



FLATLAND CHALLENGE

Deep Learning Course - Final Project

Matteo Conti, Manuel Mariani, Davide Sangiorgi

September 2020/21

M.Sc. Artificial Intelligence, University of Bologna

Table of contents

1. Flatland Improvements
2. Reinforcement Learning
3. Neural Networks
4. Implementation
5. Configuration Of The Experiments
6. Analysis and Results
7. Conclusions

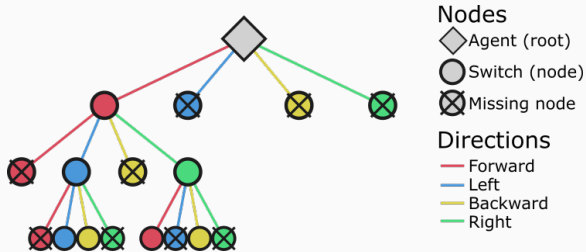
Flatland Improvements

The starting point for our observator is the tree observator from `flatland.envs.observations.TreeObsForRailEnv`.

We adapted it to the problem changing those main properties:

- the **node** structure
- the **search** strategy
- the number of **explored branches**

Explored Branches



Default Flatland Tree Observer

We preferred the **breath first** strategy with respect to the depth first strategy previously adopted for branch exploration.

Flatland switches often create railways which immediately merge again with the previous one. Those are really useful to avoid collision and traffic, but on the other hand we will have nodes explored twice during observation.

Considering that in **straightaways**:

- It's not optimal for a train to stop
- Actions *LEFT* and *RIGHT* do not matter

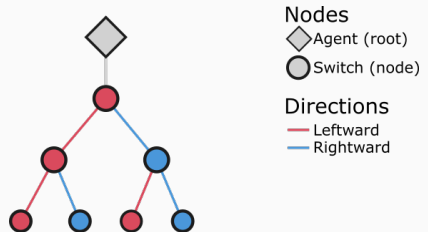
We have improved the environment by *requiring an action* only when a train is:

- At an intersection, allowing the choice of **direction**
- Approaching an intersection, allowing the choice of **stopping**

Explored Branches

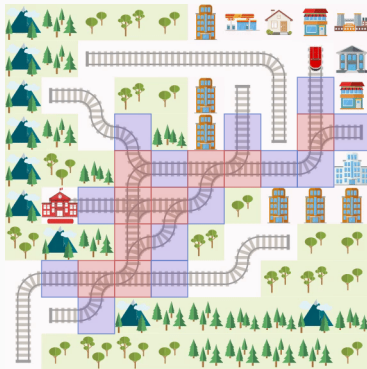
Flatland switches allow **only two** possible directions and, as already set in the default tree observer, we save only nodes corresponding to switches.

The result of the observation is then a **binary tree**: only the leftward and the rightward child are created.



Actions

We defined an **Environment Wrapper** that requires trains to provide actions only in proximity of rail switches:



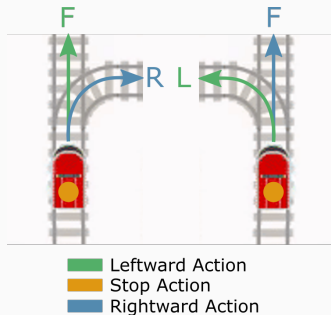
Railway sections where the train action is required.

In red: switches

In blue: tiles before or after a switch

Actions

Considering the flatland environment, we have also optimized the set of possible actions, limiting them into: *STOP*, *MOVE LEFTWARD*, *MOVE RIGHTWARD*:



Example of **custom actions**, remapped to the Flatland's actions (*Left*, *Forward*, *Right*, *Stop*)

The flatland default reward is not enough to determine if a path could be a good one or not.

Therefore we introduced a positive and a negative reward. We call them **attractive force** and **repulsive force** because they refer to gravitational force. The first represents the attraction towards the target, while the second represents the repulsion from agents moving in opposite direction.

$$force_{attractive} = \frac{MASS_{TARGET}}{dist^2}$$
$$force_{repulsive} = \sum \frac{MASS_{AGENT} * N_{OPPOSITEAGENTS}}{dist^2}$$

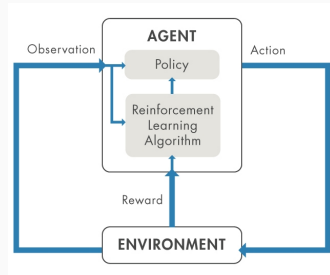
Reinforcement Learning

Reinforcement Learning

Reinforcement Learning is a technique in which **agents** interacting with an **environment**, learn the best **actions** given the **observations** from the environment, by *maximizing* a **reward function**.

Execution of a RL step:

1. Agent **observes** the environment
2. Agents chooses an **action**
3. Environment executes the action, advancing its **state**
4. A **reward** is provided to the agent, which is then used by the **RL Algorithm** to learn the best action



Q-Learning

The most simple RL algorithm is **Q-Learning**. It uses:

- A **Q-Table** that maps *state-action* pairs into numeric "quality" values

$$Q : S \times A \rightarrow \mathbb{R}$$

- **Temporal Difference Learning**, that accounts for the delayed future rewards as an effect of an action

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot [r_t + \gamma \cdot \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t)]$$

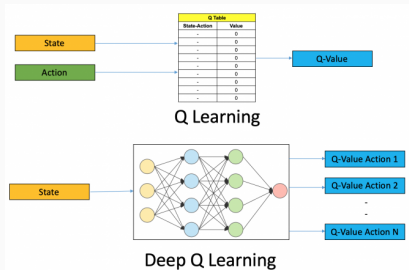
where

- s_t, a_t, r_t are the state, action and reward in the timestep t
- α is the *learning rate*
- γ is the *discount factor*

Deep-Q-Learning

Since traditional Q-Learning uses a table, it quickly becomes unfeasible as the observation space and action space increases. To solve this, **neural networks** are used to *approximate* the Q-Table.

The networks, instead of mapping state-action pairs into a single Q-Value, map states to **multiple Q-Values**, one for each action.



To train the network, the **loss** function is defined as the *distance from the expected Q-Value and the prediction*:

$$L(\theta) = (\mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s', a')] - Q(s, a, \theta))^2$$

The loss is computed using an **experience memory** that contains (s_i, a_i, r_i, s_{i+1}) tuples.

In this project the experiences stored inside the memory are sampled **randomly** with an uniform distribution.

Double Deep-Q-Learning

A variant of DQN, called **Double Deep Q-Learning** that reduces **bias overestimation** by:

- Using **two neural networks** with parameters θ and θ' , alternating their roles
- A loss function using the second network to estimate the expected Q-Value

$$L(\theta) = (\mathbb{E}_{s'}[r + \gamma Q(s', \bar{a}|\theta')] - Q(s', a'|\theta))^2$$

$$\bar{a} = \arg \max_{a'} Q(s', a'|\theta)$$

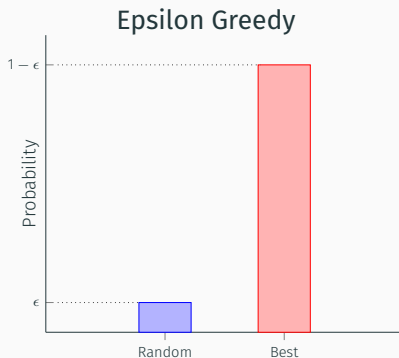
- Periodically updating the first network with the parameters of the second, either with a full overwrite (hard update) or a weighted mean (soft update)

Policies are functions $\pi(a|s)$ that determine which actions to choose in a given state.

- **In testing** the action associated to the best Q-Value is always chosen
- **In training** the actions are chosen with a random probability, to explore novel strategies

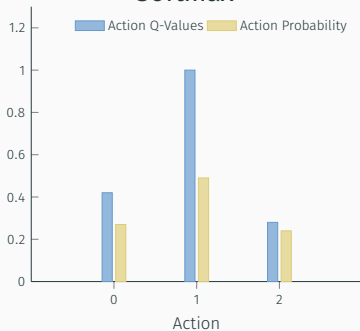
We used three policies to test their performance in training speed:

- **Epsilon Greedy Policy**: selects a random with probability ϵ , otherwise the best action currently learnt
- **Softmax**: the Q-Values are normalized using the softmax-function and used as probabilities to select the action
- **Boltzmann**: a variation of softmax, which uses an hyper-parameter called *temperature* τ to modify the distribution.
 - Higher $\tau \rightarrow$ more deviation (more spread)
 - Lower $\tau \rightarrow$ less deviation (higher peaks)



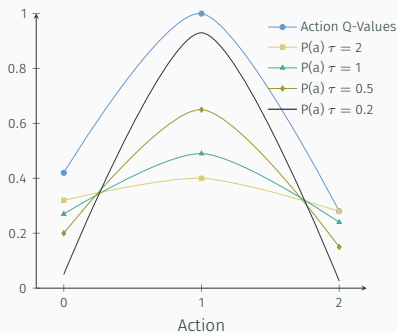
$$a = \begin{cases} \text{random action} & \text{if } r < \epsilon \\ \arg \max_a Q(s, a) & \text{if } r \geq \epsilon \end{cases}$$

Softmax



$$P(a_i) = \frac{e^{Q(s, a_i)}}{\sum_a e^{Q(s, a)}}$$

Boltzmann



$$P(a_i) = \frac{e^{Q(s, a_i)/\tau}}{\sum_a e^{Q(s, a)/\tau}}$$

Neural Networks

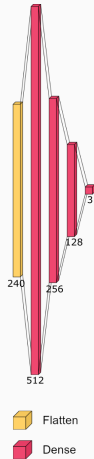
Sequential

In the project, all neural networks use a **Sequential** (Feed Forward) architecture.

Three networks were designed, each with its own characteristics, to compare their performance:

Network	Parameters	Main Feature
seq1	288,003	Dense layers
seq2	128,387	Dense layers
conv1	7,551	Conv1D preprocessor + Dense layers

Dense

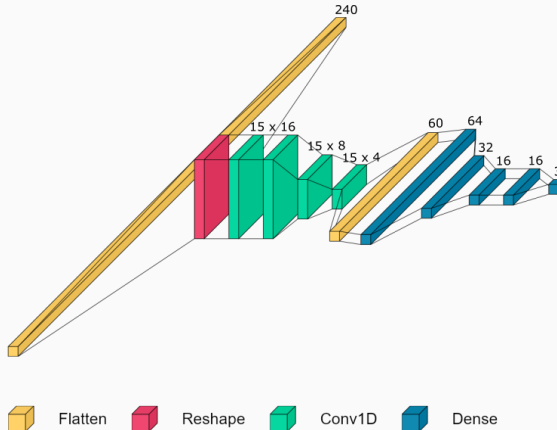


The *seq1* network structure



The *seq2* network structure

Convolutional 1D



The *conv1* network structure

Convolutional 1D

The main characteristics of the **conv1** network are:

- The input is reshaped into a 15x16 matrix
- A series of 1D convolutional layers is used as a **Preprocessor** to reduce the dimension of the observation
- The reduced observations, representing the "quality" of the network, are then used by a sequence of Dense layers to determine the final Q-Values

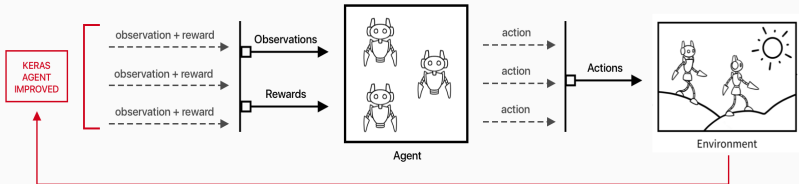
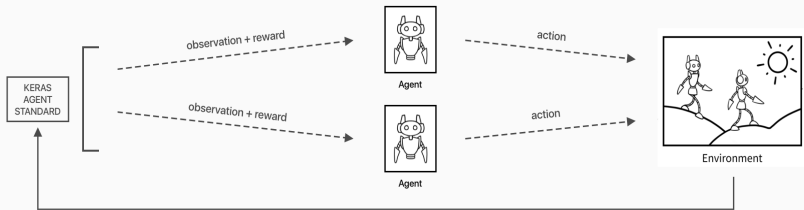
The advantage of this approach is a greatly reduced number of parameters, since the 1D convolution is parametrized only along one of its dimensions

Implementation

Frameworks

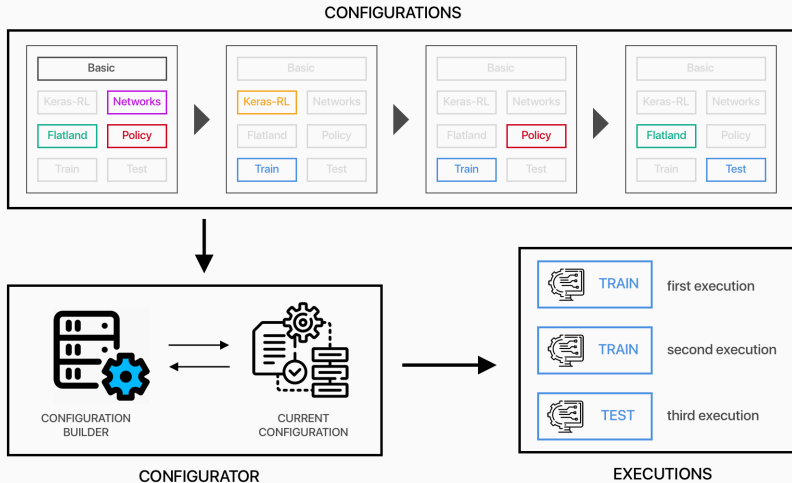
	Keras-RL	TF-Agents	Tensorforce
custom-env	✓	✓	✓
custom-actions	✓	✓	✓
custom-memory	✓	-	-
framework-based	-	✓	✓
policies	✓	✓	-
callbacks	✓	-	-
DQN	✓	✓	✓
DDQN	✓	✓	✓
DDQN (dueling)	✓	-	✓
multi-actions	✓	-	-
multi-agents	-	-	-

Keras-RL with Multi-Agent support



Configuration Of The Experiments

Baseline and Builder



The static configuration of the Flatland environment used inside all the training/testing experiments is:

```
1 {  
2     "map_width": 28,  
3     "map_height": 21,  
4     "n_cities": 2,  
5     "max_rails_in_city": 4,  
6     "max_rails_between_cities": 4,  
7     "cities_grid_distribution": false,  
8     "malfunction_rate": 0.0001,  
9     "malfunction_min_duration": 1,  
10    "malfunction_max_duration": 5  
11 }
```


Analysis and Results

Evaluation of the three network **topologies**: **seq1**, **seq2** and **conv1**, on 100 episodes of maximum 500 steps each.



Number of steps to reach the target



Mean action value

Policies

Training performance for the **policies**: **Epsilon-Greedy** and **Boltzmann**, using the *seq2* network, carried out over 500,000 training steps

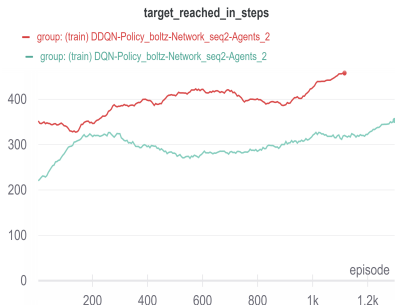


Number of steps to reach the target

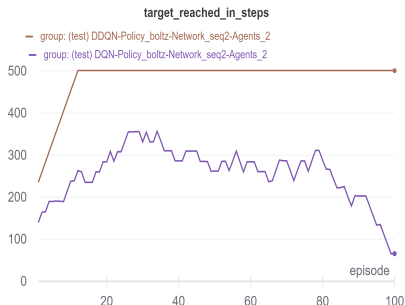
RL Algorithms

Training and testing performance of **DQN** and **DDQN**, using the *seq2* network and *boltzmann* policy.

Training was carried out over 500,000 training steps and then tested on 100 episodes with 500 steps maximum each.



Training - Number of steps to reach the target

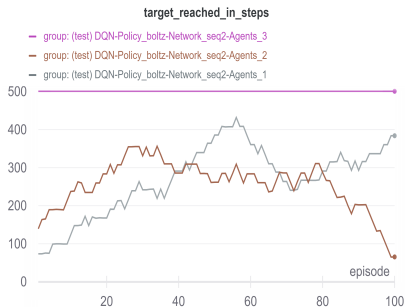


Testing - Number of steps to reach the target

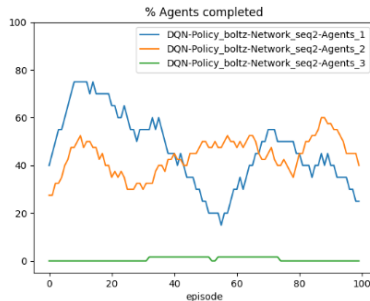
Agents

Finally, to evaluate the performance in the multi-agent environment, a **DQN** with **seq2** network and **boltzmann** policy was tested in environment with **one**, **two** and **three** agents.

Testing was carried out over 100 episodes, of 500 steps each.



Number of steps (for each agent) to reach their target



Percentage of agents reaching their target

Conclusions

Improvements

Considering the power and ease of use of our custom JSON configurator, a number of **improvements** could be made using this project as a valid starting point:

- **Hyperparameter tuning** (learning rate, optimizer, policy parameters)
- **Policy annealing**
- Tuning the parameters and topology of the **1D Convolutional network**
- Using **prioritized experience** replay
- Different **environment** setup

Final Remarks

This project has been a wonderful opportunity to:

- **Learn** Reinforcement Learning and Deep Learning concepts in a first hands experience
- **Understand** that Deep Learning, particularly Reinforcement Learning, is a trial and error process that requires patience and perseverance
- **Discover** new libraries and resources
- **Organize** a team, making the best use of each of the member's strengths
- **Think outside the box**, to design and develop our own ideas to solve complex problems
- And many more!!

Thank you for your time and thanks for the opportunity!

- Davide, Manuel, Matteo