influenced by the work of Bacchiani, Molinari and Patander [4]. That work also aims to apply the A3C algorithm in a cooperative multi agent environment and investigates communication free cooperation. Unlike the flatland challenge, the goal of this work is to cooperate on a road traffic environment. By applying the A3C algorithm, the work shows that it is possible learn cooperation by treating the other agents as part of the environment. Both the works of Bacchiani, Molinari and Patander as well as the work of S. Huschauer use a shared policy for all acting agents.

1.2. Goal of this work

- Formuliert das Ziel der Arbeit
- Verweist auf die offizielle Aufgabenstellung des/der Dozierenden im Anhang
- (Pflichtenheft, Spezifikation)
- (Spezifiziert die Anforderungen an das Resultat der Arbeit)
- (Übersicht über die Arbeit: stellt die folgenden Teile der Arbeit kurz vor)
- (Angaben zum Zielpublikum: nennt das für die Arbeit vorausgesetzte Wissen)
- (Terminologie: Definiert die in der Arbeit verwendeten Begriffe)

The aim of the work is to explore the use of the A3C algorithm in the flatland multi agent environment and to improve on the approach of S. Huschauer [2]. This work is targeted towards an audience with a brief understanding of deep reinforcement learning. A basic introduction into the topic is given in section 2.1. Also, the details of the flatland environment can be found in section 2.2. For a deeper understanding of the complex flatland environment, it is recommended to study the flatland documentation and specification [5] as well as the official flatland introduction [1].

1.3. Zielsetzung / Aufgabenstellung / Anforderungen

2. Technical Foundation

2.1. Reinforcement Learning

Basic Definitions

In recent years, major progress has been achieved in the field of reinforcement learning (RL) [6],[7], [8]. In RL, an agent learns to perform a task by interacting with an environment $\mathcal{E}$. On every timestep t the agent needs to take an action u. The selection of this action u is based on the current observation s. The success of the agent is measured by reward $\mathcal{R}$ received. If the agent does well, it receives positive reward from the environment, if it does something bad, there is no or negative reward. The goal of the agent is now to take an action that maximizes the expected future reward $\mathbb{E}[\mathcal{R}_{t+1} + \mathcal{R}_{t+1} + \mathcal{R}_{t+1} + ...|s_t]$ given the current observation s.
The current observation s_t, also known as the current state is used to determine which action u to take next. An agent can observe its environment either fully or partially.

Value Based vs. Policy Gradient Based Methods

Reinforcement learning methods are categorized into value-based methods and policy-based methods [9],[10]. Those variants differ on how they select an action u from a given state s. Value-based RL algorithms work by learning a value function $\mathcal{V}(s)$ through repeated rollouts of the environment. $\mathcal{V}(s)$ aims to estimate the future expected reward for any given state s as precisely as possible. Using this approximation $\mathcal{V}(s)$ we can now select the action u that takes the agent into the next state s_{t+1} with the highest expected future reward. This estimation $\mathcal{V}(s)$ is achieved by either a lookup table for all possible states or a function approximator. In this work, we solely focus on the case that $\mathcal{V}(s)$ is implemented in form of a neural network as function approximator. To train the neural network, we try to minimize the squared difference between the estimated reward $\mathcal{V}(s)$ and the actual reward:

$$loss_{value} = (\mathcal{R} - \mathcal{V}(s))^2$$

The second category of reinfocement learning algrithms are the so called policy gradient based methods. These methods aim to aquire a stochastic policy π that maximizes the expected future reward $\mathcal{R}$ by taking actions with certain probabilities. Taking actions based on probabilities solves an important issue of value based methods, which is, that by taking greedy actions with respect to state s, the agent might not explore the whole state space and misses out on better ways to act in the environment.

Asynchronous Advantage Actor Critic Algorithm

The progress in RL has led to algorithms that combine value based and policy gradient based methods. To enhance the process of learning policy π, the policy loss gets multiplied by the difference between actually received reward $\mathcal{R}$ and the estimated future reward $\mathcal{V}(s)$. This difference is called the advantage $\mathcal{A}$.

$$\mathcal{A} = \mathcal{R} - \mathcal{V}(s)$$

This advantage is then used to update the policy.

$$loss_{policy} = log\pi(s_t) * \mathcal{A}$$

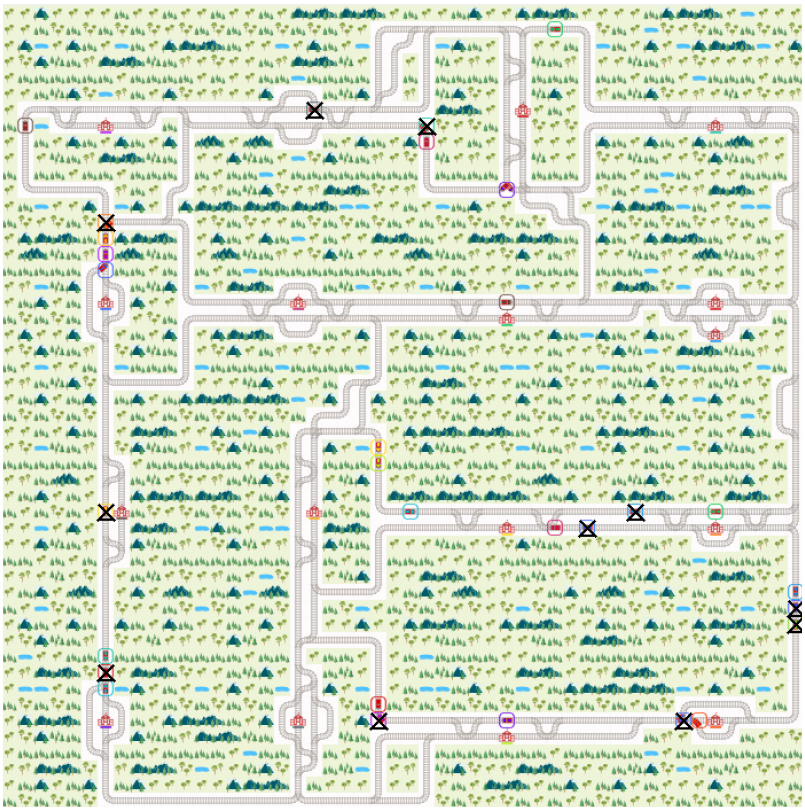
This means that for actions where the received reward $\mathcal{R}$ exceeds the expected reward $\mathcal{V}(s)$ the policy update gets multiplied by a positive advantage. Therefore, the update of the neural network gets update into a direction that favors the experienced actions based on the seen states.

Relation to this Work

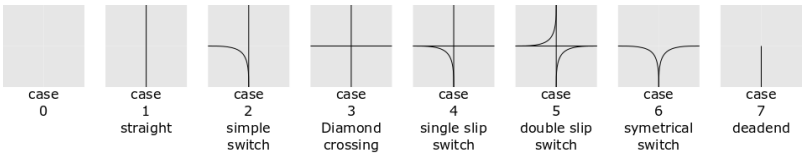
The goal of this work is to apply an RL algorithm to the vehicle rescheduling problem. Based on the work of S. Hubacher (source!!!), we use a distributed RL algorithm that learns a policy to control the traffic of trains on a rail grid. To do so, we use the asynchronous advantage actor critic algorithm [3] and expand its definiton to the use case of multiple agents, similar to [4].

2.2. The Flatland Rail Environment

The flatland environment is a virtual simulation environment provided by the Swiss Federal Railway SBB and the crowdsourcing platform AICrowd. The goal of this environment is to act as a simplified simulation of real train traffic.



Using flatland, we can train RL algrithms to control the actions of trains, based on observations on the grid. Flatland has a discrete structure in both its positions and its timesteps.The whole rail grid is composed out of squares that can have connections to neighbouring squares. In certain squares, the rails splits into two rails. On those switches, the agent has to make a decision which action it wants to take. Dependent on the type of switch, there are different actions available.



An exception poses switches that are approached from a side that does not allow to take an action, e.g. approaching a *case 2* switch from the north side. All rail parts, independent of if it is a switch also allow to take the actions to do nothing (remain halted, or keep riding), to go forward or to brake. The action space is therefore defined by:

$$U = \{\text{Do nothing, go left, go forward, go right, brake}\}$$

It is important to note that trains do not have the ability to go backwards and therefore need to plan ahead to avoid getting stuck. To learn which actions to take, the agents have to learn to adapt to an unknown environment due to the fact that the environments are randomly generated and differ on each episode. Depending on the given parameters, the size and complexity of the grid can be adjusted. This allows for dynamically changing the difficulty for the agents.
The goal of each agent is to reach an assigned target train station as fast as possible. Agents that reach this destination are removed from the grid which means, they can no longer obstruct the path of other trains.

Agent Evaluation

AICrowd an SBB provide a system for agent evaluation. This system evaluates the policy on a number of unknown environments and outputs the percentage of agents that reached their destination as well as the received reward while doing so. The evaluation reward scheme is thereby as follows:

$$R_t = \begin{cases} -1, & \text{if } s_t \text{ is not terminal} \\ 10, & \text{otherwise} \end{cases}$$

All submissions to the flatland challenge are getting graded by the percentage of agents that made it to destionation. (Source) Additionally we use our own evaluation parcour with an increasing difficulty of environments to get more insight into the agents strenghts and weaknesses.

Observations

The flatland environment allows to create observation builders to observe the environment for each agent. While it is possible to observe the whole grid, this does usually not make sense due to the fact that many parts of the rail grid are not relevant to a single train. Flatland offers by default two different observation builders.

GlobalObsForRailEnv creates three arrays with the dimensions of the rectangular rail grid. The first array contains the transition information of the rail grid. For each cells, there are 16 bit values, 4 for each possible direction a train is facing.

TreeObsForRailEnv creates a graph with sections of the grid as nodes from the perspective of the train. This means, only the switches which the train is actually able to take define a single node. As an example, a train on a *case 0* switch heading from north to south is not able to make a decision on this switch and therefore, the TreeObservation does not put the sections before and after the switch into two different nodes but just into a single node.
IMAGE mapping TreeObservations
The nodes of the tree observation offer a number of fields that allow to select specific features to create numeric input vectors for function approximator such as neural networks. The tree observation builder offers 14 distinct features for each rail section. This includes:

- Dist. own target encountered: Cell distance to the own target railway station. Inf. if target railway station for agent is not in this section.
- Dist. other agent encountered: Cell distance to the next other agent on this section.
- Dist. to next branch: The length of this section.

- Dist. min to target: The cell distance to the target after this section is finished.
- Child nodes: The nodes the agent is able to take after this section ends. Each child node is associated with a direction (left, forward, right).

3. Approach and methodology

3.1. Basic considerations

As described under section 1.1, our work is based on the work of S. Huschauer [**?**]. We take the idea of using the A3C algorithm to solve the flatland problem and try various modifications in an attempt to improve its performance.
We proceed by giving an intution, what we want to achieve by changing the specified part, followed by an experiment to either prove or disprove our hypothesis.
For training purposes, we started by reimplementing the algorithm by ourselfes. This enabled us from the beginning to gain a deeper understanding of how the algorithm works and where we could find possible areas for improvement. From there, we iteratively added these potential improvements to later compare them against the original version. In this work, we proceed by comparing the final version to versions without these features. It is important to note, that the training process of reinforcement learning and especially multi agent reinforcement learning is hard to evaluate. Depending on the initial weights of the neural networks and the shape of the environments, the performance may vary on each restart. Also, the number of workers can

3.2. A3C implementation for flatland

Originally, the asynchronous advantage actor critic algorithm (A3C) has been designed for use in a single agent environment. By applying it in a multi agent environment, we implicitly convert the environment into a non-stationary environment. While applying A3C in a multi agent setting, the other agents can be viewed as part of the environment. This means, the behaviour of the environment changes while training, due to the fact that the behaviour of the other agents changes.

Gupta et al. [11] finds, that methods like Deep-Q networks (DQN) and Trust region policy optimization (TRPO) are not performing well in a multi agent environment, due to the combination of experience replay and non-stationarity of the environment. We therefore suggest, that it is not recommendable to keep an experience replay buffer with older episodes. Otherwise the sampled experience might represent old agent behaviour which is then learned.

3.3. Enhanced observations

The flatland environment provides a base to build custom observation builders that can be used to create a state representation for the agents. Based on the provided TreeObsForRailEnv (see section 2.2), we implement a custom observation builder which we use to produce an input vector to our neural network. This observation builder takes the the current state of the environment and produces a fixed size numeric vectors with values between 0 and 1 for each agent. This input vector should fulfil an number of requirements:

- Each rail section the agent possibly rides on next should be visible to the agent.
- The agent should be able to detect, whether there is another train coming the opposite direction on any section.
- The agent should be able to detect on each switch which turn is closer to his target.

- On switches, the agent should be able to see if a turn does lead to his target, even if it is not the fastest way. If this is the case, taking this turn might even be a good option to evade possibly blocking situations.
- For the next grid tile, the agent should be able to detect if it is a switch and if so, if it is one the agent can make a decision on. (see section 2.2 for non-usable switches).

3.4. Distrubuted architecture and parallelism

Für die vorliegende Arbeit wurden die unten aufgeführten Programme eingesetzt.

3.5. Action space reduction

Für die vorliegende Arbeit wurden die unten aufgeführten Programme eingesetzt.

3.6. Curriculum learning

3.7. Entropy balancing

In RL, it is of great importance to find the right combination of exploration and explotation. During exploration, the agent explores as much of the state space as possible. This enables the agent to later exploit the found states which are beneficial. Without this exploration, there is a chance that the agent settles on suboptimal policies too quickly and ignores parts of the state space the agent has never seen. The policy is characterized by a probability distribution of actions.

3.8. Agent communication

Für die vorliegende Arbeit wurden die unten aufgeführten Programme eingesetzt.

- (Beschreibt die Grundüberlegungen der realisierten Lösung (Konstruktion/Entwurf) und die Realisierung als Simulation, als Prototyp oder als Software-Komponente)
- (Definiert Messgrössen, beschreibt Mess- oder Versuchsaufbau, beschreibt und dokumentiert Durchführung der Messungen/Versuche)
- (Experimente)
- (Lösungsweg)
- (Modell)
- (Tests und Validierung)
- (Theoretische Herleitung der Lösung)

3.9. (Used Software)

We used the following tools in our project.

Working Environment

- Microsoft Windows 10
- Ubuntu 19.04

Visual Studio Code

- Visual Studio Code 1.40

Documentation

- XeLateX with Visual Studio Code
- XeLateX with WebStorm

Programming language

- Python 3.6

Python modules

- Flatland-rl 1.3 - 2.1.10
- Tensorboard 2.0
- Keras x.x
- Cython x.x
-

3.10. Basic considerations

3.10.1. Round 1

We started with rebuilding the A3C algorithm from S. Huschauer to get a better knowledge how A3C works.
We made some experiments with different observations: TreeObservations and GlobalOberservations. Because we made better and faster progress with GlobalObservatione we continued with those and combind them with a convolutional network. Right after starting this project, we faced a problem regarding the reward distribution.

3.10.2. Round 2

3.11. Measurands

3.12. Experiments

3.13. Solution approach

3.14. Testing and submissions

3.15. Theoretical derivation of the solution

4. Infrastructure

4.1. Infrastructure

We used various computers and servers to train our model. Most of the time did we train on a test environment server of the ZHAW School of Engineering [12]. The server had a total of 56 CPU cores and 721Gb memory. We connect 3 Openstack machines with 8 CPU cores each to this server, to increase our training.
The flatland environment was running on each cpu core and the results where sent to a webserver which updated the model and sent the new neuronal network weights back to the clients.
The reason why we used CPU cores over GPU performance is that the reinforcement algorithm A3C which we used performs better on CPUs instead of GPUs.

4.1.1. Distributed training

We decided during round 1 of the Flatland challenge to change the training of the neuronal network to a distributed approach.
We used the server with the most memory and cores to run a web server, which distributed the neuronal network, the observations and also certain input parameters at training start.
The workers, all other CPUs on which an instance of the Flatland environment was running, received the files and compiled them into c code, to execute them.
The general cycle for each worker is as following:

1. Get current model from web server
2. Execute episode
3. Get updated model from web server
4. Calculate gradient and upload weights

5. Resultate

- (Zusammenfassung der Resultate)

6. Diskussion und Ausblick

- Bespricht die erzielten Ergebnisse bezüglich ihrer Erwartbarkeit, Aussagekraft und Relevanz
- Interpretation und Validierung der Resultate
- Rückblick auf Aufgabenstellung, erreicht bzw. nicht erreicht
- Legt dar, wie an die Resultate (konkret vom Industriepartner oder weiteren Forschungsarbeiten; allgemein) angeschlossen werden kann; legt dar, welche Chancen die Resultate bieten

7. Verzeichnisse

Bibliography

[1] [Online]. Available: https://www.aicrowd.com/challenges/flatland-challenge

[2] S. Huschauer, "Multi-agent based traffic routing for railway networks."

[3] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.

[4] G. Bacchiani, D. Molinari, and M. Patander, "Microscopic traffic simulation by cooperative multi-agent deep reinforcement learning," 2019.

[5] [Online]. Available: http://flatland-rl-docs.s3-website.eu-central-1.amazonaws.com/

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013. [Online]. Available: https://arxiv.org/pdf/1312.5602.pdf

[7] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Online]. Available: https://science.sciencemag.org/content/362/6419/1140

[8] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," 2019.

[9] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, Aug 1988. [Online]. Available: https://doi.org/10.1007/BF00115009

[10] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, ser. NIPS'99. Cambridge, MA, USA: MIT Press, 1999, pp. 1057–1063. [Online]. Available: http://dl.acm.org/citation.cfm?id=3009657.3009806

[11] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems*, G. Sukthankar and J. A. Rodriguez-Aguilar, Eds. Cham: Springer International Publishing, 2017, pp. 66–83.

[12] [Online]. Available: https://www.zhaw.ch/en/university/

List of Figures

List of Tables

(Glossar)

In diesem Abschnitt werden Abkürzungen und Begriffe kurz erklärt.

Abk	Abkürzung
XY	Ix Ypsilon
YZ	Ypsilon Zet

Listings

A. Anhang

A.1. Projektmanagement

- Offizielle Aufgabenstellung, Projektauftrag
- (Zeitplan)
- (Besprechungsprotokolle oder Journals)

A.2. Weiteres

- CD mit dem vollständigen Bericht als pdf-File inklusive Film- und Fotomaterial
- (Schaltpläne und Ablaufschemata)
- (Spezifikationen u. Datenblätter der verwendeten Messgeräte und/oder Komponenten)
- (Berechnungen, Messwerte, Simulationsresultate)
- (Stoffdaten)
- (Fehlerrechnungen mit Messunsicherheiten)
- (Grafische Darstellungen, Fotos)
- (Datenträger mit weiteren Daten (z.B. Software-Komponenten) inkl. Verzeichnis der auf diesem Datenträger abgelegten Dateien)
- (Softwarecode)

Projektarbeit 2019 - HS: PA19_wele_01

Allgemeines:

Titel:	Reinforcement Learning mit einem Multi-Agenten System für die Planung von Zügen
Anzahl Studierende:	2
Durchführung in Englisch möglich:	Ja, die Arbeit kann vollständig in Englisch durchgeführt werden und ist auch für Incomings geeignet.

Betreuer:

HauptbetreuerIn: Andreas Weiler, wele

NebenbetreuerIn: Thilo Stadelmann, stdm

Zugeteilte Studenten:

Diese Arbeit ist zugeteilt an:
- Ralph Meier, meierr18 (IT)
- Dano Roost, roostda1 (IT)

Fachgebiet:

DA	Datenanalyse
DB	Datenbanken
SOW	Software

Studiengänge:

IT	Informatik

Zuordnung der Arbeit :

InIT	Institut für angewandte Informationstechnologie

Infrastruktur:

benötigt keinen zugeteilten Arbeitsplatz an der ZHAW

Interne Partner :

Es wurde kein interner Partner definiert!

Industriepartner:

Es wurden keine Industriepartner definiert!

Beschreibung:

Reinforcement Learning ist der Zweig des maschinellen Lernens, der sich damit beschäftigt, in einer gegebenen Umgebung durch Interaktion automatisch herauszufinden, was das beste "Rezept" (die sog. "Policy") ist, um ein bestimmtes Ziel zu erfüllen. In jüngster Zeit erregten grosse Erfolge der Methodik im automatischen Gameplay (Dota2, QuakeIII, Atari, Go, ...) einiges an Aufsehen. Aber wie die monatlichen Treffen des "Reinforcement Learning Meetups Zürich" zeigen (https://www.meetup.com/de-DE/Reinforcement-Learning-Zurich/), gibt es auch immer mehr vielversprechende Anwendungen in Industrie und Wirtschaft.

Die Hauptfrage bei dieser Arbeit ist: Wie können Züge lernen, sich automatisch untereinander zu koordinieren, um die Verspätung der Züge in grossen Zugnetzwerken zu minimieren. Die Betreuer dieser Arbeit haben bereits eine enge Zusammenarbeit mit der SBB zu diesem Thema aufgegleist, die als Grundlage den gerade gemeinsam ausgeschriebenen KI Wettbewerb "Flatland Challenge" hat (siehe Link unten). In dieser Projektarbeit geht es darum, einen (Deep) Reinforcement Learning Ansatz für Flatland zu implementieren und zu evaluieren.

Informations-Link:

Unter folgendem Link finden sie weitere Informationen zum Thema:
https://www.aicrowd.com/challenges/flatland-challenge

Voraussetzungen:

- Spass an der Arbeit mit Daten und Data Science Tools
- Starkes Interesse am Thema Künstliche Intelligenz, insbesondere Reinforcement Learning
- Sehr gute Programmierfähigkeiten (Python-Kenntnisse können im Projekt erworben werden)
- Pragmatisches und systematisches Vorgehen beim Experimentieren und genauen Auswerten
- Freude am wissenschaftlichen Arbeiten und den ersten eigenen Versuchen in angewandter Forschung

Die Betreuer haben viel Freude am Thema und mehrere Ideen zum Starten auf Lager; sie freuen sich auf leistungsfähige Studierende und ggf. (bei guten Resultaten) eine gemeinsame wissenschaftliche Publikation aus der Zusammenarbeit.