

Comparison Q-Learning and CRM

In this little report we will compare the performance of Q-Learning and the CRM algorithm developed for reward machines on a simple environment to see how they differ.

DoorKey Environment

The DoorKey environment comes from the minigrid package that builds grid environments compatible with gymnasium, the reinforcement learning library.

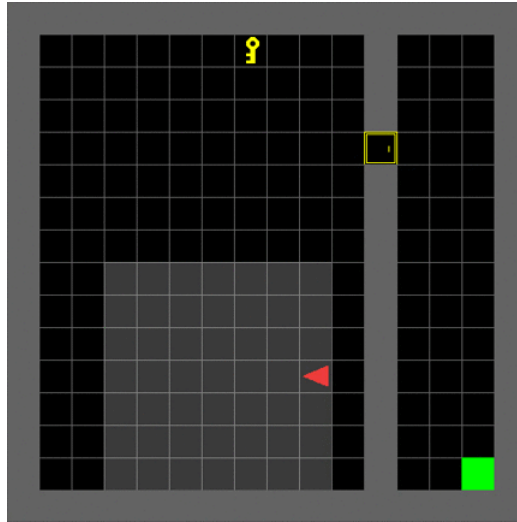


Figure 1: DoorKey environment

The environment consists of an $n \times n$ grid which is separated by a wall with a locked door. The agent starts in one half and in the other half is the exit. In order to reach the exit the agent must pick up a key to unlock the door and get to the other side. Reaching the exit is the goal and upon succeeding the agent is rewarded with 1. If the agent does not reach the goal in a specified number of steps then it is rewarded with 0. See figure 1 for an example of the DoorKey environment.

There are seven available actions for the agent: turning left and right, stepping forward, picking up the key, dropping the key, opening/closing the door and doing nothing. The state the agent receives is its current direction and its field of view, which is a $m \times m$ box in front of the agent facing in the

same direction but is blocked by walls. The tiles in the field of view consist of IDs, colors and a possible state, e.g. open, closed and locked for the door. For convenience this state is encoded as a single integer using the MD5 hashing function to make it immediately usable in the algorithms below.

Q-Learning

The baseline we want to compare the reward machine implementation against is simple Q-Learning.

The Q-Learning table Q is initialized with uniformly random values in $[0, 1]$. And updated using the Bellman equation

$$Q(s_t, a_t) \leftarrow (1 - \alpha) Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) \right)$$

where s_t is the state, a_t the action and r_t the reward at time t , α is the learning rate and γ the discount factor.





During training the action is selected using ϵ -greedy action selection, where with probability ϵ a random action is selected and otherwise the the current best action according to table Q . Each training episode the ϵ value is decaying exponentially until it reaches a minimal value by $\epsilon \leftarrow \max(\epsilon_{\min}, \delta\epsilon)$. During testing we only exploit the Q table.

The parameters for our implementation of Q-Learning are the following:

- $\alpha = 0.1$
- $\gamma = 0.9$
- $\epsilon = 1$
- $\delta = 0.995$
- $\epsilon_{\min} = 0.01$

CRM

In order to use "Q-Learning with counterfactual experiences for reward machines" (CRM) algorithm described in the paper "Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning" from 2022, we need to construct a reward machine. In figure 2 below you see the reward machine as a diagram with states u_0 , u_1 and u_2 . Each edge is labeled with the proposition that has to be fulfilled and the reward that the agent is rewarded in that case. The propositional symbols used are

-  ... agent has picked up the key
-  ... door is opened
-  ... timeout, i.e. maximum number of steps reached
-  ... exit, i.e. the goal

So state u_0 basically is the agent trying to acquire the key, u_1 trying to open the door and u_2 reaching the exit.

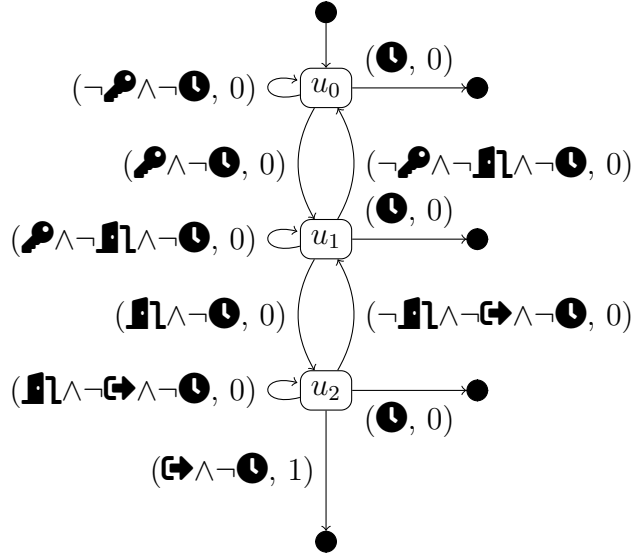


Figure 2: Reward machine for the DoorKey environment

The idea of the CRM algorithm is to use the different states of the reward machine to learn from counterfactual experiences, i.e. the Q -table is updated for all reward machine states given the current experience (s_t, a_t, s_{t+1}) and not just the current one, which should speed up the convergence.

The hyperparameters are the same as in the Q-Learning case.

Results

We run both the Q-Learning and CRM algorithm on the 5×5 DoorKey environment as well as the 6×6 one for 1000 and 10000 episodes respectively. We tracked the training error, i.e. the temporal difference error $r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$, as well as the total reward for each episode, so in this case either 0 or 1, and the number of steps taken. We repeated this experiment 100 times and plotted the average below.

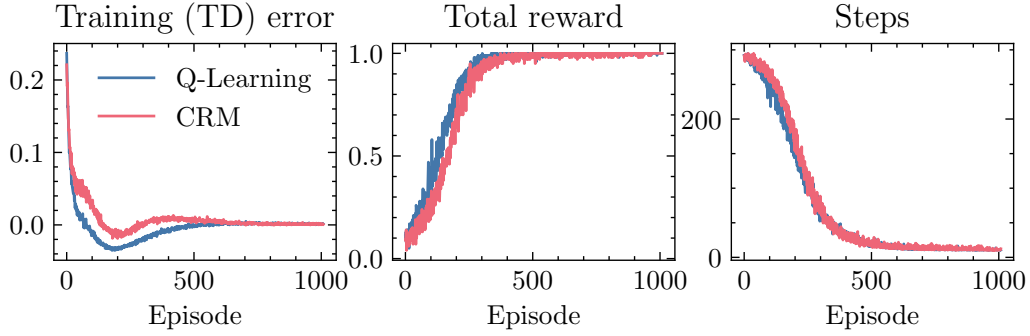


Figure 3: Average over 100 runs of the 5×5 DoorKey environment

As seen in figure 3 in roughly the first 500 episodes the training error fluctuates but settles close to 0 afterwards for both algorithms. Since the total reward can only be 0 or 1 in this case, it can be seen as a percentage of how many runs achieved the goal in that episode. Thus the Q-Learning converges slightly faster to the optimum than CRM contrary to our expectation. Even less pronounced is the advantage of Q-Learning in the number of steps. They both settle at about 10 steps to finding the exit in the end.

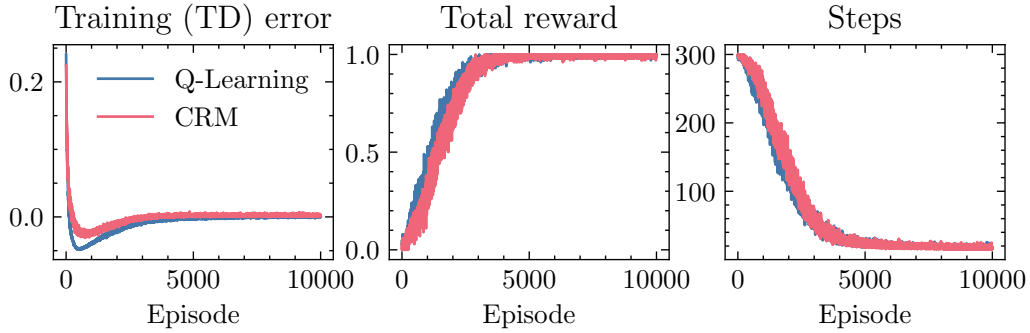


Figure 4: Average over 100 runs of the 6×6 DoorKey environment

A similar picture can be seen in figure 4 for the 6×6 environment, but needed 10 times more episodes to that of the 5×5 environment. The result is basically the same but with a little more variance. The steps it takes for the converged agents to find the exit is 17 for Q-Learning and 16 for CRM. So perhaps CRM finds a better strategy.

Conclusion

???

References

- My implementation on GitHub
- Parts taken from original implementation by Rodrigo Toro Icarte and Toryn Klassen on GitHub
- Gymnasium by Farama foundation
- Minigrid by Farama foundation