2021 Flatland Challenge, Team Learn

Richard Hájek, Ladislav Maleček, Matouš Melecký, Jaroslava Schovancová

Abstract

In this paper we report on experiments performed to address the 2021 Flatland Challenge with Deep Reinforcement Learning approaches. Leveraging the flatland-rl library [6], we studied a Multi-Agent Reinforcement Learning (MARL) system with different number of agents, evolution epochs, and depth of the observation tree, and concluded that the longer evolution and lower number of agents, the better for the completion rate and the scores of the simulation. With these consideration in mind we were able achieve completion rates 0.35 to 0.99, and scores -0.7 to -0.13.

Introduction

The Flatland Challenge [6] tackles a principal challenge of the transportation world: *How to efficiently manage dense traffic on complex railway networks?* As this is a real-world problem that many transportation and logistics companies face, novel approaches have the potential to shape the way traffic management systems are implemented around the world.

While taking part in the *NAIL052 Seminar on Artificial Intelligence 2* course, we chose to approach the challenge using the deep reinforcement learning techniques, exploring the Single-Agent Reinforcement Learning (SARL) and Multi-Agent Reinforcement Learning (MARL) approaches, notably focusing on MARL algorithms for cooperative settings, in particular the Multi-agent Actor Critic approach.

Our findings, particularly in the collaborative MARL area, are summarized in this paper, structured as follows: in Section *Single-Agent Reinforcement Learning* we summarize the SARL approach. In Section *Multi-Agent Reinforcement Learning* we provide a brief overview of the MARL approach, focusing particularly on the Actor Critic algorithm. In Section *Considerations* we summarize what we learned from reviewing the approaches taken by the winners of the 2020 Flatland Challenge, pointing out the considerations we made to prepare a series of experiments, that are summarized in Section *Grid Scan of a Parameter Space*. In Section *Summary* we summarize our findings.

Acronyms

Throughout the paper we use the acronyms listed in Table 1.

RL	Reinforcement Learning
SARL	Single-Agent Reinforcement Learning
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
DQN	Deep Q-Networks
TD	Temporal Difference
MADDPG	Multi-Agent Deep Deterministic Policy Gradient
PPO	Proximal Policy Optimization

Table 1: Acronyms.

Single-agent Reinforcement Learning

The task of single-agent reinforcement learning is to train an agent that would be able to act within some environment it perceives while at the same time maximize some predefined reward function. The agent should learn how to act through trial and error.

The agent perceives the state of the environment through a set of observations and receives reward signals. Usually, the time is discrete as continuous time poses additional challenges. Traditional solutions of the reinforcement learning work on basis of some dynamic programming and iteratively operate over a simple data table, that captures its current reward function. However, these approaches struggle when the environment becomes more complex and do not scale well. Due to this, the field of deep reinforcement learning has recently emerged, where deep neural networks are used as function approximators instead of the simple table storing the estimated reward function.

The problem of navigating a train, that lies at the core of the Flatland challenge, can be well modelled as a single-agent reinforcement learning problem. The agent is the train in the network and it observes some local subsection of intersections and acts upon reward signals provided by some heuristics, e.g. distance to its goal. The extension of the single-agent-reinforcement learning, described in Section *Multi-Agent Reinforcement Learning*, can further be used to solve the whole flatland challenge problem of multiple trains navigating through a common environment.

In the next Section *Formalization and Key Concepts* the formalization of the problem is presented and key concepts are introduced. Then, in Section *Algorithms*, a brief overview of reinforcement learning algorithms is presented.

In the following sections, the SARL will be referred to as simply reinforcement learning, or RL.

Formalization and Key Concepts

The formalism traditionally used to describe the environment of the reinforcement learning is the Markov Decision Process (MDP). It is a tuple (S, A, R, P, ρ_0), where

1. S is the set of states
2. A is the set of actions
3. $R : S \times A \times S \to \mathbb{R}$, noted as $r_t = R(s_t, a_t, s_{t+1})$ is the reward function, that depends on the current state of the world, the action taken and the resulting state.
4. $P : S \times A \to P(S)$ is the transition probability function
5. ρ_0 is the starting state distribution.

The state $s \in S$ is a complete description of the world. The agent can observe the whole state, in which case the environment is fully observable, but also it can be constrained to only a subset of the state, making the environment partially observable.

The policy function $\pi : S \to A$ is the mechanism through which the agent makes decisions. The policies in deep reinforcement learning can be implemented as a deep neural network. The policy can be deterministic or stochastic. A trajectory T is a sequence of states and actions in the world. Return G of the agent is the sum of rewards obtained during time. In the case of infinite-horizon reinforcement learning, meaning that the number of steps can be infinite, the reward is discounted by some factor $\gamma < 1$ dependent on the time:

$$G = \sum_{t=0}^{\infty} \gamma^t r_t. \tag{1}$$

Key information the agent needs to decide which actions to take is provided by the value functions. A value of a state or state-action pair is the expected return obtained by starting in the said state-action pair and then following the policy π.

The on-policy action-value function (also called the Q-function) can be defined as

$$Q^\pi(s, a) = E_\pi[G|s_0 = s, a_0 = a]. \tag{2}$$

The value function $V^\pi(s)$ is similar, however, it only depends on the starting state. The optimal action-value function Q^* is the expected return if the agent follows an optimal policy, meaning that it is the maximum of the of all possible on-policy action-value functions. The optimal value function is defined in a similar manner. It is evident that if the Q^* is known, the optimal action can be obtained from it directly.

Advantage function $A^\pi(s, a)$ can be defined as

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s). \tag{3}$$

It expresses the relative advantage of the particular action a in the state s compared to the action selected by the policy.

The Bellman equation is used to compute the optimal action-value function Q^*

$$Q^\pi(s_t, a_t) = E_{s' \in S}[r(s, a) + \gamma \max_{a'} Q^\pi(s', a')]. \tag{4}$$

Algorithms

As shown in Fig. 1, the RL algorithms can be classified into two main groups: model-free and model-based. The model-based algorithms learn the full model of the world, while the model-free algorithms side-step the model by directly learning the Q-function or policy. More commonly, the model-free approach is employed, as estimating the whole model is often implausible.

Q-Learning learns an approximator for the optimal action-value function $Q^*(s, a)$. This is accomplished by optimization based on the Bellman equation. The policy can be inferred directly from the Q-function. During training, an ϵ-greedy policy is used. This means that, for some parameter $0 < \epsilon < 1$, the agent picks the action as provided by the so-far learned policy with probability of ϵ and with probability $1-\epsilon$ it performs some other random action. This helps to balance the exploitation-exploration problem. In the beginning, ϵ is low, as the initial policy is usually random. It is then assumed that the policy improves during the training and the ϵ is progressively increased. Often, experience buffer is used, hence the experience can be reused and randomly sampled. Therefore, this algorithm is off-policy, meaning that the current policy is different from the one used when training (past experiences that are used while training were generated using previous versions of the current policy). DQN [5] is a typical example of a Q-learning method. It has been one of the first algorithms that used deep neural networks in the RL.

Policy optimization techniques directly approximate the policy function. The parameters of the policy are searched for using gradient ascent on the advantage function. This approach is on-policy and it can easily estimate both deterministic and stochastic policies. One example of a policy optimization algorithm is REINFORCE.

Actor-critic approaches combine the policy optimization with Q-Learning. The actor is an approximator that learns the policy and effectively controls how the agent behaves. The critic evaluates the action by computing the value function. The training of critic and actor is performed separately. One of the prominent examples of an actor-critic approach is the Advantage Actor Critic (A2C) [4].

Multi-agent Reinforcement Learning

Scaling up from Single-agent RL to Multi-agent RL, the action spaces become joined action states, with one dimension per agent. The MARL approach can be challenging to grab: we cannot use a simple-agent RL algorithm for each agent, as with the single-agent we assume that the transition probabilities remain constant. However, with multiple agents they can change, therefore we need to use a joined transition table. This breaks or invalidates the basic framework of most theoretical analyses in the single-agent setting (but it is not necessarily a deal breaker for more advanced algorithms).

In case we used a single policy to manage all agents, we would experience an exponential growth of action space with respect to the number of agents, which is known as combinatorial nature of MARL.

Focusing on the decentralization vs. centralization aspect of the agent behaviour, if we lean towards full decentraliza-

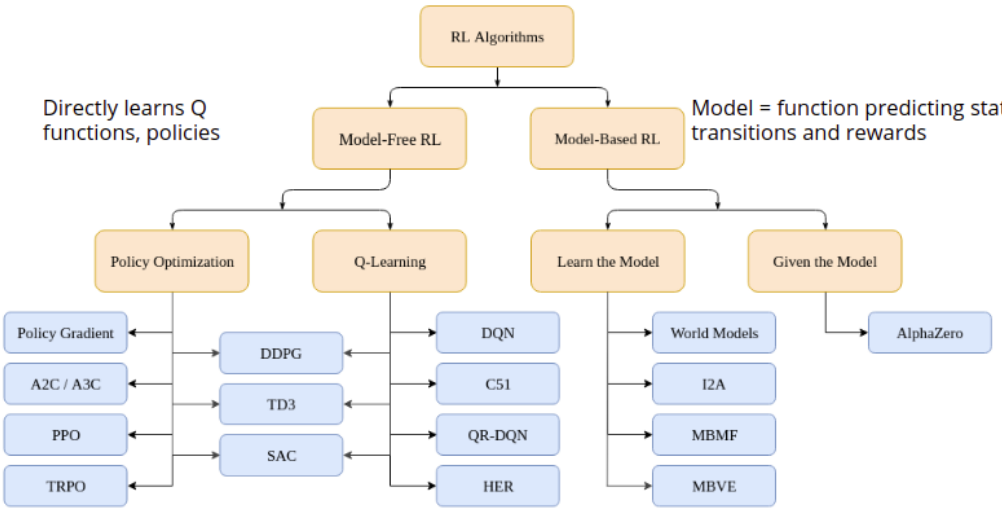

Figure 1: Taxonomy of reinforcement learning algorithms. Courtesy of [1].

tion, we would have to tackle the non-stationarity, as there is a need for communication between agents in order to mitigate the limited observability of a single agent. If we lean towards fully centralized approach, we would have to fight the scalability, but the agent would have had a full observation and could have acted upon it without any communication issues.

Additionally, with a MARL system we need to find a proper balance to address instability of experience replay, increased variance in policy gradients, as well as tackling data efficiency due to the curse of dimensional sparsity.

MARL – Overview

MARL Systems and the Cooperation Distinction Review [9] discusses three different MARL information structures: fully cooperative, fully competitive, and the mix of the two.

In a fully cooperative system, the agents collaborate to optimize a common long-term return. Learning goals in MARL are multidimensional, and objectives of solo agents do not necessarily align. In a fully cooperative MARL system, the reward function is shared among all the agents. Recognised variations may be

- *Common accumulated reward* – it allows the use of single-agent RL with identical value and Q function.
- *Team-average reward* – the agents are allowed to have different reward functions which can be kept private, but the goal for cooperation is to optimize the long-term average reward model.

The team-average reward approach probably is not exactly what we look for, due to the agents needing to be able to handle many different maps & situations. One of the possible considerations may be to define different reward functions for the slower and the faster trains.

The common accumulated reward is a special case of team-average reward approach. It requires incorporation of communication protocol, as it cannot predict the behavior of agents with different and hidden reward function (policy).

MARL Systems and the Centralization Distinction A fully decentralized approach suffers from observability of only the local action and reward, and thus from non-convergence in general. Majority of cooperative MARL settings involve homogenous agents with a common reward function that aligns all agents interests.

Highlighting applications of cooperative setting is robot team navigation, mostly decentralized setting with network agents sharing some personal information/observations.

It is essential to make the distinction based on centralization or cooperation to hold for the execution. Training can be, and often is, done in the centralized fashion, e.g. action-critic approaches that will be mentioned later in this section.

MARL Algorithms for Cooperative Setting

Review [3] discusses these three MARL algorithms for cooperative settings: Q-Learing and Deep Q-Networks (DQN), Policy Gradient Algorithms, and Multi-Agent Actor Critic.

Q-Learing and Deep Q-Networks The Q function is being periodically updated with the most recent parameters, can, and often does, use bootstrapping, in order to stabilise learning, benefiting also from the replay buffer technique, one can store previous "situations", transitions.

The Q-Learning can be directly applied to multi-agent settings, by separating each agent and learning its own Q-function. Nevertheless, due to the non-stationarity, due to multiple agents, it can hinder convergence, as it violates the Markov assumption required for convergence. Another difficulty arises from the use of a replay buffer, since the transition probabilities change in a non-stationary environment.

Policy Gradient Algorithms Most of the advanced techniques use Temporal Difference (TD) approach, where the bootstrapping uses just some of the previous steps, and the Q value function defined from the current policy, using the gradient directly on the policy itself, and always moving in the direction of the current observed situation.

The inner workings of the algorithm leads naturally to high variance gradient estimates. We suppress this to a certain degree in single-agent settings by utilizing baseline estimators, but that proved to be difficult in multi-agent settings due to the non-stationarity.

Multi-Agent Actor Critic One of the criteria taken into account in Multi-agent Actor Critic approach is the locality of information: the learned policies use solely the local information, and do not assume a differentiable model of the environment (all advanced settings), neither do assume any particular structure of communication, which does not need to be differentiable. The lead idea of Actor Critic algorithm is the use of centralized training with decentralized execution. This approach renders Q-Learning unnatural to use, because the Q function cannot infer the same information from two very different inputs. But action-critic methods are well suitable due to the critic being used only in training.

The gradient for N agents parametrized by θ_i and policies π_i reads

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s\sim p^{\boldsymbol{\mu}}, a_i \sim \boldsymbol{\pi}_i} \left[\nabla_{\theta_i} \log \boldsymbol{\pi}_i(a_i|o_i) Q_i^{\boldsymbol{\pi}}(\boldsymbol{x}, a_1, \ldots, a_N) \right], \tag{5}$$

where the $Q_i^{\boldsymbol{\pi}}(\boldsymbol{x}, a_1, \ldots, a_N)$ is a centralized action-value function that takes as input actions of all agents $a_1, \ldots, a_N$, in addition to some state information x. Q functions for each agent are learned separately, which results in a possibility of having an arbitrary reward function. We can further extend the above idea to work with deterministic policies, together they are called multi-agent deep deterministic policy gradient (MADDPG).

The primary motivation behind the MADDPG is the suppression of non-stationarity. The policies change but not in regards to the global critic function, which sees the environment as stationary, which contributes to the stabilization of the training.

MARL in Flatland context

We have mentioned common approaches in MARL and now we will specify how we can deploy these techniques to our very specific domain.

State and action representation

In Flatland, the environment is represented by a 2D grid array of arbitrary size, where each cell represents one place on the grid. Cell specifies one of few possible situations, either there is nothing, a station or one of many types of train tracks. We can see the representation on Figure 2.

The naive approach is feeding the RL algorithm the grid without any processing. Such approach results in a solver with a limited size of the input and with low transfer-ability between different maps. In order to alleviate the problem, the grid transformed into an observation tree. Observation tree is a representation of the immediate surroundings of the agent. The size of this observation tree depends on the training hyper-parameters. The tree consists of crossing types, lengths between crossings, train station types.

Actions are represented in a simple go left, go right, continue straight bases. We only allow agents to make an action when they reach a junction. This is a side effect of discretization the problem in time from continuous real time to discrete steps between decisions.

Other rules that are incorporated into the environment are: only one train can be present on a cell, each train has its starting and finishing position, trains cannot turn around on a single cell.

Reward

All of the aforementioned algorithms require a reward function. The possible methods of rewarding agents are plenty, for example, punish the agents if they get stuck by colliding into each other, reward them for successfully reaching the final state. Furthermore specific reward function are mentioned in the next section.

Transferability

One of the desired properties is being able to apply the already trained agents to many different maps and situations, preferably being able to add and remove custom amount of agents. To our best knowledge this is mostly achievable only for decentralized approaches where a single agent is controlled independently from others. The decentralization here means only that the RL algorithm is fed only a state of one agent, the weights can possibly be shared. With usage of the decentralized approach are only be limited by the amount of space on the map for each agent and some inherently learned behavior that expect some specified agent density.

Considerations

During the survey phase of the project we have studied approaches of the winning teams in the 2020 and 2019 competitions, notably the JBR_HSE team's approach [2], and the Netcetera's ai-team-flatland [8].

JBR_HSE approach

The key lessons we took from the JBR_HSE approach follow:

- domain-specific simplifications of the task greatly improve the results: agent cannot make a decision when not at an intersection, and agent's episode ends after it reached the goal or got itself into a deadlock (while the default behaviour was continuation along the path),
- two observations have been used: a local tree observation with fixed depth, where each edge and the vertex to which it leads has a vector of features calculated for a given section of the path – e.g. the length of the path, the distance from the end of the edge to the destination, etc. The signs of the minimum distance to the destination, noting whether the agent is at the intersection or in front of it, the agent got into the deadlock, etc.

- each agent is assigned an ID,
- reward function weighting in deadlocks and successful accomplishments:

$$r_t = 0.01 \star \Delta d - 5 \star \text{is\_deadlocked} + 10 \star \text{succeeded}, \quad (6)$$

  where Δd denotes a change of distance to the goal since the decision has been made.
- used a Proximal Policy Optimization (PPO) algorithm [7] with Actor Critic approach. The Critic approximates the value function, while the Actor learns a randomized policy that maximizes the expected value of total reward,
- the team has added communication mechanism, where agents generate messages and send them to all neighbouring trains, and any received messages are averaged into a single message. A self-attention mechanism was used to determine the weights with which the messages will be averaged,
- one particularly interesting idea we noticed was observation that most deadlocks occurred due to the same start times of the trains. Therefore, the team took actions so that only some agents are picked to start their journey, and only when one of them finishes, the next agent is allowed to start its journey. At the training time, agents are picked based on how close they are to their goal. This approach performed well for small environments. During the application time, classifier is trained to determine probability of reaching the goal – agents wait until the probability becomes sufficiently large.

ai-team-flatland approach

The key lessons we took from the ai-team-flatland approach [8] follow:

- the whole environment has been represented as a map of switches with directions, i.e. a directed graph,
- the team tried different RL approaches, however, without further specifications of which approaches have been tried,
- in the 1st round, the base implementation leveraged a DQN
- reward function considerations: the team added more information about the future deadlocks, and penalization for getting into one; they decreased negative reward per step, and added a penalty for stopping and for invalid actions, included valid actions in the observation vector,
- the team propagated information about tracks behind an agent, in order to avoid agent blocking other agents behind it,
- the team applied a voting system resembling an ensemble system: several models trained, that worked in different states and environments, next action was the most frequent action proposed by models.
- in the 2nd round, the team ran with a sparser, bigger network with more trains, with different speeds and introduced malfunctions,
- the team identified several problems: as it is easy to get into a deadlock, the outcome rarely gains positive reward; deadlocks were more costly, e.g. a train blocked the only route in the local area; local observations were limiting, as an agent got information about other agents too late,
- the team considered a prioritized approach as a way to solve conflicts between agents: a conflict occurred for active agents e.g. when two or more agents wanted to occupy the same cell, or in a case of a swap conflict, when they wanted to surpass each other from different directions.
- the team decided to build a un-directed conflict graph, where the edges represent conflits. By assigning priority to the agents, they converted the conflict problem to a graph coloring problem, where two agents in a conflict cannot have the same priority,
- applying a heuristic, they gave a higher priority to the agents with the highest degree, i.e. highest number of conflicts, and checked that the blocked agent cannot have the highest priority,
- for agents that were ready to depart, they chose the fastest agents that have a free path,
- the team ensured global observation, i.e. supplemented the agents with information about the whole path to goal.

Grid Scan of a Parameter Space

We implemented a toy model of a MARL system in cooperative mode with action critic algorithm, in order to better compare and understand the benefits of different approaches. However, we focused in particular on a grid scan of a parameter space that we describe below.

For our experimentation we adopted the flatland-rl library [6] enhanced by a simple DQN algorithm, which serves as a baseline application for the Flatland challenge, prepared by the organizers. Our main objective was to understand its behavior under different conditions.

We identified parameters for our study: the number of agents, the number of evolution epochs, and a depth of the observation tree that we fed into the network. With these parameters, which define the simulation parameters, we performed a grid scan of the phase space, and compared the completion rate and the score of each simulation run.

We ran the simulation for $N \in \{1, 2, 4, 8\}$ agents, with different number of epochs $E \in \{1, 32, 64, 128, 256, 512\}$, and the depth of the observation tree $d \in \{2, 4\}$. The completion rates and scores for the simulations with different N, E, and d are shown in Table 2.

From both the completion rate plots, and the scores plots, it seems that

- the lower the number of agents N, the better,[1]
- the higher the number of epochs of the training, the better,
- for the choice of the depth of the observation tree $d \in \{2, 4\}$, we see only a little difference, where the smaller d seems to have converged slightly faster.

[1] Completion rate closer to 1 is better, while scores closer to 0 are better.

Each simulation produces a dynamic visualisation of the activity of the trains. In Fig. 2 we show a snapshot of this map for a simulation with 2 trains, and in Fig. 3 with 8 trains. The higher the number of trains (agents), the larger the map.

We have not performed a systematic study of simulations with $N > 8$, as from our initial grid scan it seemed that it would not bring much benefit, in terms of completion rate and the scores dependence.

Summary

When approaching the Flatland challenge, we have reviewed options which Single- and Multi-Agent Reinforcement Learning systems can offer to study a transportation system. We have studied approaches taken by the 2020 Flatland challenge winners.

We implemented a toy model of a cooperative Multi-Agent Reinforcement Learning (MARL) system with action critic algorithm. And with the flatland-rl library integrated with a DQN algorithm, we studied a Multi-Agent Reinforcement Learning system with different number of agents ($N \in \{1, 2, 4, 8\}$), evolution epochs $E \in \{1, 32, 64, 128, 256, 512\}$, and depth of the observation tree $d \in \{2, 4\}$, and concluded that the longer evolution and lower number of agents, the better for the completion rate and the scores of the simulation. In our simulations, taking into account these observations, we achieved completion rates 0.35 to 0.99, and scores -0.7 to -0.13. We briefly reviewed the dynamic visualizations of maps of the simulations.

Acknowledgements

Computational resources for this project were in part supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

Contributions

Jaroslava Schovancová

Team leader and organiser; past approaches research; report co-author & editor.

Ladislav Maleček

Reinforcement learning research and programming; report co-author & editor.

Matouš Melecký

Related work and past approaches research; report co-author.

Richard Hájek

Programming and parameter optimization; experimental results; report co-author.

References

[1] Joshua Achiam. Spinning up in deep reinforcement learning, 2018. URL: https://spinningup.openai.com/en/latest/index.html.

[2] Florian Laurent, Manuel Schneider, Christian Scheller, Jeremy Watson, Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Konstantin Makhnev, Oleg Svidchenko, Vladimir Egorov, Dmitry Ivanov, Aleksei Shpilman, Evgenija Spirovska, Oliver Tanevski, Aleksandar Nikov, Ramon Grunder, David Galevski, Jakov Mitrovski, Guillaume Sartoretti, Zhiyao Luo, Mehul Damani, Nilabha Bhattacharya, Shivam Agarwal, Adrian Egli, Erik Nygren, and Sharada Mohanty. Flatland competition 2020: Mapf and marl for efficient train coordination on a grid world, 2021. http://arxiv.org/abs/2103.16511 arXiv:2103.16511.

[3] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2020. http://arxiv.org/abs/1706.02275 arXiv:1706.02275.

[4] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL: http://proceedings.mlr.press/v48/mniha16.html.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. http://arxiv.org/abs/1312.5602 arXiv:1312.5602.

[6] Sharada Mohanty, Erik Nygren, Florian Laurent, Manuel Schneider, Christian Scheller, Nilabha Bhattacharya, Jeremy Watson, Adrian Egli, Christian Eichenberger, Christian Baumberger, Gereon Vienken, Irene Sturm, Guillaume Sartoretti, and Giacomo Spigler. Flatland-rl : Multi-agent reinforcement learning on trains, 2020. http://arxiv.org/abs/2012.05893 arXiv:2012.05893.

[7] John Schulman, Oleg Klimov, Filip Wolski, Prafulla Dhariwal, and Alec Radford. Proximal policy optimization, 2020. URL: https://openai.com/blog/openai-baselines-ppo/.

[8] Evgenija Spirovska. Leverage reinforcement learning for building intelligent and adaptive trains that can successfully navigate a railway, 2020. URL: https://blog.netcetera.com/leverage-reinforcement-learning-for-building-intelligent-and-adaptive-trains-that-can-successfully-9f4cdef80162?gi=ac3088425cec.

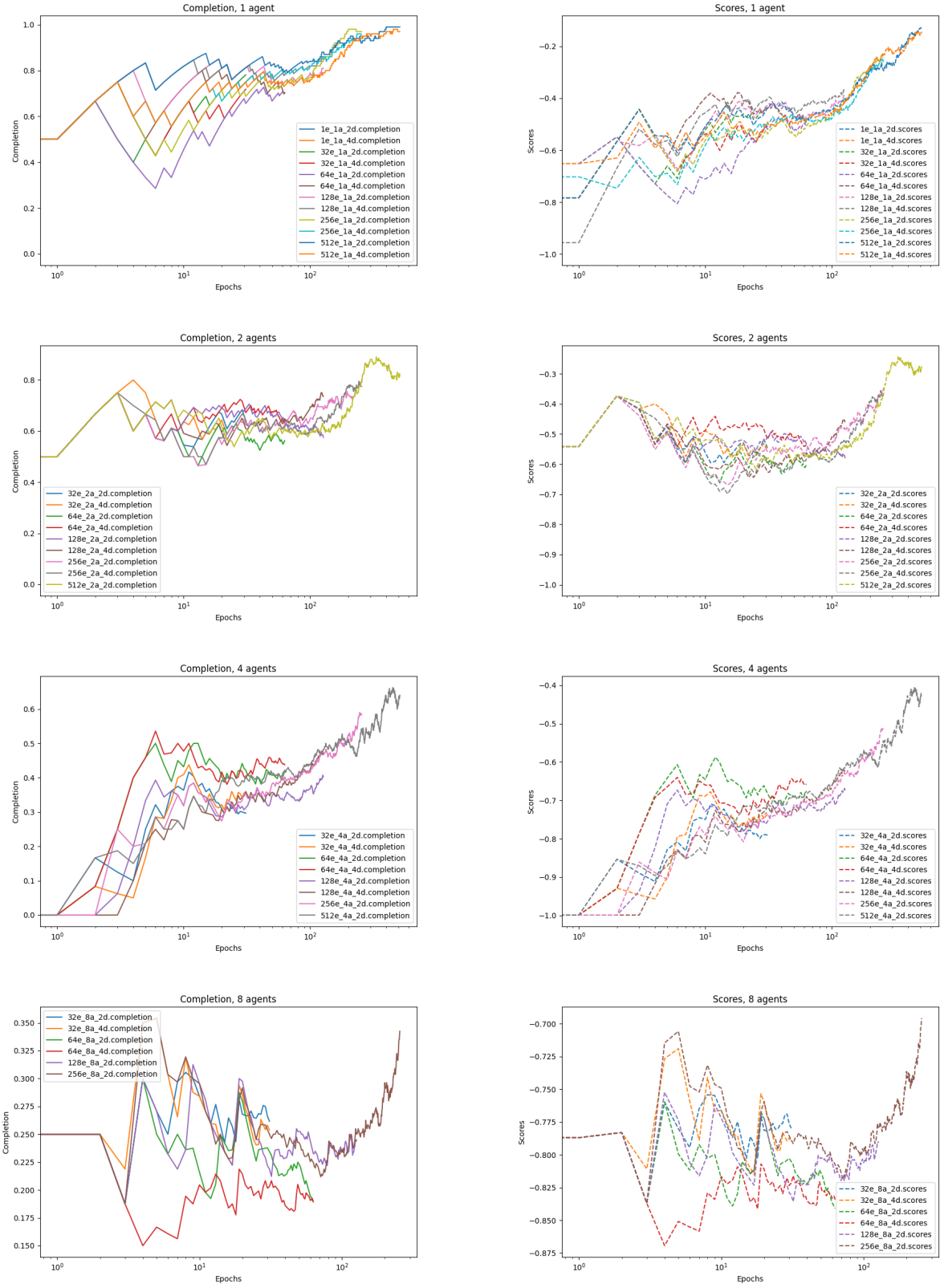

Table 2: Grid scan results of the multi-agent simulations, for $N \in \{1, 2, 4, 8\}$ agents, with different number of epochs $E \in \{32, 64, 128, 256, 512\}$, and different depth of the observation tree $d \in \{2, 4\}$. In the left column we show the completion rate, in the right column the score.

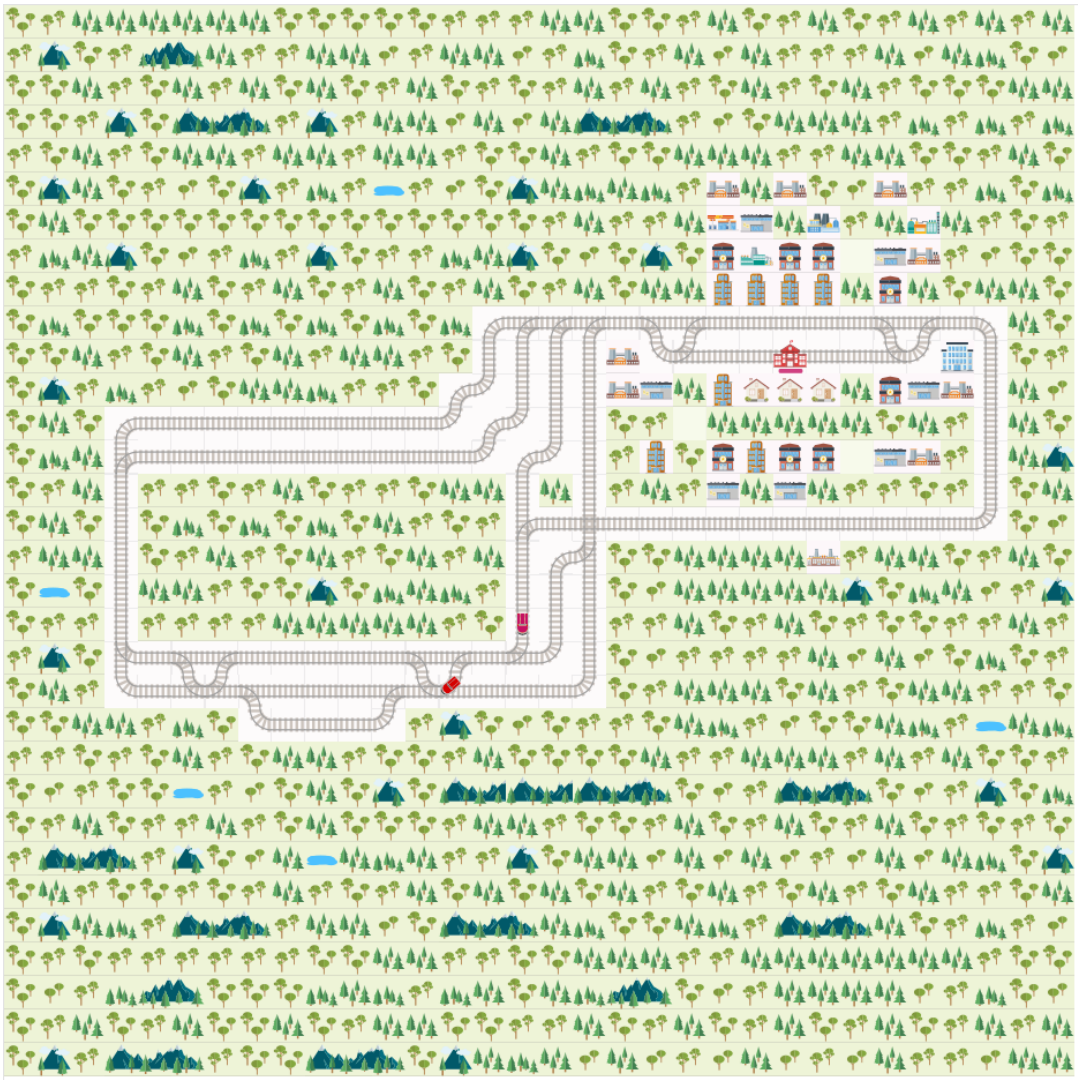

Figure 2: Map of a simulation with 2 trains.

[9] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms, 2021. http://arxiv.org/abs/1911.10635 arXiv:1911.10635.

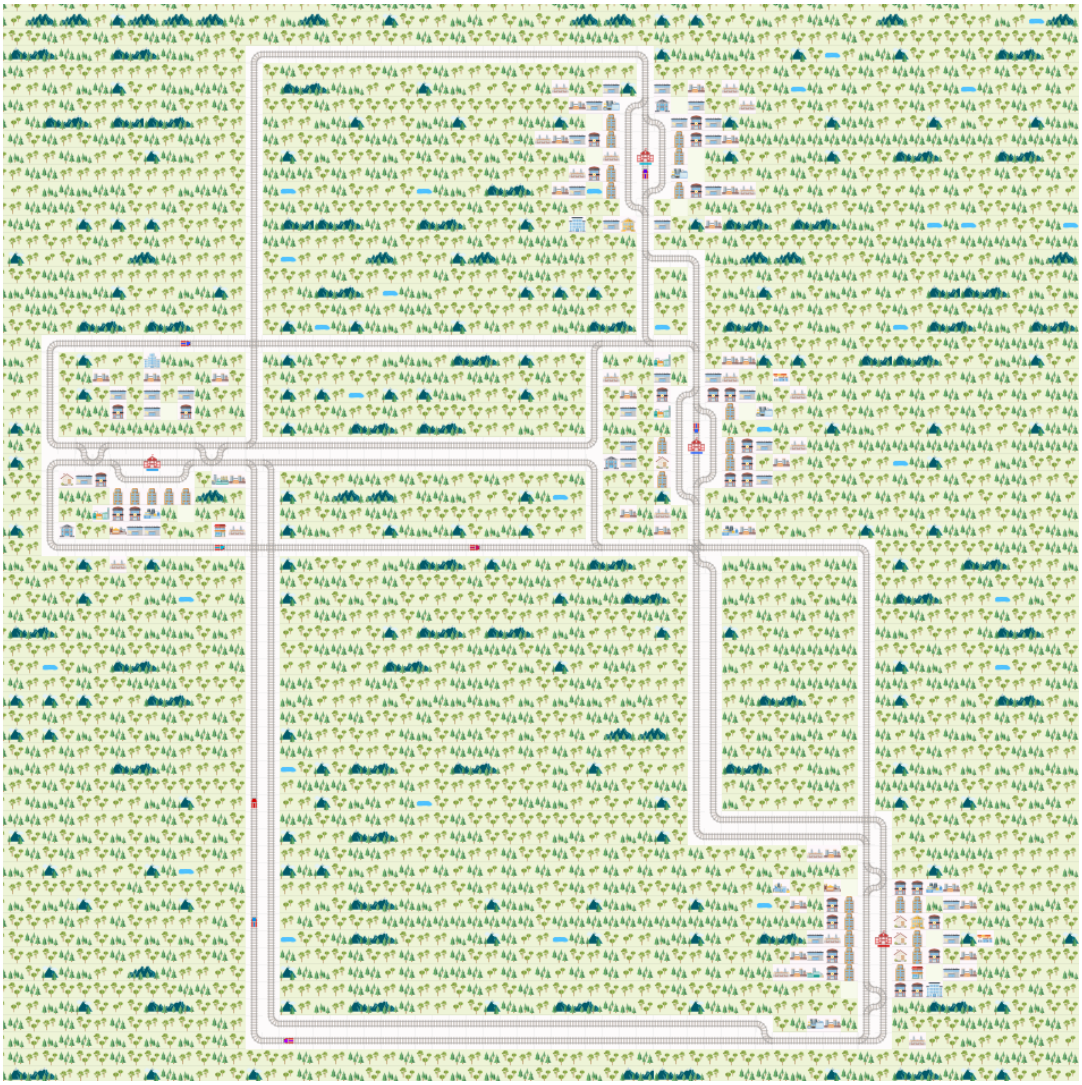

Figure 3: Map of a simulation with 8 trains.