Flatland Challenge

G. Berselli, R. De Matteo, M. M. L. Pulici

July 19, 2021

Contents

1 Introduction

1.1 Flatland Challenge

The aim of the challenge [1], is to achieve efficient management of railway traffic. In particular, this problem is tackled in a simple grid world environment, in order to simulate and experiment different scenarios.

In more detail, the goal consists in making a number of trains arrive at their destinations minimizing the travel time. Even though for simple environments the train can follow explicit plans, as complexity increases the problem of mapping all possible states becomes intractable, for this reason a class of algorithms known as Reinforcement Learning (Section 1.2) can be exploited.
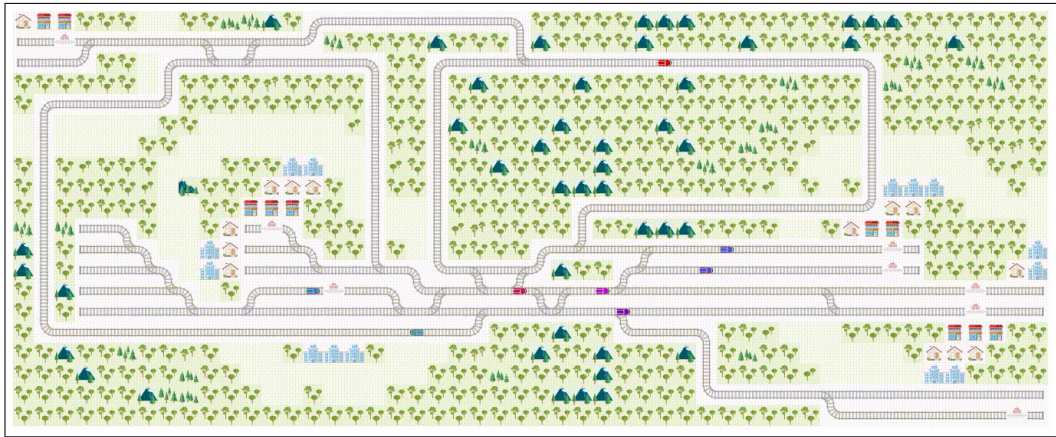
Figure 1: A possible Flatland instance.

1.1.1 The environment

The Flatland environment [2] consists in a discrete time simulation, meaning that each action is performed with a constant time step. At each step, an agent for each simulated train chooses an action. An agent is defined as "an entity that can move within the grid and must solve tasks" [2]. More precisely, each agent can choose between two actions: waiting or moving in a direction. Each agent has an individual starting position and its goal is to

reach its target destination. Of course, two agents can not occupy the same cell at the same time.

Each cell in the Flatland grid can take the form of any of 8 tile types, as shown in Fig. 2. More configurations can be obtained by rotating and mirroring the 8 basic tiles.
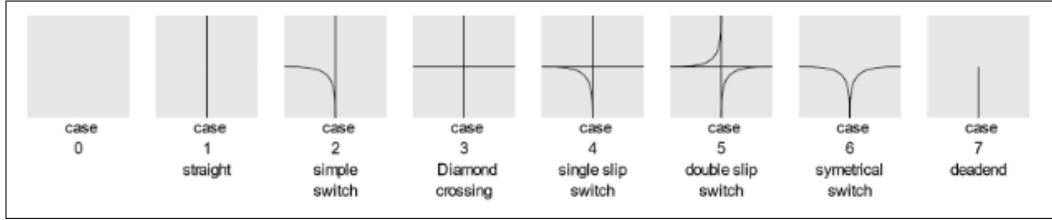

Figure 2: The 8 cell types.

When the tile is of the straight type, the agent can only choose to continue moving or to stop. In presence of a simple or a double switch, the train is forced to decide in which of the offered direction to move. When arriving at a dead end, an agent can only stop or go backward.

1.2 Reinforcement Learning

The term Reinforcement Learning [3] refers to the class of agents which rely on feedbacks, or rewards, produced by the agents themselves when interacting with the environment. The way Reinforcement Learning works is to use observed rewards to learn the best possible policy for the given environment. In other words, the agent has no prior knowledge of the environment and must learn how to behave based only on posterior trial-and-error feedbacks.

There are three basic agent designs for Reinforcement Learning:

- Utility-based agent, which learns a utility function on states and uses it to select actions;

- Q-learning, which learns an action-utility function giving the expected utility of an action given a state;

- Reflex agent, which learns a policy mapping directly from states to actions.

4

In addition to these designs, Reinforcement Learning can be either passive, meaning that the policy is fixed and the task is to learn the utilities of states, or active, referring to agents which must also learn how to act.

1.2.1 Q-learning

For the Flatland Challenge, Q-learning is used. This class of algorithms learns an action-utility representation instead of simply learning the utilities. The value of an action-state tuple is typically indicated as $Q(s, a)$. There is a direct correlation between Q-values and state utilities, as expressed by the following equation:

$$U(s) = \max_a Q(s, a). \tag{1.1}$$

The main difference between Q-functions and basic utility information si that no model of the form $P(s'|s, a)$ is needed, either for learning or for action selection. This characteristic makes Q-learning a model-free method.

At equilibrium, when the Q-values are correct, the following equation must hold:

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a'). \tag{1.2}$$

The issue with this approach is that the model of state transitions $P(s'|s, a)$ needs to be learned as well. To solve this problem, the so-called temporal-difference approach can be used. It consists in updating the Q-value every time an action is executed, using the following equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a) \right). \tag{1.3}$$

A close alternative to Q-learning can be found in the Sarsa (State-Action-Reward-State-Action) algorithm. The concept is very close to that of Q-learning, but Eq. (1.3) is replaced by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(R(s) + \gamma Q(s', a') - Q(s, a) \right). \tag{1.4}$$

The only difference between Eqs. (1.3) and (1.4) is that $\max_{a'} Q(s', a')$ is

simply replaced by $Q(s', a')$. In other words, instead of the maximum of the possible Q-values from the state reached in the transition, the actual value of the state s' reached after taking action a' is used. The value update takes place once at the end of each s, a, r, s', a' cycle, as suggested by the name.

If the agent is a greedy one, always choosing the action with the best Q-value, there is no difference between the two algorithms. If, on the other hand, there is some exploration happening, there is a crucial difference: Q-learning is said to be an off-policy algorithm, because it does not care about the policy used; SARSA, on the contrary, is considered on-policy. Even though Q-learning is more flexible, being able to learn well even with bad exploration policies, it is not as realistic, since it does not take into consideration possible external uncontrollable events.

1.2.2 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) [4] refers to a version of Reinforcement Learning where multiple agents interact in a common environment. There are three main broad categories which classify MARL:

- Cooperative: when all agents have a common goal and work together;

- Competitive: when agents compete to accomplish a goal;

- Mixed: when agents are grouped in teams, with intra-group cooperation and inter-group competition.

2 Implementation

2.1 Observation

2.1.1 Predictor

2.2 Deep Q-Network Agent

Learning using Deep Q-Networks (DQN) [5] is a relatively new paradigm, having been introduced in 2014. In general, the main feature of DQN algo-

rithms is the exploitation of Deep Neural Networks to learn the Q-values of a problem. There are many versions of DQN algorithms, of which the most common are:

- Fixed Q-targets;
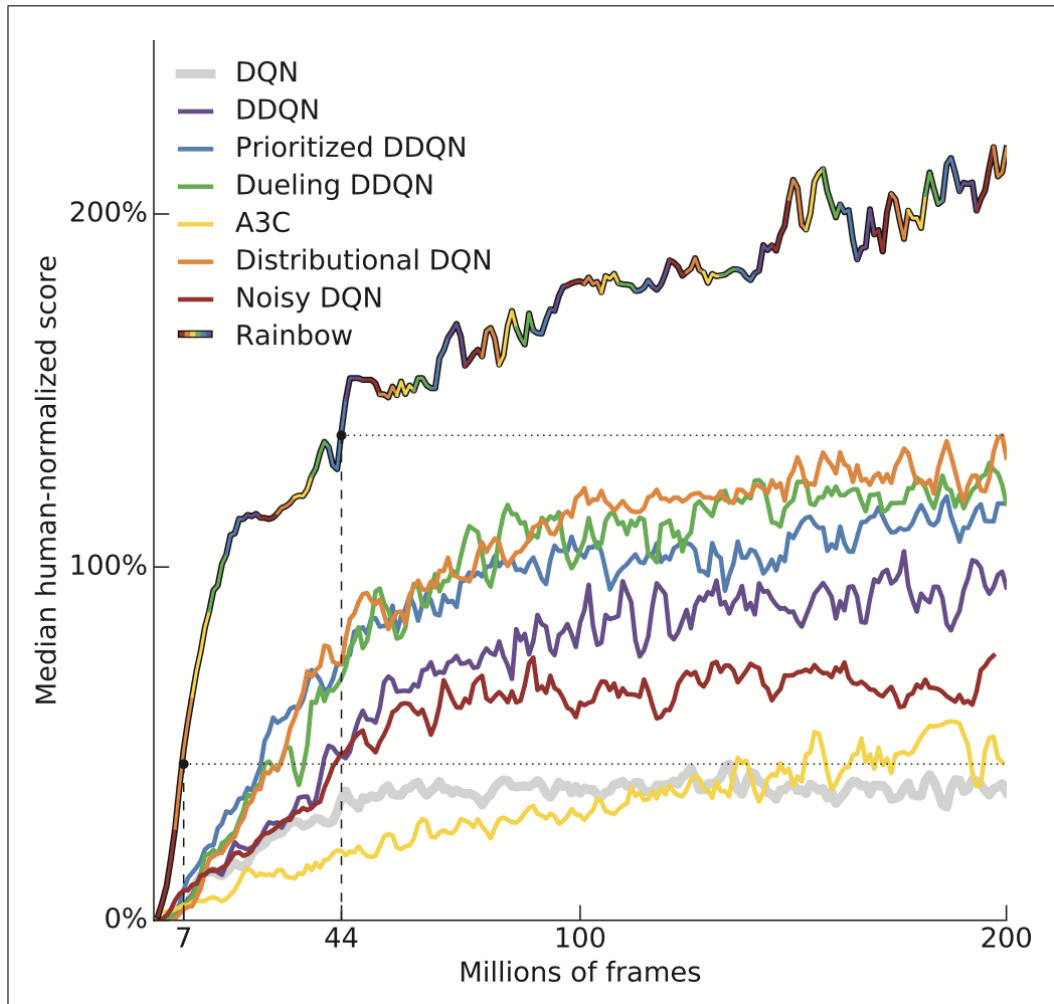
- Double DQN;

- Dueling DQN.

Figure 3: Comparison of various DQN approaches, including the state-of-the-art Rainbow agent.

2.2.1 Fixed Q-targets

When Eq. (1.2) is used in a Neural Network, the same parameters are used for estimating both the target and the Q-value. This is an issue, since it means that there is a big correlation between the target and the changing parameters: at every step, both the Q-values and the target values shift, leading to big oscillations in training.

A possible solution is to use fixed Q-targets, meaning that a separate network is created with a fixed parameter for estimating the target. Every T steps, the parameters are copied from the DQN to update the target network. This way, learning becomes more stable because the target function stays fixed for some time.

2.2.2 Double DQN

An alternative to simple Fixed Q-targets is the so-called Double DQN. The aim of this approach is to tackle the problem of checking that the best action for the next state is the one with the highest Q-value. In general, the accuracy of Q-values depend on both the action chosen and what states have been explored. Therefore, at the beginning of the training there is no information about the best action to take: taking the maximum Q-value can lead to false positives. This natural tendency of DQN to given higher Q-values to suboptimal actions complicates learning.

The Double DQN approach tries to fix this problem by using two separate networks to decouple the action selection from the target Q-value generation. First, the DQN network is used to select the best action to take for the next state, then the target network calculates the target Q-value of the state-action combination. So, the Double DQN method reduces Q-value overestimation and makes training faster and stabler.

2.2.3 Dueling DQN

The third DQN version is based on decomposing the Q-value in two parts:

$$Q\left(s, a\right) = V\left(s\right) + A\left(s, a\right) \tag{2.1}$$

where $V(s)$ is the value of being in state s and $A(s,a)$ is the advantage of taking action a at state s. Dueling DQN focuses on separating the estimator of the two values, using one stream to estimate the state value $V(s)$ and another one to estimate the advantage of each action $A(s,a)$. Decoupling these two values becomes particularly useful for states where any action does not affect the environment in a relevant way.

2.2.4 Further DQN approaches

Of course, these three methods are not the only ones to deal with DQNs. A particularly exhaustive analysis of all the different improvements to the DQN algorithm is represented by the seminal work by Hessel et al. [6]: in the paper, different DQN approaches are compared, culminating in the creation of a state-of-the-art integrated "Rainbow" agent, exploiting all the advantages of the most successful approaches. These results are shown as a reference in Fig. 3.

2.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [7], [8] is a technique designed to alternate between sampling data through interaction with the environment and optimizing an objective function using stochastic gradient ascent. The main feature of PPO is the use of the clipped surrogate objective:

$$L^{\mathrm{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \mathrm{clip}\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon \right) \hat{A}_t \right) \right]. \qquad (2.2)$$

Expectations are computed over a minimum of two terms: a normal objective and a clipped objective. Because of the min operator, the clipped objective behaves differently when the advantage estimate is positive or negative.

The effect of clipping is shown in Fig. 4. On the left situation, when the selected action has a better-than-expected effect, the loss function flattens out when the action is much more likely under the current policy compared to the old one. This is done to prevent overdoing an action update by taking
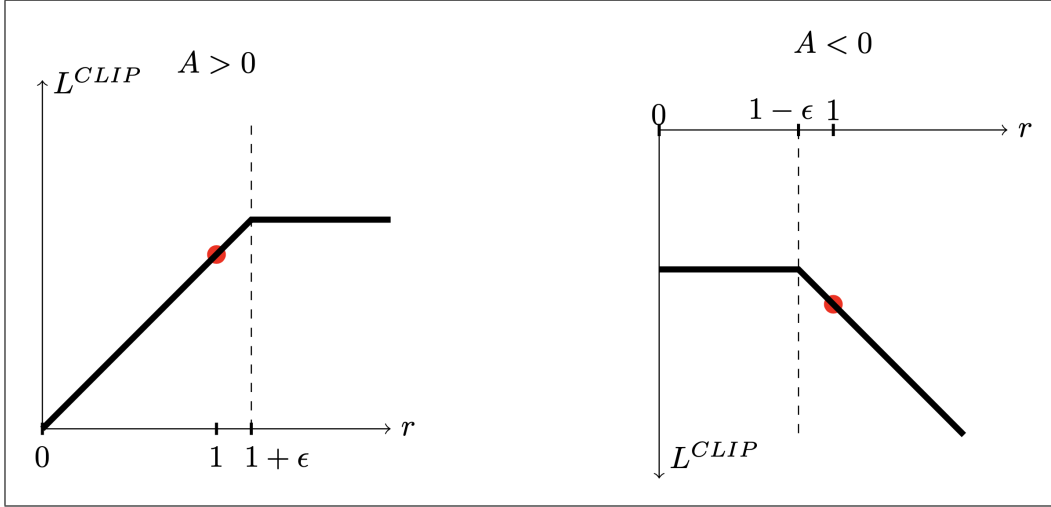
Figure 4: The L^{CLIP} function for positive advantages $A > 0$ and negative advantages $A < 0$. The red circles show the starting points for the optimization.

a step too far. The same happens to the graph on the right: the loss function flattens out when the action is much less likely under the current policy.

2.4 Action Selectors

In the learning process of a Reinforcement learning agent, exploration [9] plays a crucial role: in order for an agent to properly learn from the interaction with the environment, it must be exposed to as many states as possible. Since an agent needs the right experiences to learn a good policy, but also needs a good policy to obtain the environment, a balance, known as exploration-exploitation tradeoff, needs to be reached. There are various action selection approaches which can be used by the agent, of which the main ones are:

- Greedy Approach;

- Random Approach;

- ε-Greedy Approach;

- Boltzmann Approach;

- Noisy Approach.

2.4.1 Greedy Approach

The Greedy Approach is the most basic method of selecting an action. It consists in always opting for the action with the highest Q-value, regardless of the values of the other choices. This approach is illustrated in Fig. 5.
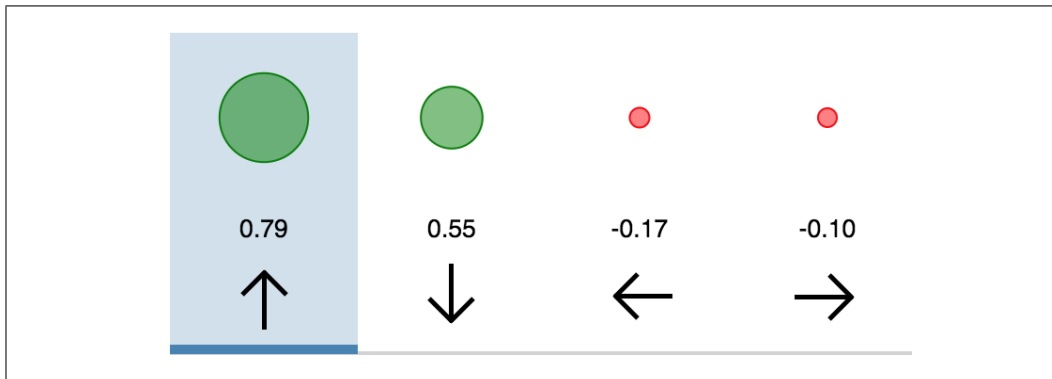

Figure 5: Action selection distribution for the Greedy Approach.

At first, this approach might appear good, as the agent always opts for the action it thinks to be the best. The main shortcoming of this method is that it almost always provides a suboptimal solution, since no alternate solutions are explored. In other words, in Greedy Approach, exploitation is favored enormously over exploration, which is almost absent.

2.4.2 Random Approach

The Random Approach is the extreme opposite of the Greedy one: instead of always checking for the best Q-value, the agent does not look at Q-values at all, picking randomly among all the possible actions. This approach is illustrated in Fig. 6.

Despite providing a lot of exploration, this approach is obviously deficient in exploiting the knowledge already learned by the agent.
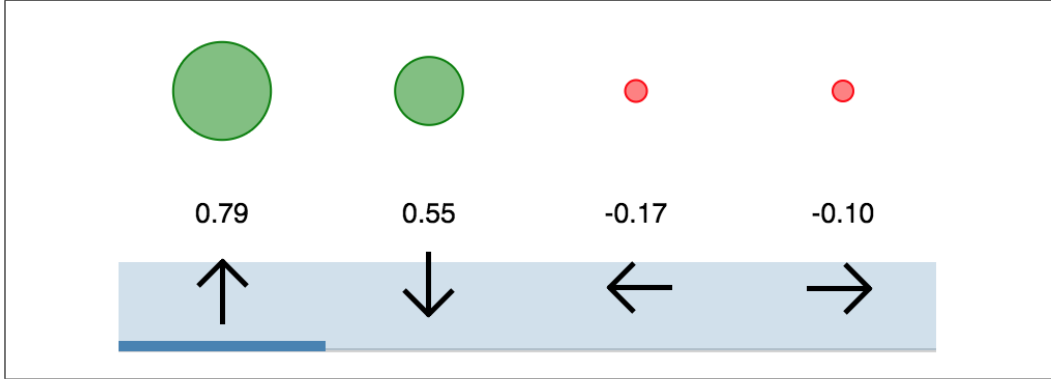
Figure 6: Action selection distribution for the Random Approach.

2.4.3 ε-Greedy Approach

The ε-Greedy Approach can be viewed as a combination of the Greedy and the Random ones. The way the agent acts in this case is by always opting for the optimal action, except occasionally it acts randomly. The choice between the two approaches is dictated by an adjustable parameter ε, which represents the probability to act randomly. This approach is illustrated in Fig. 7.
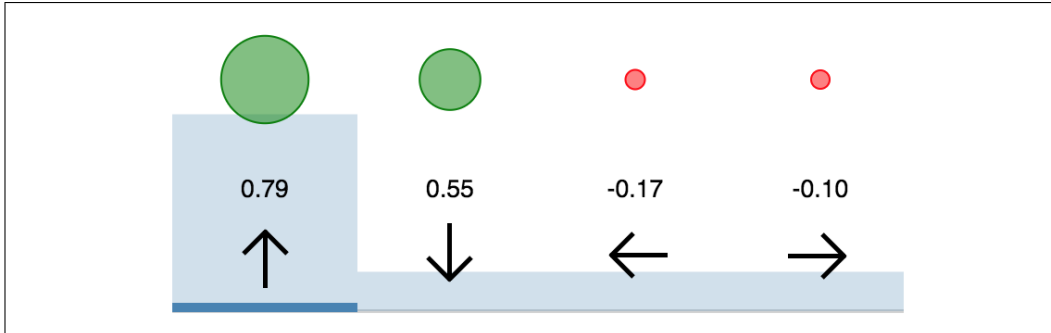

Figure 7: Action selection distribution for the ε-Greedy Approach.

This approach encountered a huge success due to its combination of simplicity and power: even though it is only a mixture of two very mediocre methods, the performance improvement is remarkable. To further enhance the agent's learning ability, the value of ε is often adjusted during training: in the beginning it starts as a big value, in order to provide maximum exploration, and it is slowly annealed as the agent obtains more information

12

about the environment. The only shortcoming of the ε-Greedy Approach is that it only takes into account whether an action is the most rewarding or not, making it not optimal.

2.4.4 Boltzmann Approach

The Boltzmann Approach takes the exploration-exploitation balance of ε-Greedy even further: instead of always taking the optimal action or acting randomly, it chooses among the various actions using individual Q-values to weigh probabilities. This approach is illustrated in Fig. 8.
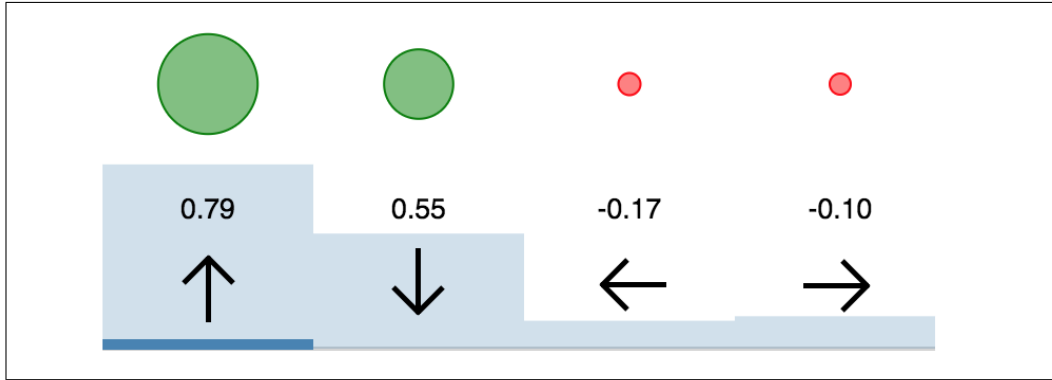

Figure 8: Action selection distribution for the Boltzmann Approach.

Compared to ε-Greedy, the Boltzmann Approach also takes into consideration the information about the values of actions other than the optimal: this way, actions which are potentially promising are given a higher priority over clearly inferior choices.

An interesting feature of the Boltzmann Approach is the use of an additional temperature parameter τ, which is annealed over time in a fashion similar to the way ε is treated. The τ parameter controls the probability distribution of the actions using the thermodynamics Boltzmann equation, which gives the name to the approach:

$$P\left(a\right) = \frac{\mathrm{e}^{\frac{Q(a)}{\tau}}}{\sum_{i=1}^{n} \mathrm{e}^{\frac{Q(a_i)}{\tau}}}, \tag{2.3}$$

where $P(a)$ is the probability of choosing action a, and a_i are all the possible action choices.

The main problem of this approach is that it builds on the assumption that the probability distribution outlined in Eq. (2.3) provides a measure of the agent's confidence in action a, while in reality what the agent is estimating is a measure of how optimal the agent thinks the action is, not how certain it is about that optimality.

2.4.5 Noisy Approach

The Noisy Approach [10] is somewhat different from all the others: instead of acting on the probability of the agent choices, it adds some noise to the output of the Neural Network itself. In general, the weights W_i are what the neural network needs to learn. Using the Noisy Approach, each weight can be expressed by the formula:

$$W_i = \mu + \sigma \cdot \varepsilon \tag{2.4}$$

where μ is a variable with random initialization, σ is a variable with constant initialization, and ε is the noise with a random value between 0 and 1.

2.5 Experience Replay

2.5.1 Prioritized Experience Replay

Prioritized Experience Replay [11] is a type of experience replay which consists in giving priority to transitions with high expected learning progress, measured by their temporal-difference error. This approach leads to two issues: firstly, the prioritization can produce a loss of diversity; secondly, it can introduce some bias.

The diversity loss problem can be solved by employing stochastic prioritization, a sampling method which interpolates between pure greedy prioritization and uniform random sampling: on the one hand, it ensures a monotonic sampling probability in a transition's priority; on the other hand, even the lower-priority transitions are guaranteed a non-zero probability. Practically,

the probability of sampling a transition i is given by:

$$P\left(i\right) = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}}, \tag{2.5}$$

where p_i is the priority of transition i and a is the amount of prioritization which is used.

The introduction of a bias can be corrected by using importance-sampling weight, given by:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P\left(i\right)}\right)^{\beta}, \tag{2.6}$$

which fully compensates for the non uniform probabilities if $\beta = 1$. For stability reasons, weights are normalized by $\frac{1}{\max_i w_i}$, si that they only scale the update downwards.

3 Results

Experiment tracking was performed using the Weights and Biases tool [12].

4 Conclusion

References

[1] (2021). Flatland, AIcrowd, [Online]. Available: `https://www.aicrowd.com/challenges/flatland` (visited on 07/19/2021).

[2] (2020). Flatland Challenge, AIcrowd, [Online]. Available: `https://www.aicrowd.com/challenges/flatland-challenge` (visited on 07/19/2021).

[3] S. J. Russell and P. Norvig, "Reinforcement Learning", in *Artificial Intelligence, A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Pearson Education, 2009, ch. 21, pp. 830–859.

[4] P. Haou. (2021). Multi-Agent Reinforcement Learning (MARL) and Cooperative AI, Medium, [Online]. Available: `https://towardsdatascience.com/ive-been-thinking-about-multi-agent-reinforcement-learning-marl-and-you-probably-should-be-too-8f1e241606ac` (visited on 07/19/2021).

[5] T. Simonini. (2018). Improvements in Deep Q Learning: Dueling Double DQN, Prioritized Experience Replay, and fixed..., freeCodeCamp, [Online]. Available: `https://www.freecodecamp.org/news/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682/` (visited on 07/19/2021).

[6] M. Hessel, W. Dabney, J. Modayil, D. Horgan, H. van Hasselt, B. Piot, T. Schaul, M. Azar, G. Ostrovski, and D. Silver, "Rainbow: Combining Improvements in Deep Reinforcement Learning", in *Proc. 32nd AAAI Conference on Artificial Intelligence*, (New Orleans, Louisiana, USA, Feb. 2–7, 2018), pp. 3215–3222. [Online]. Available: `https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/download/17204/16680`.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms", OpenAI, Aug. 28, 2017. arXiv: `1707.06347 [cs.LG]`.

[8]   T. Kim. (2021). Understanding Proximal Policy Optimization (Schulman et al., 2017), Medium, [Online]. Available: `https://towardsdatascience.com/understanding-and-implementing-proximal-policy-optimization-schulman-et-al-2017-9523078521ce` (visited on 07/19/2021).

[9]   A. Juliani. (2016). Simple Reinforcement Learning with Tensorflow Part 7: Action-Selection Strategies for Exploration, Medium, [Online]. Available: `https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-7-action-selection-strategies-for-exploration-d3a97b7cceaf` (visited on 07/19/2021).

[10]   P. H. Moghadam. (2019). Deep Reinforcement learning: DQN, Double DQN, Dueling DQN, Noisy DQN and DQN with Prioritized Experience Replay, Medium, [Online]. Available: `https://medium.com/@parsa_h_m/deep-reinforcement-learning-dqn-double-dqn-dueling-dqn-noisy-dqn-and-dqn-with-prioritized-551f621a9823` (visited on 07/19/2021).

[11]   T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay", presented at the 4th International Conference on Learning Representations (Caribe Hilton, San Juan, Puerto Rico, May 2–4, 2016). arXiv: `1511.05952` `[cs.LG]`.

[12]   *Weights and biases*, Weights and Biases, 2021. [Online]. Available: `https://wandb.ai/site` (visited on 07/19/2021).