Inverse Reinforcement Learning to give desired behaviors to autonomous agents in MiniGrid

Federico Magnolfi
Università degli Studi di Firenze
Florence, Italy
federico.magnolfi2@stud.unifi.it

Francesca Del Lungo
Università degli Studi di Firenze
Florence, Italy
francesca.dellungo2@stud.unifi.it

***Abstract*— All humans, especially children, learn how to perform tasks by imitating other humans: they deduce the actions to take from what they observe. Using *Inverse Reinforcement Learning* (IRL), autonomous agents may do so as well.**

This study is based on T-REX [2], an algorithm that uses a particular loss to learn reward functions from a set of ranked demonstrations. In this work, we apply T-REX to achieve desired agents' behaviors in the MiniGrid[3] environment of OpenAI Gym[5]. We obtain good results for simple trajectories in the empty environment.

We also build a graphical tool that allows users to create agents with desired behaviors, without requiring them any knowledge in the *Machine Learning* field.

I. Introduction

In a traditional *Reinforcement Learning* (RL) setting, an agent interacts with the environment executing actions and gaining rewards. The goal of the agent is to learn a mapping (*policy*) from perceptions to actions, to produce a behavior that maximizes the gained reward. In this situation, the learner is not told which actions to take, as in many forms of machine learning, but instead, he has to find out which actions produce maximum performance (*reward*) by trying them out.

In *Inverse Reinforcement Learning* (IRL) the problem is reversed: the goal is to learn the environment reward function from the observed behavior of an agent. IRL is very useful when the reward function is difficult to specify manually. Usually, the reward signal is iteratively tweaked by hand until the RL agent shows the desired behavior. A better way of designing a reward function might be to observe a (human) expert performing the task and then automatically extract the reward from these observations: this is what IRL is about.

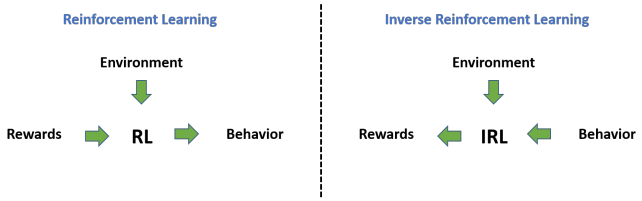

Fig. 1. High-level characterization of RL and IRL tasks.

In this work, we are interested in using IRL to learn rewards from human demonstrations. Based on *T-REX*[2] we tried to apply this technique on the *MiniGrid* environment [3]. Starting from a set of demonstrations, ordered from worst to best according to the task you want to achieve, the *T-REX* loss allows you to learn the reward function. We have created a graphical interface that allows the user to play to create demonstrations and to create agents starting from the demonstrations created, specifying the order of them.

II. Inverse Reinforcement Learning

Let S be the space of environment states and A the space of policy actions. In a Markov decision process (MDP), the immediate reward $R(s, a, s')$ is received after transitioning from state $s \in S$ to state $s' \in S$, due to action $a \in A$. Therefore, a possible reward estimation approach would be to try to approximate the mapping $(s, a, s') \rightarrow R$. The problem of this approach is that even in a simple environment there are usually many different states and so the space $S \times A \times S$ has a very high cardinality.

Trying to learn a function whose domain is such

a large space involves various problems: learning requires more data and more time, and there is also the risk of learning very irregular functions, with no good-generalization properties. For this reason, there are approaches that, to estimate the reward, use only the previous state s and the action taken a, or approaches that use only the arrival state s'. T-REX belongs to the latter group.

To summarize, we want to learn the **reward function** $R(s')$ which maps the arrival state to a real-valued reward.

A. T-REX

Given a sequence of m demonstrations ranked from worst to best $\tau_1, ..., \tau_m$, each made by a sequence of states, we want a reward function $R_\theta(s)$ s.t.

$$\sum_{s \in \tau_i} R_\theta(s) < \sum_{s \in \tau_j} R_\theta(s)$$

when $i < j$.
That is, a better trajectory should have a higher return (sum of rewards) than a worse trajectory. The probability of a trajectory j being better than a trajectory i is obtained with a softmax using the returns as logits. The function $R_\theta(s)$ is approximated by a *neural network* parameterized by θ. The T-REX loss $\mathcal{L}(\theta)$ is a cross entropy loss that considers each pair of trajectory:

$$\mathcal{L}(\theta) = -\sum_{i<j} \frac{\exp \sum_{s \in \tau_j} R_\theta(s)}{\exp \sum_{s \in \tau_i} R_\theta(s) + \exp \sum_{s \in \tau_j} R_\theta(s)}$$

III. MiniGrid Gym environment

In our project we used the *MiniGrid* [3] grid-world environment: it's a simple, lightweight and fast environment for *OpenAI Gym* [5]. The world is represented by an NxM grid of tiles, and each tile contains zero or one object (Fig. 2). The agent can move in the world to reach the green goal square, possibly interacting with other objects.

There are many different environments of various sizes and with different objects inside.

In Minigrid the observations are dictionaries, with an 'image' field that is a partial view of the environment, a 'mission' field which is a textual string describing the objective the agent should reach to get a reward, and a 'direction' field.

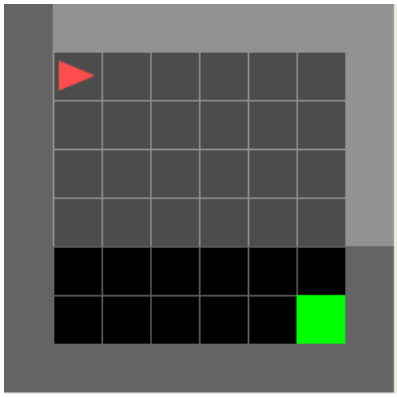

Fig. 2. Example of an empty Minigrid environment.

In particular, the 'image' field consists of 3 7x7 matrices of integer values: the first represents the objects seen by the agent in the current state, the second the colors of the objects and the last one provides information on keys and doors (if they are present).

Due to its simplicity, this kind of gym environment is particularly suitable for testing reinforcement learning and inverse reinforcement learning algorithms.

IV. Our work

As specified in [2], we want to solve the problem of inverse reinforcement learning from observations, where we do not have access to the reward function of the MDP nor the actions that were taken by the demonstrator. A set of sorted demonstrations is available: each demonstration is just a sequence of states. In our work, we:

1) experiment the T-REX loss on the MiniGrid Gym environment
2) build a graphical tool that allows users creating agents with desired behaviors

The training pipeline to create a new agent is as follows:

1) train a reward net in the given environment with given ranked demonstrations
2) train a policy net in the given environment with rewards given by the previously trained network

A. Graphical User Interface

Our graphical tool allows the users to create agents with desired behaviors in the MiniGrid

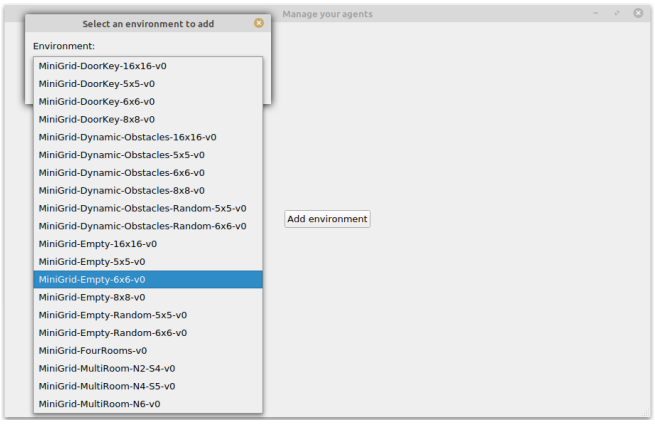

Fig. 3. *Initial window*: here you can select an environment to add from the list of available environments.

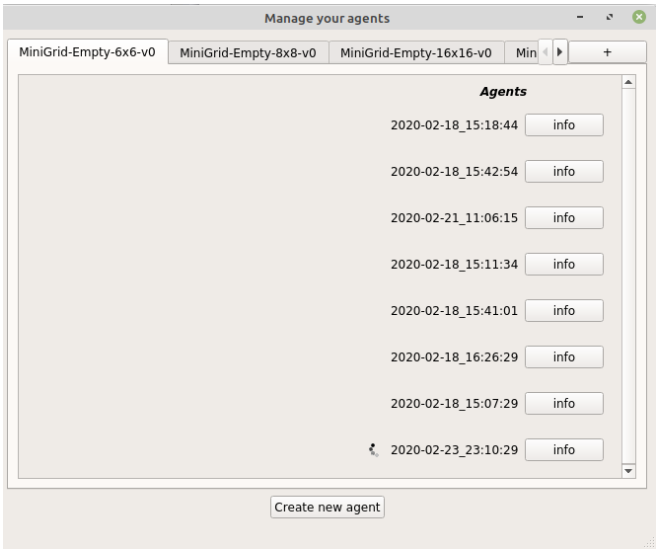

Fig. 4. *Agents window*: there is a list of all the agents created for the selected environment. For each agent there is a button to see its details. The environment can be selected by clicking on tabs.

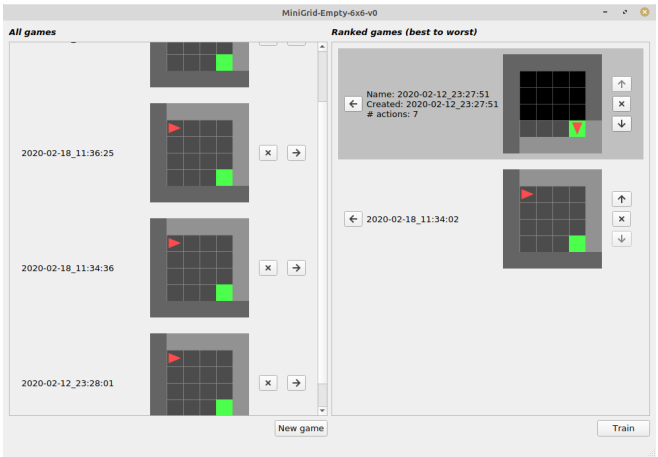

Fig. 5. *Agent creation window*: on the left, there is a list of all the games played so far in this environment. On the right, there is the ranked list of games chosen to train the agent. Use the new game button to create a new game. The training starts when clicking on the train button.

environment. The user can create new example games, order them according to "goodness" and then create a new agent starting from the selected games.

The first thing the user has to do in the **initial window** is choosing the environment he wants to consider: 20 different environments can be selected (Fig. 3). Then in the management window (Fig. 4) you can see: a different tab for each environment, all the existing agents for the selected environment and a button to create a new agent. A small symbol next to the agent's name tells if the agent's policy training has finished or is still in progress. Training details window can be opened by clicking on the info button.

In the window dedicated to the **creation of a new agent** (Fig. 5) you can see two lists of games: on the left the one with all the games created so far (of the selected environment) and on the right the ranked games. The latter list contains the games that, after having been ordered by the user from best to worst, are used to train the new agent. When the user has the mouse over a game some extra information is displayed, such as the number of steps in the game. To create the ranked games list the user can move each game from the game list on the left to the ranked games list on the right (that is initially empty) and back. It can be done through the dedicated left and right arrow buttons. Moreover, the up and down arrows on the ranked games list allow the user to change the order of the games.

And finally, when all the games have been selected and ordered, the user can proceed with agent training. All the information related to the selected agent is shown in the **agent details window** (Fig. 6): here on the right there is a list of all the games used to train the agent (in the user-defined order) and on the left, there is a progress bar that shows the status of the agent, in particular, the percentage on the bar shows the progress of

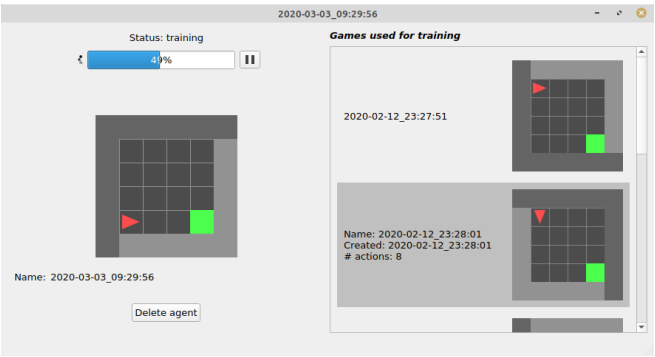

Fig. 6. *Agent details window*: on the left, there is the progress of the current training, which can be paused or resumed. The current policy behavior is displayed below, and previous behaviors can be seen after pausing the training. On the right there is the list of ranked games used to train this agent.

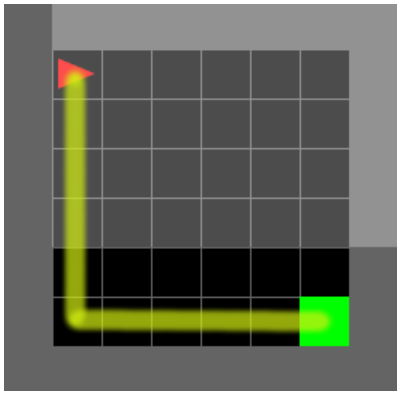

Fig. 7. Desired sub-optimal trajectory. This is the MiniGrid-Empty-8x8 environment.

the training process. The current policy behavior is shown below the progress bar. The status of the agent can be in training or trained; in the former, the user can pause the training through the button next to the training bar and a slider appears below the progress bar. The slider can be used to show the behavior of the agent at different steps of the policy training (using the training policy checkpoints saved so far).

V. Experiments

In the proposed experiments we are interested in finding a reward network capable of learning the reward from the demonstrations. To evaluate the quality of the learned reward, we use different criteria reported in V-A. We tried different reward networks, the final one is reported in V-B. Obtained results are reported in V-C.

In the empty grid environment, the agent's starting position is the cell in the upper left corner, while the goal state is in the lower right corner. In such configuration, the *optimal trajectory* is the one where the agent goes directly to the upper right corner and then down to the goal cell. This is optimal because it is the trajectory that requires the least number of steps to reach the goal.

We decide to **train an agent to follow a different trajectory** than the optimal one, in the *MiniGrid-Empty-16x16* environment. In particular, we want to train the agent so that the desired trajectory is the one "opposite" to the one described above: we want the agent to go down to the left corner and then right to the goal square (intuitively analogous to the one in Fig. 7 relative to MiniGrid-Empty-8x8 environment). This is just an example of a trajectory that is sub-optimal, there could be many other different choices, but it is simple to evaluate.

To force this behavior we create 11 different trajectories ranked so that the best one is the trajectory showing the desired behavior and the worst one is the optimal trajectory (described above). Other trajectories are created and used in the ranking between the best and the worst ones; all these "intermediate" trajectories are ordered so that the first ones are more similar to the desired trajectory and gradually become worse and worse.

A. Evaluation

In order to evaluate the quality of the learned reward, we use different evaluation criteria:

1) *ordering_quality*: we define this metric as the percentage of correctly ranked pairs of trajectories. A pair (τ_i, τ_j) with $i < j$ is correctly ranked *iff* $\sum_{s \in \tau_i} R_\theta(s) < \sum_{s \in \tau_j} R_\theta(s)$.
2) *heatmap*: is a graphical representation of data where the individual values contained in a matrix are represented as colors. In our experiments it is very useful as it offers the opportunity to easily visualize and compare the rewards obtained in each cell of the grid.
3) *policy's behaviour* after training: this can be an excellent indicator of the "goodness" of the learned reward.

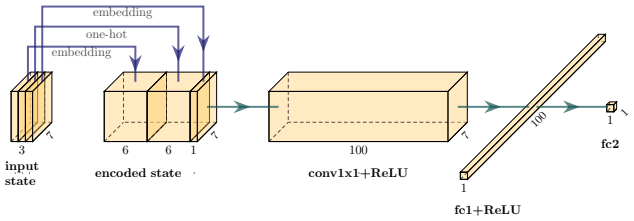

Fig. 8. Reward neural network

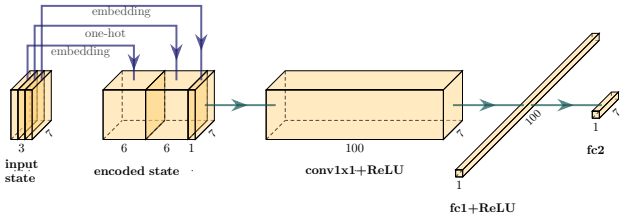

Fig. 9. Policy neural network

B. Networks

We create a *reward network* (Fig. 9) and a *policy network* with a similar structure. The networks transform the environment observation in input in the same way: first channel (objects) and third channel (objects states) values are encoded using an embedding technique, while the second channel (colors) is transformed using one-hot encoding. After that, both networks have a 1×1 2D convolutional layer with 100 neurons, a ReLU activation layer, a fully-connected layer with 100 neurons, and another activation layer.
The difference between the two networks is in the number of neurons in the last fully connected layer: the reward net has 1 neuron (the reward is the only output), whereas the policy net has 7 neurons (one for each possible action).

C. Results

After training the reward net in *MiniGrid-Empty-16x16* with target trajectory analogous to the one in Fig. 7, we create a heatmap for each direction: down (Fig.10), left (Fig.13), right (Fig.11), up (Fig.12). As we can intuitively interpret, the more intense the color the greater the reward value in the cell.

When the agent is in the starting cell (top left corner of the grid) he gets the highest reward when

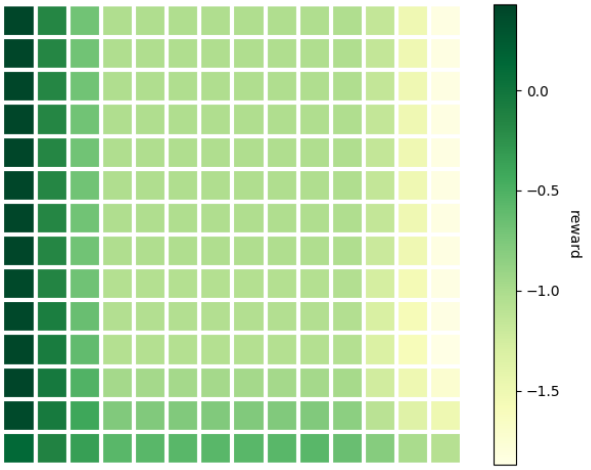

Fig. 10. Reward intensity in each cell of the 16x16 empty environment, when the agent direction is *down*.

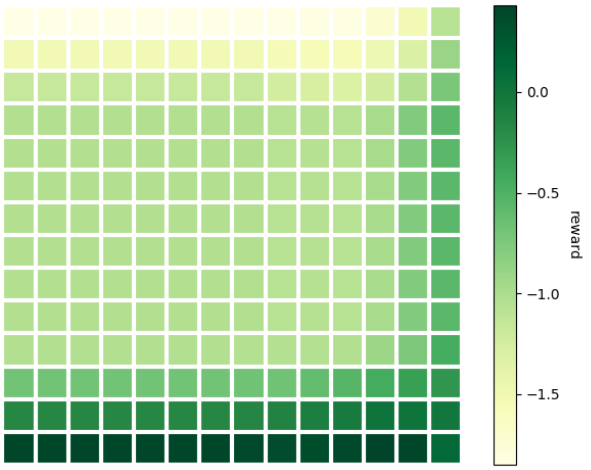

Fig. 11. Reward intensity in each cell of the 16x16 empty environment, when the agent direction is *right*.

facing *down*. Instead, when the agent faces right, the reward is negative.

When the agent is at the bottom of the grid, the right is the direction with the highest reward. This is what we expected. This is also confirmed by Fig.15 that shows the most rewarded direction in each cell of the grid environment.

If on the one hand, the rewards are positive in the directions in which we are particularly interested, on the other hand, most of the rewards values are negative: this has a positive effect, in fact this leads the agent to go in the desired trajectory by discouraging take other paths.

All the heatmaps look very similar, this could

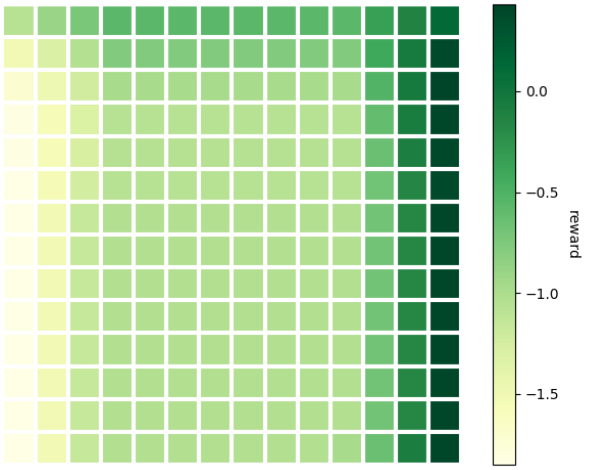

Fig. 12. Reward intensity in each cell of the 16x16 empty environment, when the agent direction is *up*.

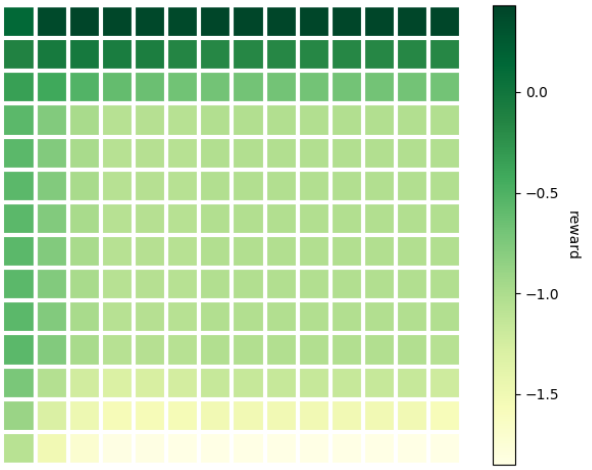

Fig. 13. Reward intensity in each cell of the 16x16 empty environment, when the agent direction is *left*.

seem a bit strange but it's not. In fact, when the agent from the initial cell reaches the goal following the desired trajectory, it describes a trajectory almost identical to the one that, starting from the goal, goes up to the upper-right corner and then left to the initial cell. The observations of the agent (7x7 matrix) in the cells of the "returning trajectory" will be almost the same as the observations in the desired trajectory, since for the majority of the states the goal is not visible.

As for the policy, after the training with the previously learned reward, doing the *most-probable* action at each step leads the agent along the desired trajectory. Instead, if we *sample* the action from the policy distribution, the desired trajectory (28-steps long) is covered $\sim 70\%$ of the times. The bar chart in Fig.14 represents the number of times the policy made paths of certain lengths. The experiments have been done on a total of 1000 sampled trajectories.

We can notice that the second higher bar in the chart is the one relative to the 30-steps long trajectory: this happens when the agent turns left three times instead of turning right before going down to the bottom of the grid. This policy has a not-negligible turning left probability in the first cell because turning left is overall a good action: it's required to follow the desired trajectory, and it's not done in the optimal trajectory that we want to avoid.
Trajectories of higher lengths correspond to paths where the policy makes more "laps" of the grid to get to the goal.

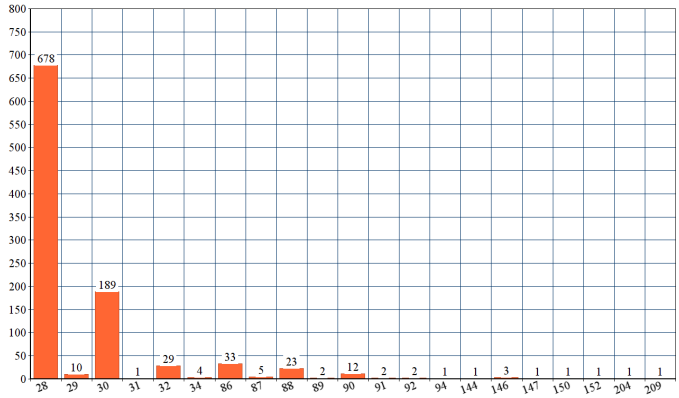

Fig. 14. Histogram of trajectory lengths. 1000 trajectories have been generated by sampling the actions from the policy network. On the x-axis: path length. On the y-axis: number of paths with this length.

D. Training details

For both reward and policy networks, here we report the details of the trainings to carry the experiments. Relative plots can be found in the appendix.
During the training of the reward network, we evaluate the *ordering_quality* metric, i.e. the percentage of correctly ranked pairs of trajectories

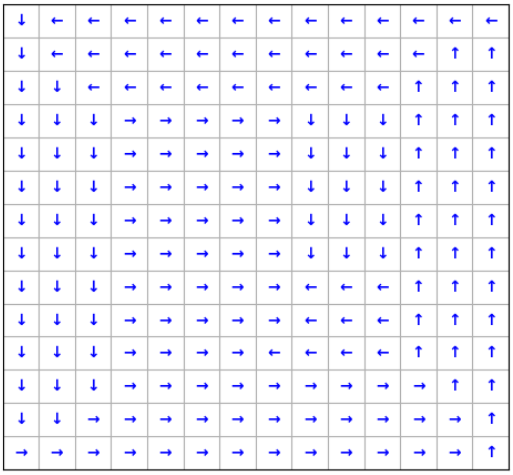

Fig. 15. For each cell of the 16x16 empty grid environment it's displayed the higher-reward direction in that cell.

(Fig.16). The value of this metric grows during training until reaches its maximum of 94%.
During the training of the policy network, we evaluate both the *trajectory length* and the *trajectory return* (Fig.17). As the episodes progress, the number of steps decreases, while the reward obtained increases. The return is widely negative at the beginning, but it becomes slightly positive in the end. The trajectory length is very high in the beginning until it almost converges to the length of the desired trajectory, which is 28.

VI. Conclusions

In this work, we tried to learn the reward function from human demonstrations. We have experienced that learning the reward for simple trajectories is a feasible task. As a case study, we took a specific type of trajectory in an Empty MiniGrid environment. The obtained rewards are consistent with the desired trajectories and are also very dense. Having a dense reward can help to speed up the learning of the policy, with appropriate algorithms: detailed studies in this regard could be done in the future.

Finally, we created a graphical user interface that allows users to create and train their agents, using their demonstrations, without having to know the details of the underlying learning algorithms.

References

[1] Stefan Schaal. Learning from demonstration. Advances in neural information processing systems, 1997.

[2] Daniel S. Brown, Wonjoon Goo, Prabhat Nagarajan, Scott Niekum. Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations, July 2019.

[3] Chevalier-Boisvert, Maxime and Willems, Lucas and Pal, Suman. Minimalistic Gridworld Environment for OpenAI Gym. https://github.com/maximecb/gym-minigrid, 2018.

[4] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction, 2014.

[5] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W. (2016). OpenAI Gym. ArXiv, abs/1606.01540.

VII. Appendix

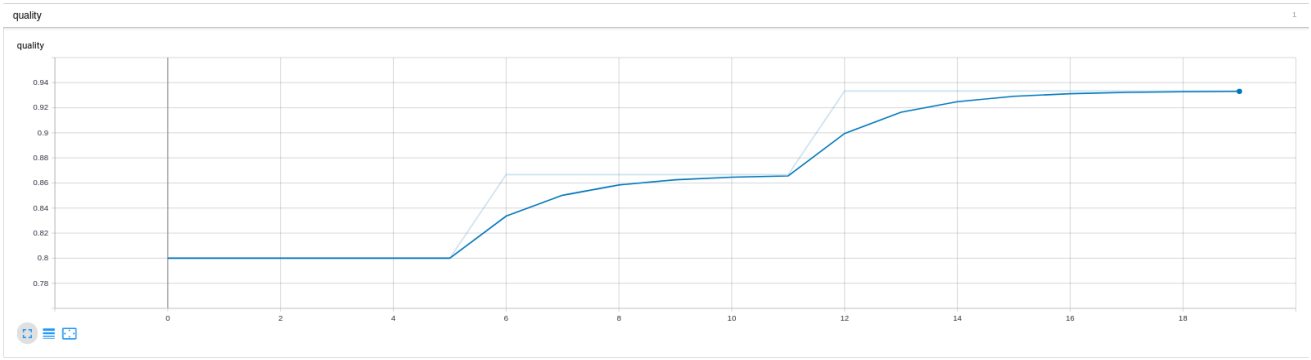

Fig. 16. Monitoring of the *ordering_quality* metric during training of the reward network. It grows until reaches 94%.

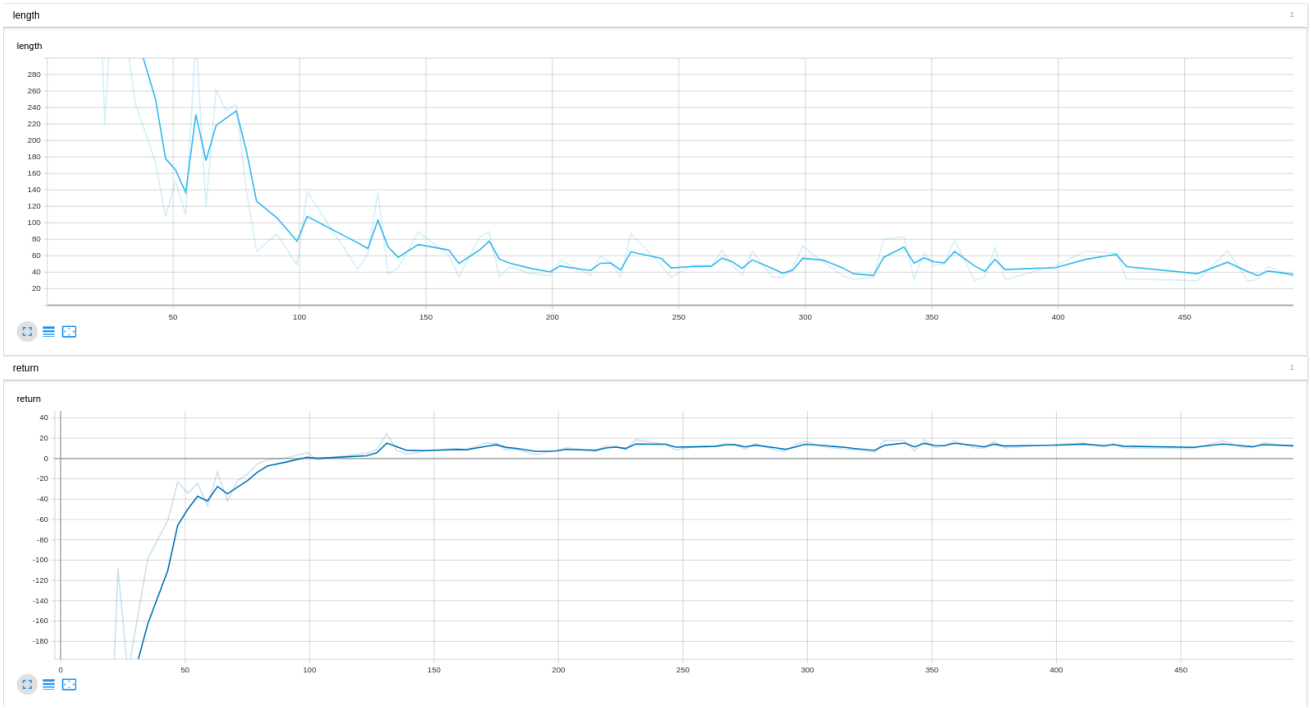

Fig. 17. Monitoring of the *trajectory length* (above) and of the *trajectory return* (below) during training of the policy network. The former descends approaching the desired trajectory length (28), while the latter grows to become slightly positive.