Implementation of Deep Reinforced Q-Learning Methods in Framing of Vehicle Rescheduling Problem

Universita' di Bologna
Faculty of Engineering and Architecture
Flatland Challenge

Moritz Valentino Huber - moritz.huber@studio.unibo.it
Andrea Pinto - andrea.pinto@studio.unibo.it

Professor	Andrea Asperti
Course	Deep Learning
Project	Flatland Challenge
Location and Date	Bologna, July 18, 2021

Abstract

The Flatland Challenge (FC) is a competition organized by AIcrowd in corporation with Swiss Federal Railways (SBB) to address the problem of train scheduling and rescheduling by providing a simple grid world environment and allowing for diverse experimental approaches. The SBB operates the densest mixed railway traffic in the world and maintains the biggest railway infrastructure in Switzerland. The goal of the FC is to make all the trains considered in the grid world arrive at their target destination with minimal travel time. Hence, the the number of steps that it takes for each agent to reach its destination shall be minimized.

In this report the use of Dueling Deep Q-Network (Dueling-DQN) is considered, in the framing of a Reinforcement Learning (RL) based approach to tackle the stated vehicle rescheduling problem. Dueling Deep Q-Networks split the information stream within the network into the state-value function and the action advantages, that ultimately get combined in the Q-function. The considered (Artificial) Neural Network (NN) has five outputs for the five potential actions that the agent can ultimately take.

There are various settings to be executed: Single, Three or Five Agent Scenario. The Single Agent Scenario almost immediately shows good results. Final adjustments then lead to an overall performance (score) of 95%. In the Multi Agent Scenario however, achieving high scores is not as easy. In the three agent setting a default overall score of $\approx 75\%$ can be achieved, which is still satisfying in comparison ti the five agent scenario (default score 56%). The major contribution to the lowered results is identified to be deadlocks arising from malfunctions or bad decisions of the agent. Hence, modified rewards for the five agent scenario are considered. Implementing a deadlock punishment for moving agents helps to increase the overall score up to $\approx 65\%$.

Contents

List of Figures

Akronyms

CNN	Convolutional Neural Networks
DQN	Deep Q-Network
Dueling-DQN	Dueling Deep Q-Network
FC	Flatland Challenge
MARL	Multi Agent Reinforcement Learning
MDP	Markov Decision Process
NN	(Artificial) Neural Network
RL	Reinforcement Learning
SBB	Swiss Federal Railways
SGD	Stochastic Gradient Decent

1 Introduction

The FC is a competition organized by AIcrowd [Moh+20] in corporation with SBB to address the problem of train scheduling and rescheduling by providing a simple grid world environment and allowing for diverse experimental approaches. The SBB operates the densest mixed railway traffic in the world and maintains the biggest railway infrastructure in Switzerland. Today (2021) there are more than 10k trains running each day, being routed over 13k switches and controlled by more than 32k signals.[Moh+20]
The goal of the FC is to make all the trains considered in the grid world arrive at their target destination with minimal travel time. Hence, the the number of steps that it takes for each agent to reach its destination shall be minimized.
A potential approach to the state problem is implementing a RL architecture to teach the considered vehicles how to schedule or reschedule themselves. Reinforcement Learning is defined as follows [Sze10]:

> *Reinforcement learning is a learning paradigm concerned with learning to control a system so as to maximize a numerical performance measure that expresses a long-term objective. What distinguishes reinforcement learning from supervised learning is that only partial feedback is given to the learner about the learner's predictions. Further, the predictions may have long term effects through influencing the future state of the controlled system. Thus, time plays a special role. The goal in reinforcement learning is to develop efficient learning algorithms, as well as to understand the algorithms' merits and limitations. Reinforcement learning is of great interest because of the large number of practical applications that it can be used to address, ranging from problems in artificial intelligence to operations research or control engineering.*

In the following the approach to takle single and multi agent scenarios is described. Beginning with the depiction of the provided Flatland environment.

2 Flatland Fundamentals

2.1 Flatland Grid World

In this section the two-dimensional grid world of Flatland is presented. The world itself contains different building blocks: empty (environment) and railway blocks. Figure 2.1.1 shows a potential Flatland grid world.

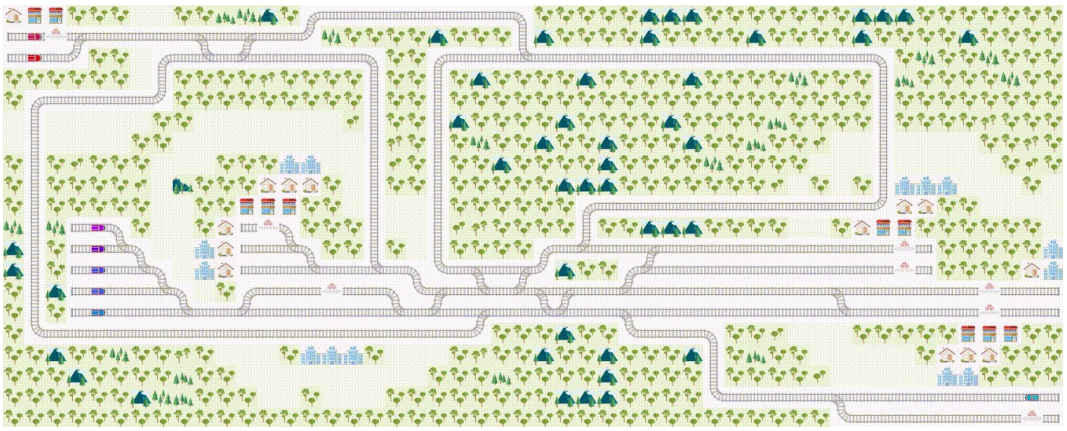
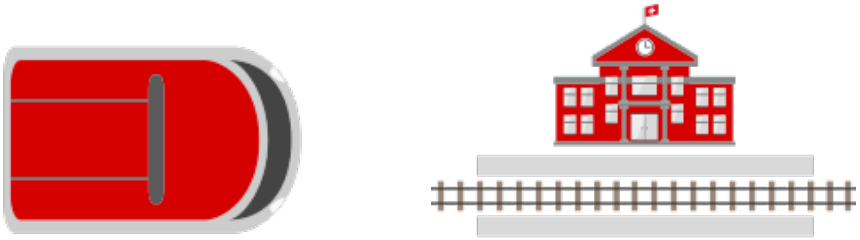


Figure 2.1.1: Flatland Grid World, reference: [Moh+20]

Environment blocks are depicted as forest or mountains and have no function in the vehicle rescheduling problem. Railway blocks are depicted as such (grey) and define the actions that can be taken by the agent. The direction of the railway blocks and hence the orientation of the agent is defined in terms of North for *Up*, East for *Right*, South for *Down* and West for *Left*.

2.2 Agents and their Destination

Since Flatland is a discrete time simulation (all actions performed happen with a constant time step), the agents can choose an action at each time step. The term agent (see Figure 2.2.1 a) defines an entity that can move within the grid and must solve tasks.



(a) Agent (b) Destination

Figure 2.2.1: Agent and Destination

An agent (train) does basically two things: wait or go into a particular direction. There are different models of trains with different properties: freight train and passenger train

(they have different speeds $v_{agent} \in (0, 1]$). An agent can move in any arbitrary direction, if the environment permits it. Transitions from one cell to the next are only valid in neighboring cells. Cells can be considered neighboring if the Euclidean distance between them is $|\vec{x_i} - \vec{x_j} <= \sqrt{2}|$. Additionally, a railway cell can only hold one agent at a time. If the agent chooses a valid action, the corresponding transition will be executed and the train's position and orientation is updated. Each agent has its individual start and destination (see Figure 2.2.1 b). [Moh+20]

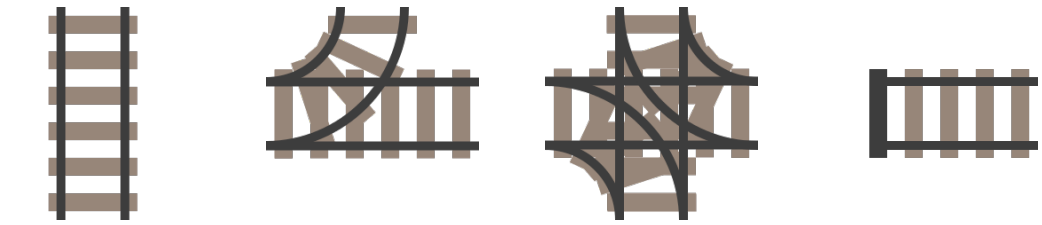
2.3 Railway Navigation

All agents in the Flatland Grid World can choose between different actions, that are defined in the *action space* [Moh+20]:

1. **Do Nothing:** If the agent is moving it continues moving, if it is stopped it stays stopped.
2. **Deviate Left:** If the agent is at a switch with a transition to its left, the agent will chose the left path. Otherwise the action has no effect. If the agent is stopped, this action will start agent movement again if allowed by the transitions.
3. **Go Forward:** This action will start the agent when stopped. This will move the agent forward and chose the go straight direction at switches.
4. **Deviate Right:** Exactly the same as deviate left but for right turns.
5. **Stop:** This action causes the agent to stop.

2.4 Railways

In the Flatland environment there are different railway architectures. The railway block itself defines the action that can be taken or observed by the agent. There are four different railway blocks, as shown in Figure 2.4.1.



(a) Straight Rail (b) Simple Switch (c) Double Switch (d) Dead End

Figure 2.4.1: Different Railway Blocks, reference: [Moh+20]

Straight (a):
This block represents a single passage. While on the block, the agents can *not* take any navigation decisions. They only decide to either continue (passing to the next connected block) or wait. Corners can be considered equally as costly as straight blocks.

Simple Switch (b):
The switch in this block forces trains who arrive from one direction (here West) to make a navigation choice (turn North or East). The cost of navigation in either direction is equally as costly. Agents coming from any other direction do not have a navigation choice.

Double Slip Switch (c):
The double slip switch represents a crossing with two switches accessible from all four directions. Hence, in every case, the agent has a navigation choice. The trains can only change directions through the switches, the crossing alone does not permit any direction changes.

Dead end (d):
Here, only stop or backward motion is possible.

As shown in Figure 2.1.1 there are many combinations of railway blocks. In total nine potential movements are executable, see Figure 2.4.2.

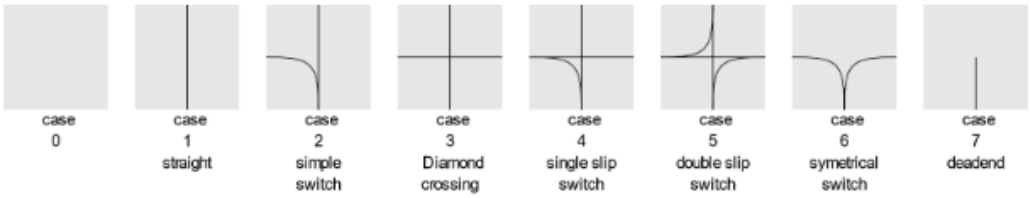


Figure 2.4.2: Cases of Movements, reference: [Moh+20]

Summing up:
Case zero represents the aforementioned environment blocks. Here no movement is possible. Case one is depicts the straight rail. Case two shows the single switch, as mentioned above. Case three however shows a diamond switch. Here two straight railway blocks cross each other. There is no possibility of a navigation choice. Case four combines the single switch with the diamond crossing. Hence, only one navigation choice is possible (here: come from West and turn South). Case five is exactly the same as the double slip switch. Case six can be interpreted as a simple switch. Curved blocks are equally as costly as straight ones. Finally the last Case is the dead end, like before.

2.5 Observations

Flatland offers different observation methodologies for the initialized agents.

Global Grid Observation
Global Observations offer the possibility to observe the whole grid world. Especially considering the use of Convolutional Neural Networks (CNN) seems promising. However flexibility is an issue here, because the smallest changes force a re-training of the network.

Local Grid Observation
Local Grid Observations offer similar advantages as the global grid observations. Here a field of view around each agent can be observed at the same time. The shape of the observation field of view can be chosen arbitrarily. Figure 2.5.1 shows a rectangular field of view specified by height h_i and width w_i.

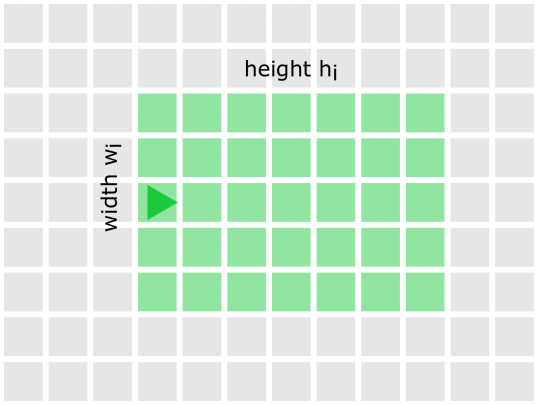
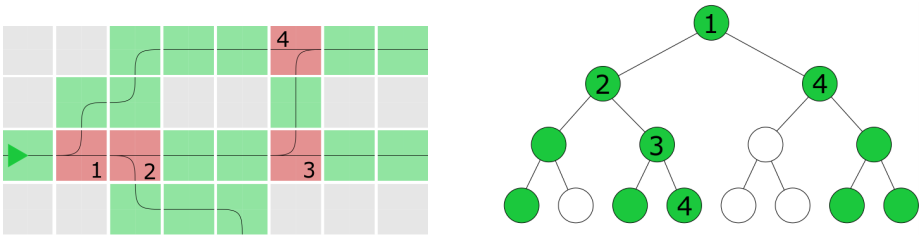


Figure 2.5.1: Local Grid Observation, reference: [Moh+20]

Local Grid Observations show high potential for the use of CNN. However, since only the railway directions are of importance, most of the time a lot of useless information is processed.

Local Tree Observation

Local Tree Observations can be implemented, in order to further enhance the information density of the local observation. This observation methodology is recommended by the producers of Flatland and will hence be used in the considered implementation. This observation methodology provides the possibility to generate observations for all agents in parallel. The basic functionality is displayed in Figure 2.5.2.



(a) Tree Observation (b) Decision Path

Figure 2.5.2: Local Tree Observation, reference: [Moh+20]

The higher level of information density requires a higher computational cost, which marks the major drawback of this approach. Generally a four branched tree is derived starting at the current position of the agent. The trees get unfolded until the individual branch reaches a dead end, switch or the target destination. In case the maximum depth of the tree is not reached, new branches are started. All non existing ones are filled up to *infinity* to guarantee, that the tree size is invariant to the number of possible transitions.

Node Information of Tree Observation

Each node is filled with information gathered along the path to the node. Currently each node contains eleven features [Moh+20]:

1. **Target on explored branch:** if the destination lies on the explored branch, the current distance from the agent in number of cells is stored.
2. **Other agent's target on explored branch:** if another agent's target is detected, the distance in number of cells from the current agent position is stored.
3. **Detection of other agent on explored branch:** if another agent is detected, the distance in number of cells from the current agent position is stored.

4. **Potential conflict on explored branch:** possible conflict detected (Only when a predictor is utilized)
5. **Detection of unusable switch on explored branch:** if an unusable switch is detected the distance is stored. An unusable switch is a switch where the agent does not have any choice of path, but other agents coming from different directions might. (see Section 2.4)
6. **Distance to next node on explored branch:** This feature stores the distance (in number of cells) to the next node (e.g. switch or target or dead-end)
7. **Minimum remaining travel distance on explored branch:** minimum remaining travel distance from this node to the agent's target given the direction of the agent if this path is chosen.
8. **Detection of another agent traveling the same direction:** in this case the number $n > 0$ of agents traveling the same direction is stored, or 0 for no agent.
9. **Detection of another agent traveling the opposite direction:** in this case the number $n > 0$ of agents traveling the opposite direction is stored, or 0 for no agent.
10. **Detection of another agent malfunctioning:** in this case the number $n > 0$ of agents malfunctioning is stored, or 0 for no agent.
11. **Detection of slowest agent in explored branch:** Slowest observed speed of agents in the same direction as the given agent

Normalisation of Tree Output
The tensors produced by the Local Tree Observation Module get pre-processed in order to assure a consistent teaching process of the considered NN. The pre-processing of the input tensors consists of three main steps:

1. Exchanging ∞ by zero
2. Normalize the individual tensors
3. Concatenate the overall input to a single input vector

3 Deep Q Learning

In this chapter the fundamentals of RL are presented. Based on this common Q-learning architectures are depicted.

3.1 Reinforcement Learning

This section is generally based on the book Reinforcement Learning by Sutton et al. [SB18] and on the book Algorithms for Reinforcement Learning by Szepesvári [Sze10].

3.1.1 General RL

Reinforcement Learning
The main goal in Reinforcement Learning is to solve a different kind of task compared to regression or classification, in these kinds of tasks we want the algorithm to be able to take an action. Usually, they are called control tasks. The main difference with other types of tasks, is that in RL there is the added dimension of time, and this means that the actions taken by the algorithm will influence what will happen in successive time steps. Obviously, we do not know what is the best action to take at each time step, we will try to teach the algorithm to accomplish some high-level goal and we can do this by 'reinforcing', hence the name RL, certain behaviours. To reinforce some behaviour usually is used the so-called reward, that in this field can be either a positive or a negative quantity. The objective of the algorithm is simply to maximize its total reward.
The data that will be used to train the algorithm will be produced by the environment; because the data produced may be a lot, we need to have a discrete element that represents the data of the environment, this will be the state. The algorithm is called agent in the RL field. The learning process can be summarized into:

1. the environment is in a state S(i), with $S(i) \in S$, S represents the whole state space
2. the agent takes an action A(i), where $A(i) \in A$, A represents the whole action space
3. that brings the agent to a new state S(i+1) and Reward R(i+1)
4. this process is repeated multiple times

Deep Reinforcement Learning
Using a neural network as an agent is very convenient because of their ability to learn relationships between values. Particularly, in RL we want the neural network to learn the action-value function. This function is used to predict the expected reward, given a state when a certain action is taken; usually symbolized with Q, is also called Q-function (see Section: Value Functions).
Two other important notions are the ones of value functions and policy. A policy is a function that maps a state to the probability distribution over the set of possible actions in that state. Because our aim is to maximize the expected rewards, we can define the optimal policy as the one that, when followed, produces maximum possible rewards. The

value function (or state-value function), symbolized with V, is a function that maps a state S(i) into the expected rewards given that we follow a certain policy.

Single Agent Setting
In a single agent standard reinforcement learning scenario (see Figure 3.1.1), the problem is modeled as an Markov Decision Process (MDP), where at each discrete time step t the agent observes the environment's state S(i) and takes an action A(i) that brings the agent to a new state S(i+1).

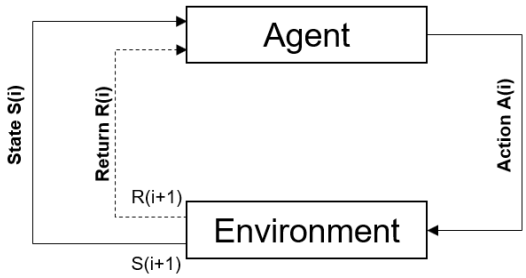


Figure 3.1.1: Single Agent Reinforcement Learning Scenario

Additionally, the agent receives a reward R(i+1) proportional to the quality of the taken action in State S(i). Usually, local rewards are considered, meaning that only the given (S(i), A(i)) pair is affecting the reward. Based on the decisions made a policy π as function $\pi = SxA$, that maximizes the expected cumulative return is learned. Hence, $R = \sum_{i=1}^{\infty} \gamma^i \cdot R(i)$, with $\gamma \in (0, 1)$ as the considered discount rate (a parameter that balances the importance between immediate or future rewards). Choosing a γ close to zero favors the current rewards while a γ close to one favors future rewards.

Multi Agent Setting
Flatland is intended to be a Multi Agent Reinforcement Learning (MARL) Problem. The definition of Actions, States and Rewards changes slightly towards joint Actions, States and Rewards respectively. The transition functions remain analogous to the single agent case.

Q and π - Value Functions
NN can be used to learn the state value function $V_\pi(S)$, that maps the state action pairs (state function). The state value function is a tool to estimate the performance of the agent at a given state. The metric hereby is the accomplished return. For a MDP $V_\pi(S)$ can be defined as:

$$V_\pi(S) = \mathbb{E}_\pi[G(i)|S(i) = S] \tag{3.1.1}$$

considering $G(i) = \sum_{k=0} \gamma^k \cdot R(k + i + 1)$ as the discounted return at time step i, given that the agent follows the policy π.

The value of taking an action A(i) at state S(i), given a policy π can be denoted by $Q_\pi(S, A)$ and can be expressed as:

$$Q_\pi(S, A) = \mathbb{E}_\pi[G(i)|S(i) = S, A(i) = A] \tag{3.1.2}$$

The relationship between the state value function and the Q-function can hence be expressed as:

$$V_\pi(S, A) = \sum_A \pi(S, A) \cdot Q_\pi(S, A) \tag{3.1.3}$$

3.1.2 Optimizing RL

Policy

RL aims to find the optimal policy π^* through maximising the overall return R: $\pi^* = argmax_\pi R$. Thus different apporaches are feasible:

Model Based Approach: The agent learns an understanding o the surrounding world and creates a model to internally represent it. Hence, predictions can be made before the next step (next action) is taken.

Model Free Approach: The agent relies on real samples of the surrounding world. It never generates predictions about upcoming actions. Generally there are two model free approaches that are generally utilized: *Value Based* (each step is taken according to the most promising action) and *Policy Based* (each step is taken according to the overall best policy) approach.

Rewards

Rewards for the purpose of RL need to be designed in a straight forward fashion, since they contribute dramatically to the learning result of the chosen value function. The Flatland implementation already provides rewards:

-1: each step taken by the agent, additionally if the agent does invalid moves.

+1: in case the agent arrives at the desired destination.

-10: (*Additional (own) Reward:*) In order to address problems like deadlocks (two or more agents block each other in the grid world, see Figure 3.1.2), a *lock_penalty* is implemented. A global check is run that assures whether agents are hindered to move. Here a differentiation between malfunction and actual deadlock is considered. Deadlocks then get punished.

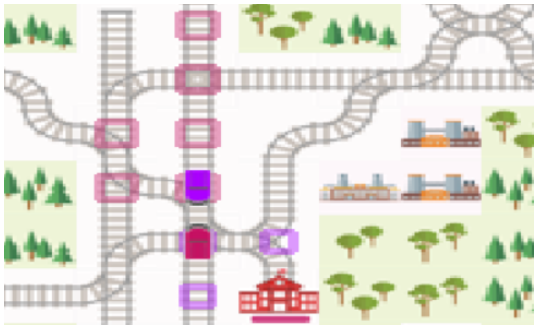


Figure 3.1.2: Deadlock between two Agents

Actions

In RL problems, roughly speaking, the agent searches for the best of the feasible policies. Hence, the agent would be constantly searching the variety of policies in order to improve the current performance, ultimately archiving the global optimum of performance (thus the best possible policy). In the domain of training NN, this can be achieved using Gradient Decent Methods like Adam [KB15] or Stochastic Gradient Decent (SGD) [LGN20].

Because of this local search perspective, another key aspect of the RL algorithm is to balance the exploration-exploitation tradeoff. Exploration defines exploring a large portion of the search space with the objective of finding additional promising solutions that are yet to be refined. Exploitation however is related to the refinement of an already promising solution, by intensifying the search in its neighborhood. Practically speaking,

the agent at each time decides weather to follow the optimal policy *(exploitation)* or to chose a different action *(exploration)*.
Over time and hence training steps of the agent, the exploration parameter ϵ shall decrease in order to emphasise maximum rewards. There are different methods of decaying the exploration parameter ϵ:
ϵ - **Greedy:** The ϵ-greedy action selector is already implemented in the Flatland Rail Environment Generator. The exploration parameter $\epsilon \in [0, 1]$ is a random number (derived from a normal distribution) that defines what action is going to be taken.

Greedy: set $\epsilon = 0$ and avoid decaying it (always select the best action)

Random: set $\epsilon = 1$ and avoid decaying it (always select a random action)

By slowly decreasing the value of ϵ during training, the agent initially needs to explore the grid world taking random decisions to then more and more follow the learned policy.

3.1.3 Deep Q-Learning Network Architectures

Deep Q-Network
The Q-learning is a method of earning optimal action values, which are represented by the Q function. The main idea is that our neural network will predict the expected rewards given a state-action pair, which is then compared to the Bellman equation (see Equation 3.1.2). Hence, for Q-Learning a Deep Q-Network (DQN), a simple feed forward network that uses the aforementioned loss function to learn, is implemented.

Dueling Deep Q-Network
In 2016 researchers from Google DeepMind [VGS16], proposed a variation on the architecture of the DQN. The main idea was to split the stream of data into two separate ones (dueling):

one for the **state-value** function

and one for the **action advantages**.

These two streams are then combined with an aggregation layer, in order to produce an estimate of the state-action value function Q. The benefit of those separate streams is that the dueling architecture can learn which states are (or are not) valuable and learn how to deal with them instead of learning which action is the best in every possible state. This is particularly useful when there are a lot of states where an action does not bring any effect on the environment. Figure 3.1.3 illustrates a potential Dueling-DQN Architecture.

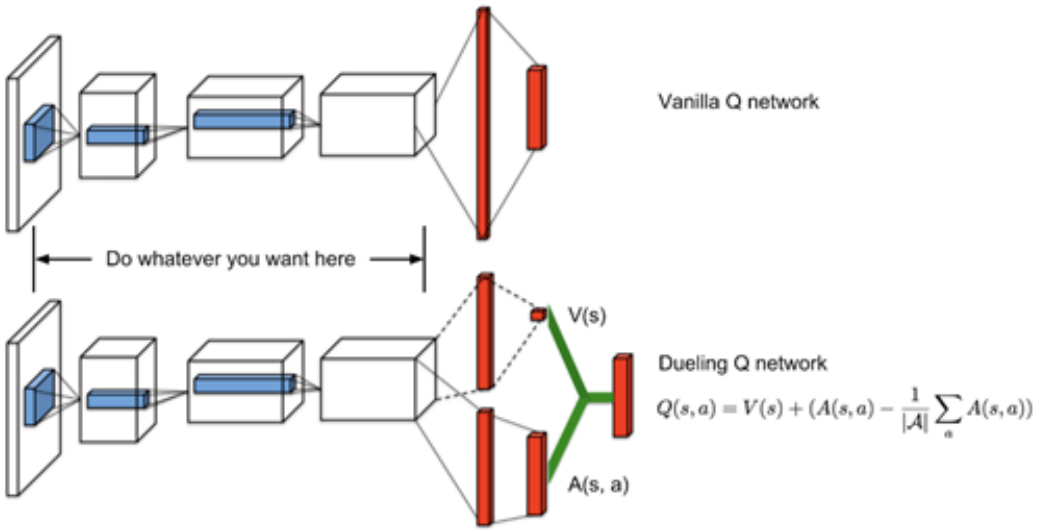


Figure 3.1.3: Dueling Deep Q-Network, reference: [VGS16]

3.1.4 Training Techniques

There are various techniques that can improve learning stability and speed, which are described in the following.

Experience Replay
Experience replay is mainly used to solve the problem of catastrophic forgetting. This happens when the gradient bounces around a certain value, because of similar state-actions during the online training. To tackle this, we want to emulate what happens in batch learning:

the state S(i)

the action A(i) taken in S(i)

the obtained reward of the next state

and the next state S(i+1) are stored in a buffer (as a tuple).

We can then sample randomly from this buffer and compute the gradient on a batch of experiences. This is useful also because generating those tuples can be computationally expensive and so, by storing them, we can use them multiple times to train the network.

Target Network
Instead of trying to train the network at every time step, we may want to adopt a certain behaviour for a period of time and then train the network, this means that we don't want to change the parameters at each timestep. We can implement this simply by using a second neural network that mimics the *main* one; we will train the first network and use the second one to compute the *max Q* that we've seen in the Bellman equation. After a certain amount of steps, the parameters of the first network will be copied into the second one. This has been shown to lead to a better stability of the learning process.

4 Implementation and Results

4.1 Proposed Architecture

We decided to implement the Dueling-DQN architecture because it's the one that has given the best results in many RL settings. Our model will take a vector of size **231** as input, because we're considering the **Tree Observation** from Flatland with a max depth of **2**. The output will be a vector containing **5** values, specifically **five Q-values**, because the action space is composed by **5 possible actions**. To improve the model's performance, we implemented also an *experience replay buffer*, a *target network* and we also used *gradient clipping*, a technique that was used also in the DeepMind paper. Additionally improvements in the rewarding are considered, specifically the aforementioned deadlock punishment.

4.2 Results

The first implementation of the agent was done in Tensorflow. We spent almost 3-4 weeks trying to understand why the network was not learning but we were not able to solve the problem. Hence, we decided to use Pytorch and, without any change in the logic, we obtained some good results. Unfortunately, this time loss prevented us from spending more time on improvements for the multi agent setting.

4.2.1 Single Agent Setting

The first tests were done, in order to understand whether the NN was working properly and if it was able to learn. The environment setting of Flatland was the following:

1. width = 35
2. height = 35
3. n_cities = 3
4. max_rails_between_cities = 2
5. max_rails_in_city = 3

For the single agent scenario, every training session has been done on 1k episodes, of 300 steps each, with 100 training loops after every episode with a batch size of 512.

Learning Rate
The first comparison was done on different learning rates to choose one. In the following plot (see Figure 4.2.1) we can see the result of the comparison of three different learning rates.

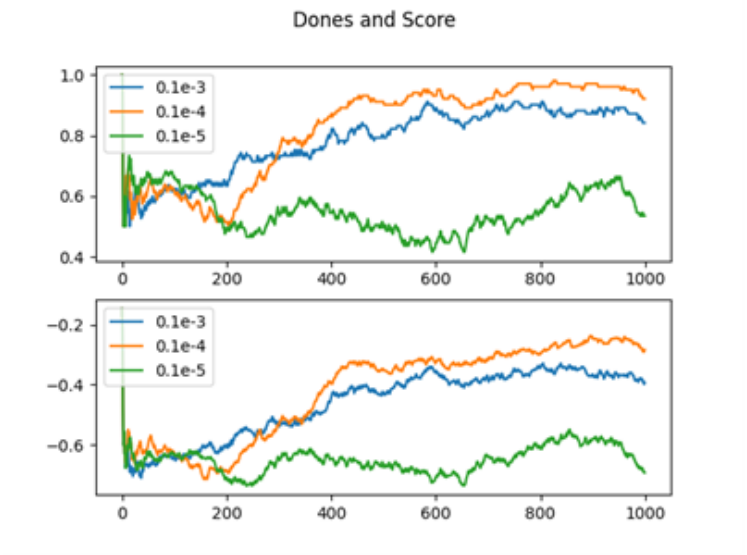


Figure 4.2.1: Learning Rate Single Agent Setting

The best results are obtained using the the second learning rate ($1,0 \cdot 10^{-4}$). The third one (green) is too small to give to the network the possibility to learn properly. Hence, a learning rate of $1,0 \cdot 10^{-4}$ is used, which shows an associated score of 92% (92% of trains arrive at destination).

Modified Rewards

To improve the learning speed and capability, it is crucial to tweet the default rewards of the Flatland library. We considered the two most important ones: the step penalty, which is a negative reward given at each time step to every agent, and the global reward, that is a positive reward given when an agent is 'done' (i.e. reaches the target). The default ones are listed in Section 3.1.2 It's clear that those values are not proportional, our first thought was to either lower the step penalty value or increase the global reward. To have a clearer view of how these changes influenced the learning phase, we have tried different combinations of these two values. In total we tried 3 different combinations, that are:

1. step $= 0.2$ and global $= 10$;
2. step $= 0.01$ and global $= 1$;
3. step $= 1$ and global $= 100$;

Apparently, the permutation of values that leads to a faster learning is the one that penalize the step penalty and maintains the default global reward. We thought that the third combination could lead to a similar results but it seem to be a little bit slower even compared to the first combination of rewards. Because of this outcome, we decided to use the second combination as *the best one* (score $= 93\%$) for the following training cycles (see Figure 4.2.2).

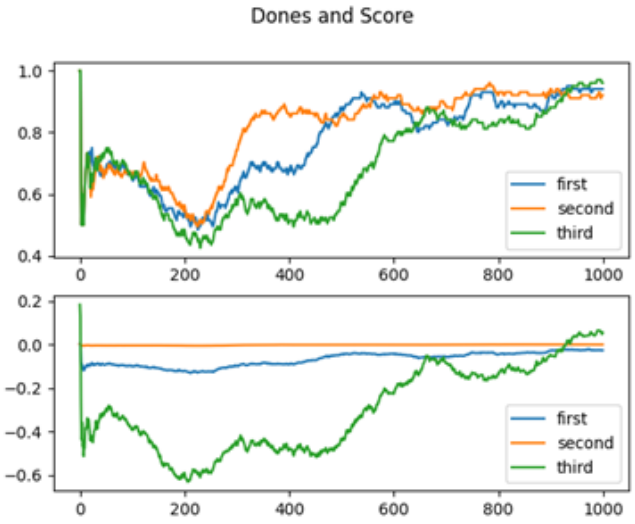


Figure 4.2.2: Rewards Single Agent Setting

Gradient Clipping

This practice is more used in Recurrent Neural Networks, to avoid the so-called gradient explosion problem, than it is in Reinforcement Learning but in the paper of DeepMind [VGS16], they showed that clipping gradients can lead to performance improvement. We first computed the mean of the norm of the gradient in 1000 episodes, which resulted to be around 5, so we decided to clip the norm to that value and then we tried also to double that. The results can be seen in Figure 4.2.3.

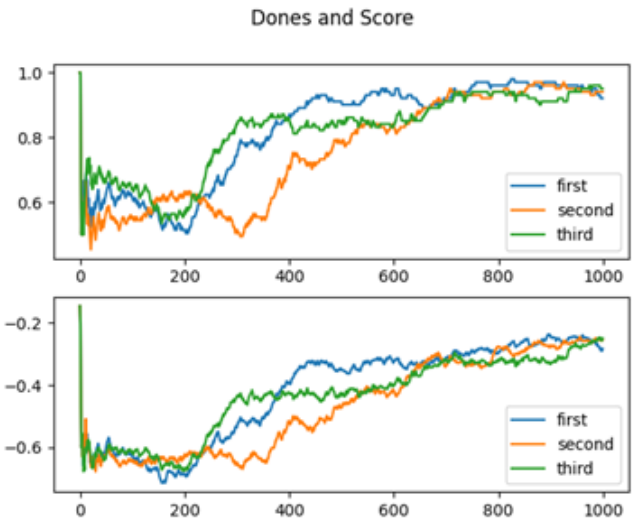


Figure 4.2.3: Gradient Clipping Single Agent Setting

The results are not that noticeable in later episodes, but in the first episodes clipping the gradient norm to 10 seems to increase slightly the learning speed. Also in this case, clipping the norm to 10 has be considered as the best option and used in later training cycles. Ultimately a score of 95% can be achieved.

4.2.2 Three Agent Setting

For the training of the multi agent setting we used all the different choices that seems to improve the performance of the training phase in the single agent setting.

For the training of three agents, we modified the Flatland environment a little bit in order to have sufficient *space* to perform the tests. The specification of the environment are:

1. width = 40
2. height = 40
3. n_cities = 4
4. max_rails_between_cities = 2
5. max_rails_in_city = 3

We also decided to increase the number of steps to 350, the number of episodes to 2k and the number of training iteration for episode to 200, see Figure 4.2.4.

Figure 4.2.4: Default Three Agent Setting

As expected, the percentage of dones is lower if compared to the single agent scenario. This is mostly due to the fact that in the multi agent scenario there is the problem of deadlock, which is the major causes of the lower amount of dones. Still, the peak of the percentage was around 79% and the mean, in later episodes, was around 74% which is a decent result considering, that there are no changes from the single agent scenario.

4.2.3 Five Agent Setting

Also in this case we modified the settings of the Flatland environment, that are:

1. width = 50
2. height = 50
3. n_cities = 5
4. max_rails_between_cities = 2
5. max_rails_in_city = 3

Default Performance

We increased the number of steps to 400 per episode. In the results we can observe a behaviour that has appeared also in other training sessions: the percentage of dones begins to drop after a certain amount of episodes, even if the network first showed some improvements. The other times we fixed this simply by increasing the number of training iteration per episode but, since it takes a lot of time to re-train the network, we decided to use the obtained results, see Figure 4.2.5.

Figure 4.2.5: Default Five Agent Setting

It's clear that, by increasing the numbers of agents, the percentage of dones struggle to increase because of the deadlocks (see Figure 3.1.2).
We think that the work done is a good starting point for future improvements in the observations and in the predictors, that are the customizable sections of the Flatland library that can surely lead to performance improvements. To increase the current score of 56%, the agent needs to have more information on the path of other agents and should be better aware of the distance and the best path to choose in order to reach the target.

Modified Rewards

Figure 4.2.5 shows, that the training results after 1000 episodes decrease steadily. that is why for this subsection only 1000 episodes are considered. Deadlocks show to be the greatest problem in this vehicle rescheduling problem. As already mentioned in Section 3.1.2, implementing an additional reward for deadlocks might be useful. Hence, attempts to tackle this problem are presented in the following.
First of all the default rewarding system is researched, to better understand if simple stop and start punishments can already hinder agents from getting stuck. Figure 4.2.6 shows the best results obtained. Strangely, a negative global reward seems to be beneficial for the overall Score (here score = 60%).

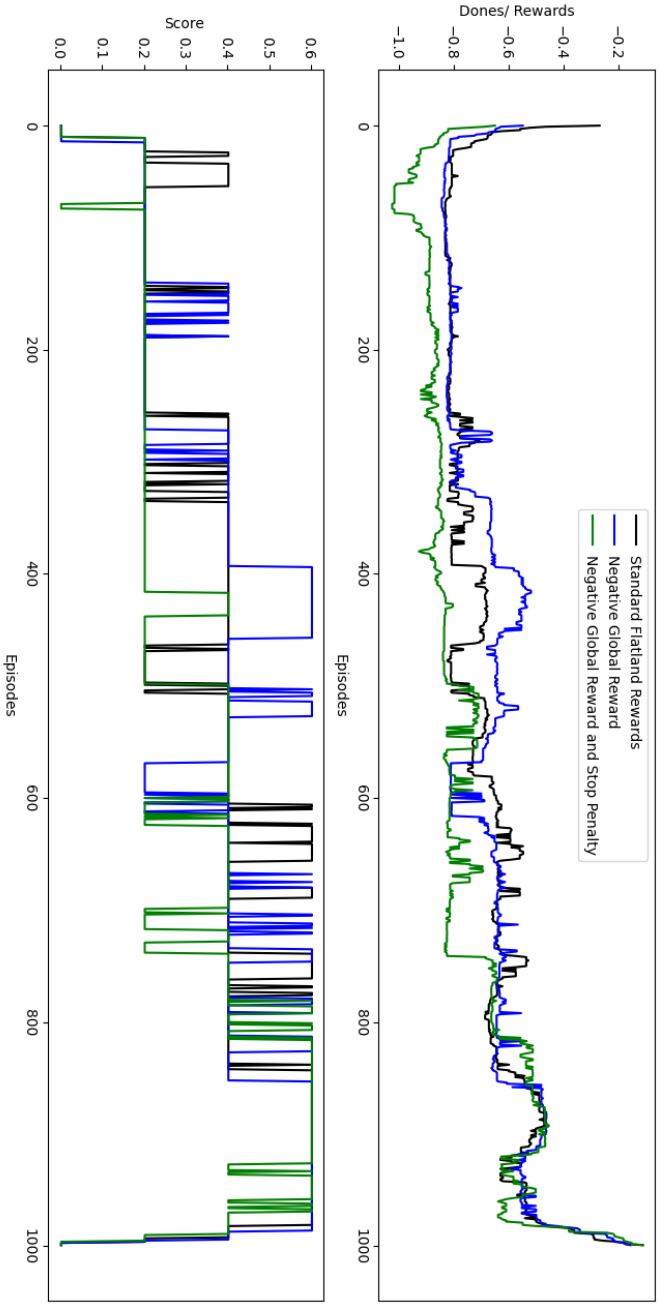


Figure 4.2.6: Reward Five Agent Setting

Based on the aforementioned results, two different approaches to punish deadlocks are considered. A deadlock punishment if the agent is already stuck and wants to initialize a future action (blue) or a punishment if the agent is moving and the cell it wants to move to is not free (because of another agent) (green), see Figure 4.2.7. In the second case it needs to be clarified if the agent blocking the cell is malfunctioning or not, because in some cases the potential deadlock might resolve itself after time.

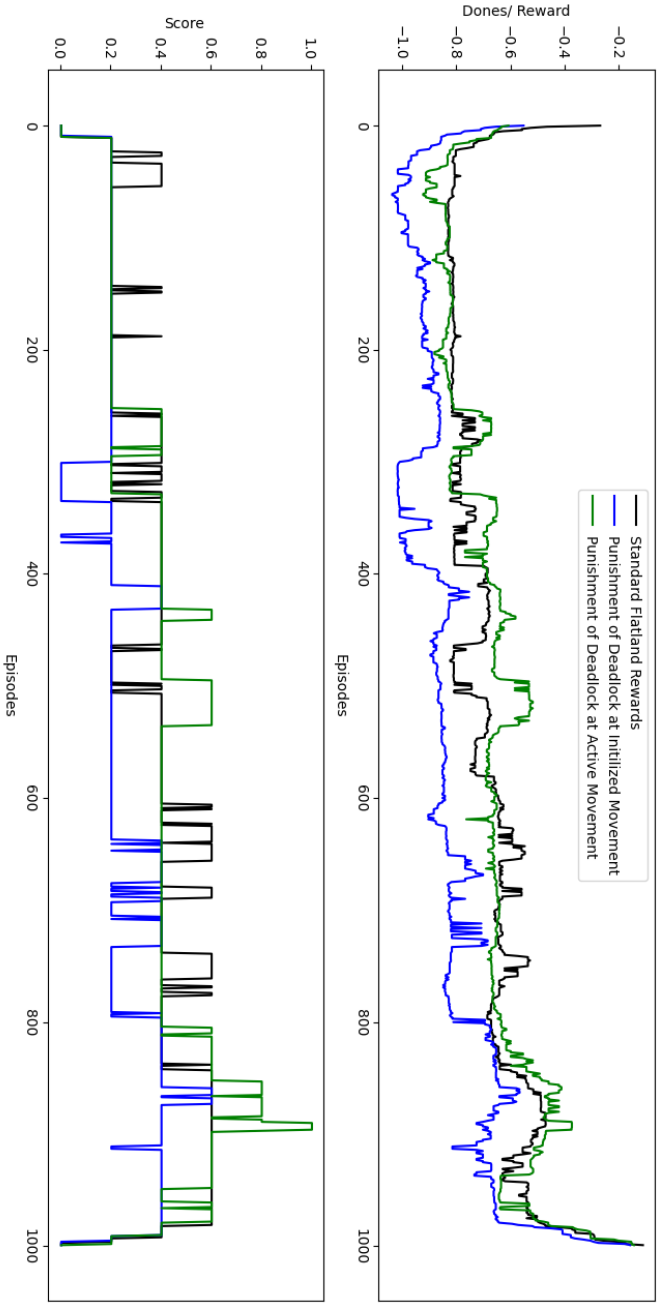


Figure 4.2.7: Deadlock Reward Five Agent Setting

The best results in this scenario are obtained utilizing the moving agent strategy. The overall score can be improved by $\approx$ 10% from 56% (default score after 1000 episodes with 5 agents) to 65% overall score after 1000 episodes with 5 agents.

5 Conclusion

The FC is a competition organized by AIcrowd [Moh+20] in corporation with SBB to address the problem of train scheduling and rescheduling by providing a simple grid world environment and allowing for diverse experimental approaches. The SBB operates the densest mixed railway traffic in the world and maintains the biggest railway infrastructure in Switzerland. The goal of the FC is to make all the trains considered in the grid world arrive at their target destination with minimal travel time. Hence, the the number of steps that it takes for each agent to reach its destination shall be minimized.
In this report the use of Dueling-DQN is considered, in the framing of a RL based approach to tackle the stated vehicle rescheduling problem. Dueling Deep Q-Networks split the information stream within the network into the state-value function and the action advantages, that ultimately get combined in the Q-function. The considered NN has five outputs for the five potential actions that the agent can ultimately take.
There are various settings to be executed: Single, Three or Five Agent Scenario. The Single Agent Scenario almost immediately shows good results. Final adjustments then lead to an overall performance (score) of 95%. In the Multi Agent Scenario however, achieving high scores is not as easy. In the three agent setting a default overall score of $\approx 75\%$ can be achieved, which is still satisfying in comparison ti the five agent scenario (default score 56%). The major contribution to the lowered results is identified to be deadlocks arising from malfunctions or bad decisions of the agent. Hence, modified rewards for the five agent scenario are considered. Implementing a deadlock punishment for moving agents helps to increase the overall score up to $\approx 65\%$.
To further increase the performance of the considered multi agent scenario various actions can be considered. Here specifically the deadlock problem has to be taken into account, since malfunctions are always a potential thread to the overall score.
Agents need more information about the path of other agents and should be made aware of the distance and the best path to choose in order to reach the target. Hence, a modified observation module in combination with a modified reward module seems to be a promising way forward.

References

Books

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. 2nd. The MIT Press, 2018. ISBN: 9780262039246.

[Sze10] C. Szepesvári. *Algorithms for Reinforcement Learning*. 1st. University of Alberta, 2010. DOI: https://doi.org/10.2200/S00268ED1V01Y201005AIM009.

Paper, Articles, Online Sources

[KB15] D. P. Kingma and J. L. Ba. *ADAM: A Method for Stochastic Optimization*. Tech. rep. 2015. URL: https://arxiv.org/pdf/1412.6980.pdf (visited on 12/03/2020).

[LGN20] S. Lau, J. Gonzales, and D. Nolan. *Principals and Techniques of Data Science*. Tech. rep. 2020. URL: https://www.textbook.ds100.org/ch/11/gradient_stochastic.html (visited on 12/04/2020).

[Moh+20] Sharada Mohanty et al. "Flatland-RL : Multi-Agent Reinforcement Learning on Trains". In: (2020). arXiv: 2012.05893 [cs.AI].

[VGS16] Hado Van Hasselt, Arthur Guez, and David Silver. *Deep reinforcement learning with double q-learning*. Tech. rep. 1. 2016.