Multi-Agent Reinforcement Learning on Trains

Team:

Ivașcu Vlad-Alexandru

Balan Marius-Alexandru

Peltea Mihai-Bogdan

Tcaciuc Ioan-Gabriel

This is a real-world problem faced by many transportation and logistics companies around the world such as the Swiss Federal Railways and Deutsche Bahn.

Our goal is to make all the trains arrive at their target destination with minimal travel time. In other words, we want to minimize the number of steps that it takes for each agent to reach its destination.

Flatland Environment

The trains in Flatland have strongly limited movements, as you would expect from a railway simulation. This means that only a few actions are valid in most cases.

Here are the possible actions:

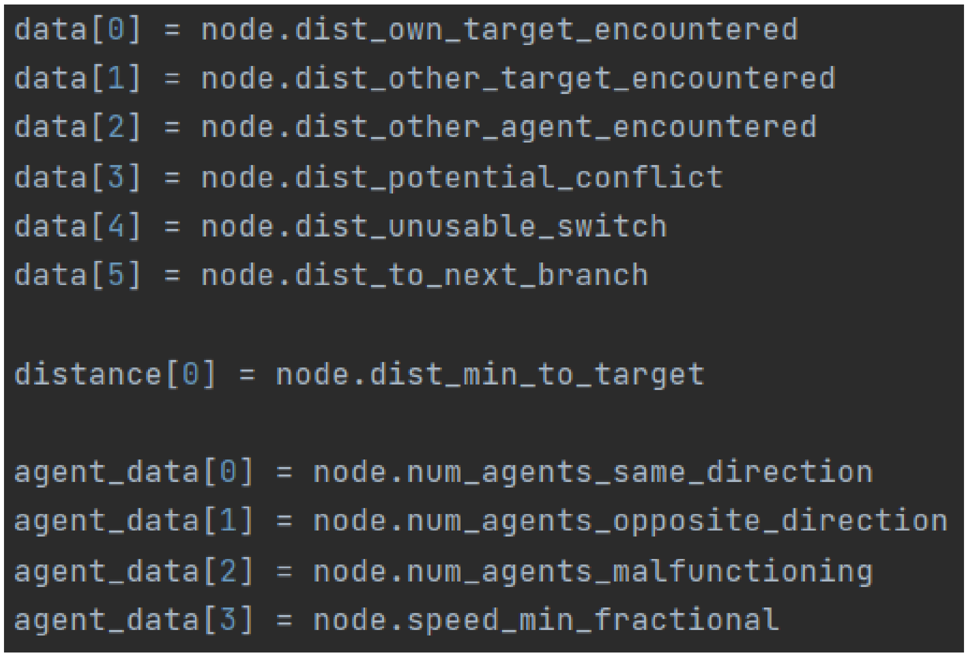
- **DO_NOTHING**: If the agent is already moving, it continues moving. If it is stopped, it stays stopped
- **MOVE_LEFT**: Change direction towards the left, where possible.
- **MOVE_FORWARD**: The agent will move forward.
- **MOVE_RIGHT**: Change direction towards the right, where possible.
- **STOP_MOVING**: Causes the agent to stop.

At each time step, each agent receives a combined reward which consists of a local and a global reward signal.

Locally, the agent receives the local reward = −1 for each time step, and local reward = 0 for each time step after it has reached its target location. The global reward signal only returns a non-zero value when all agents have reached their targets, in which case it is equal with 1.

For the agents' training we used the Tree Observation class, found in the flatland environment. The tree observation exploits the fact that a railway network is a graph and thus the observation is only built along allowed transitions.

Given an observation radius, we move along the path and for every switch, dead-end or when the target is reached, a new node is created. A node has 4 children, for every possible direction the agent can go towards and it contains the information it has gathered along the way. The information is stored using 12 features, of which we use the following 11:

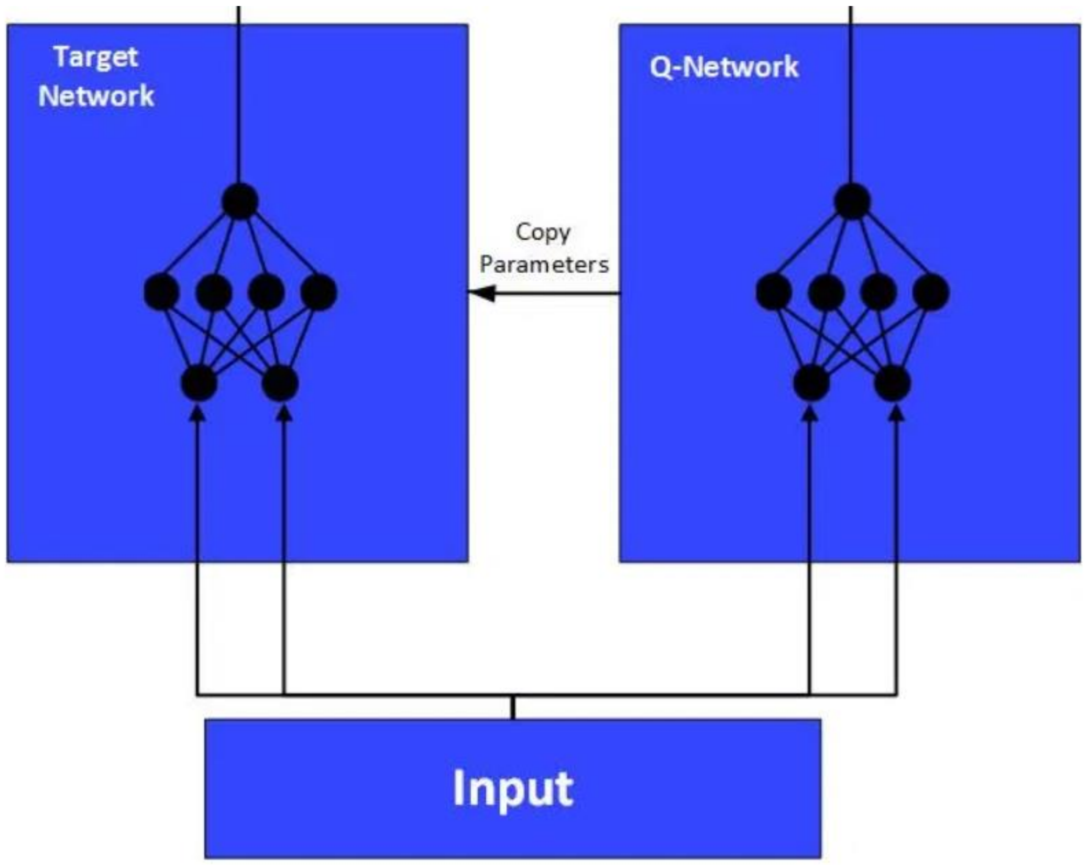


```
data[0] = node.dist_own_target_encountered
data[1] = node.dist_other_target_encountered
data[2] = node.dist_other_agent_encountered
data[3] = node.dist_potential_conflict
data[4] = node.dist_unusable_switch
data[5] = node.dist_to_next_branch

distance[0] = node.dist_min_to_target

agent_data[0] = node.num_agents_same_direction
agent_data[1] = node.num_agents_opposite_direction
agent_data[2] = node.num_agents_malfunctioning
agent_data[3] = node.speed_min_fractional
```

We store this data in 3 buffers, as seen above and we normalize it by bringing it in the interval [-1, 1]. This process is repeated for each agent and the result represents their specific observation.

These observations are then given as input to the neuronal networks. The output of the target network will be an action for the current agent and the output of the q-network will be the predicted reward for the next state. Based on how well the networks are trained, the action may be detrimental for future steps, so we use epsilon greedy algorithm to balance the exploration with exploitation.

Neural networks architecture:



Target network:

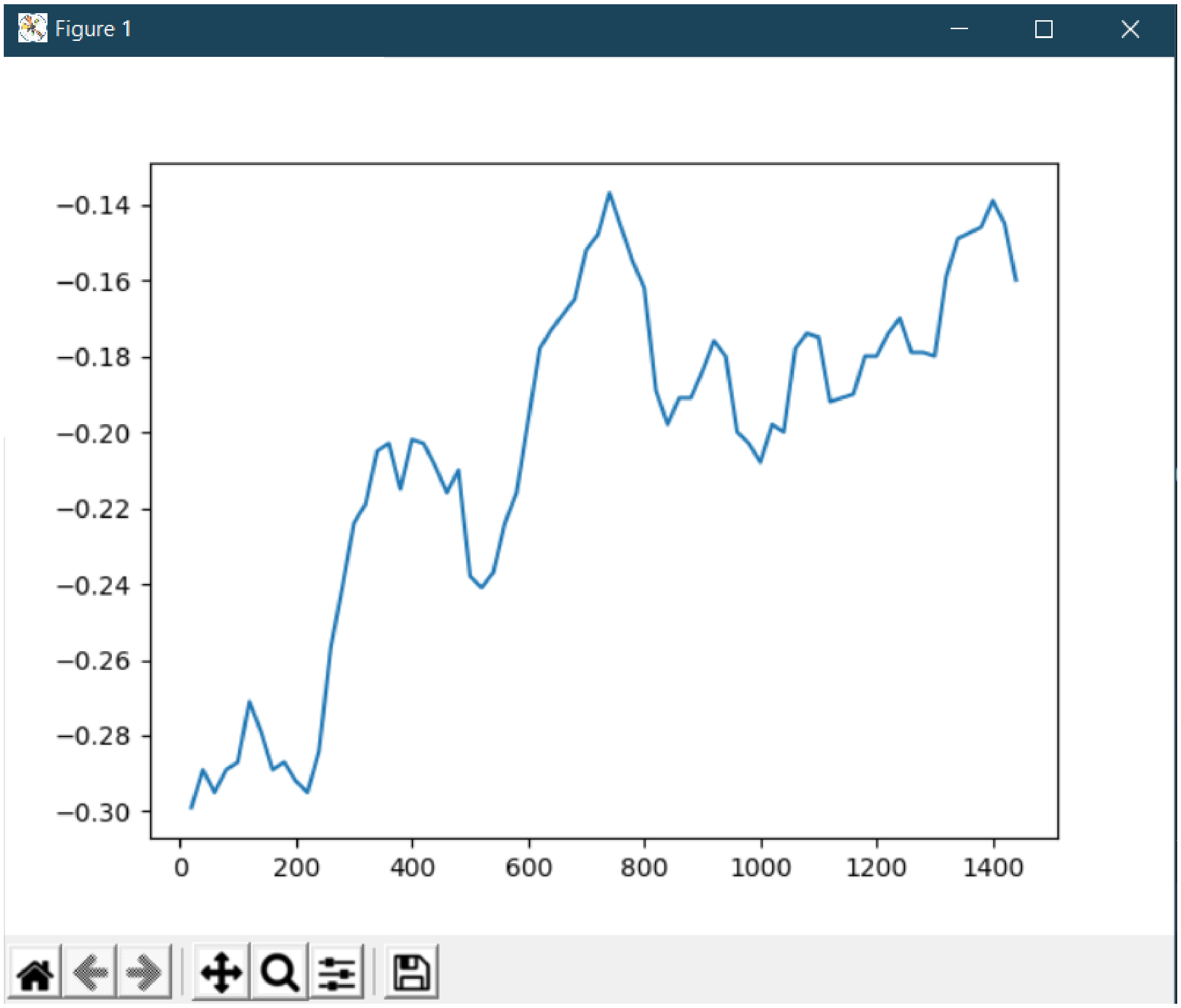
- **Input layer size:** the size of the observation
- **Hidden layer size:** 150
- **Output layer size:** 5

Q network:

- **Input layer size:** the size of the observation
- **Hidden layer size:** 150
- **Output layer size:** 1

For the training part we use q-learning a reinforcement learning algorithm with the q-table as a neural network. Each iteration is stored in a replay buffer until an episode is over. This replay buffer returns a random sample which is used for backpropagation in the target network. This ensures the independence assumption in stochastic gradient descent and that the q-network does not forget about older values during training.

Training results:



For each iteration we store the score, which tells us how well the agents performed. The higher the score is, the better the agents should perform on different configurations. After running multiple tests, we found this to be generally true, with few exceptions where lower scores provided better results than some higher ones. We can attribute this to the fact that the neural networks get stuck on a local minimum, instead of reaching the global minimum.

We split the tasks as follows:

- Balan Marius-Alexandru and Ivașcu Vlad-Alexandru modelling the neural networks and the agent class.
- Tcaciuc Ioan-Gabriel and Peltea Bogdan-Mihail processing the observations, the training and testing.