TME 12: # Implicit Curriculum RL

In this TME, we explore the methods of Implicit Curriculum RL for sparse-rewards environment.

DQN with Goal

Our first environment `plan2Multi` has a method `sampleGoal` that uniformly samples an intermediate state-goal among a set of predefined goals. With that in mind, we can reuse a simple Double DQN agent: instead of learning a mapping from state-action pairs to values, the agent now learns a mapping from an 3-uplet state, action, goal. In practice, we simply concatenate the current episode's goal to extend the observation. We plot the test reward, computed on 100 episodes every 100 iterations:

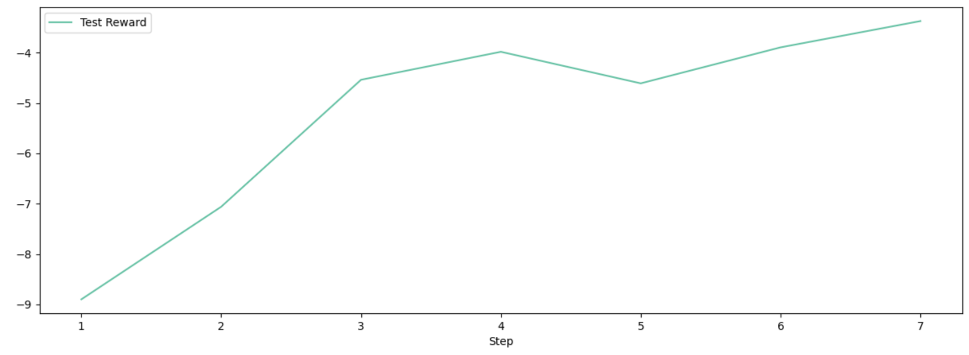
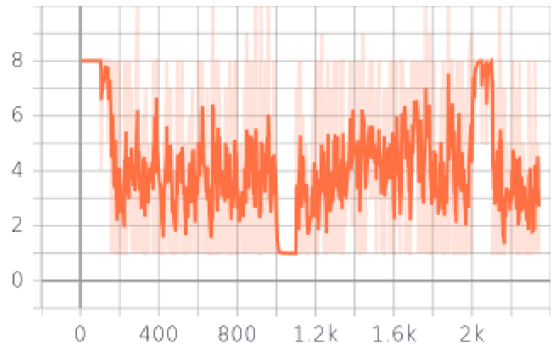
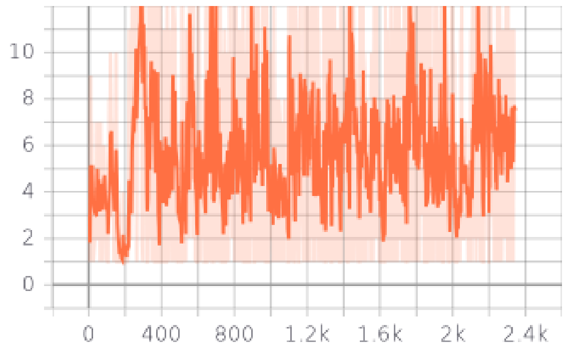


Figure 1: add image

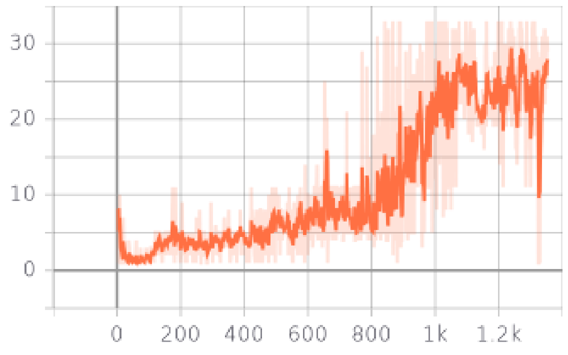
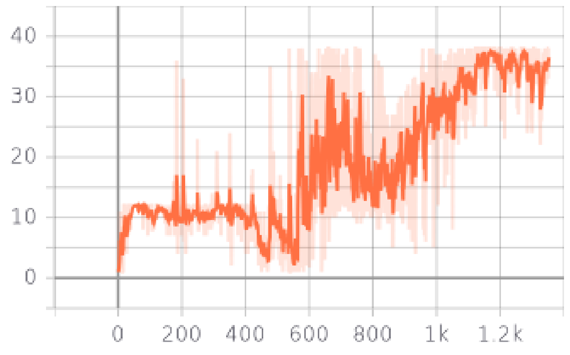
Our second environment `plan2` does not come with a set of predefined goals to sample from. With only one goal, relatively far away from its starting state, the agent never learns to navigate in the grid away from its initial state, as we can observe by looking at (x,y), the final coordinates of the agent at the end of each episode:

Figure Evolution of x	Evolution of y
	

Hinsight Experience Replay

We implement the Hinsight Experience Replay (HER), which fills the Replay Buffer of our agent with new transitions, with the last visited state at the end of episode as the transition's goal.The `HER` is implemented as a wrapper to the agent. At each call of the `act` function, it checks if the episode is done, creates the supplementary transitions and adds them in the agent's buffer.

As a result, our cumulative rewards starts to increase at the third testing session, as we observe the agent terminating more and more frequently its episode in the final state: (33,38):

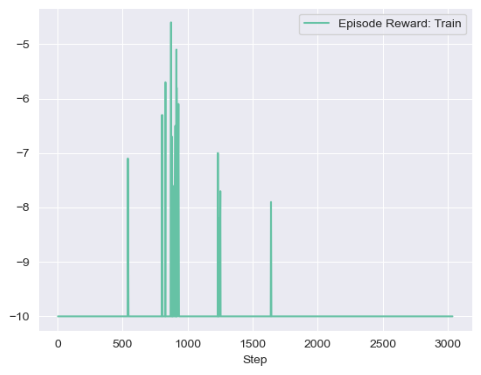
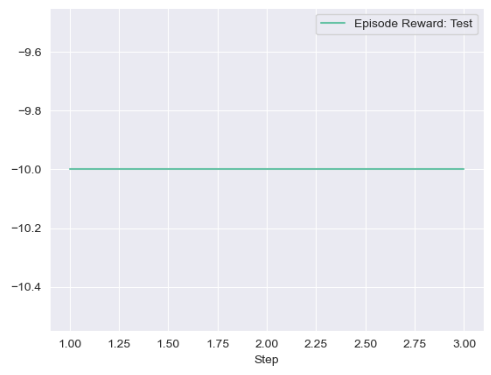
Figure Evolution of x	Evolution of y
	

Iterative Goal Sampling

We implement the Iterative Goal Sampling (IGS) as a wrapper for our Double DQN (see `curriculum.py`). To sample goals, we implement a new type of weighted FIFO Buffer. We maintain goal weights as the entropy of the successes over trials ratio, and compute the corresponding sampling probabilities with a softmax function after every new addition to the buffer.

Unfortunately, we were not able to sucessfully depart from the initial episode reward of *-10*. We assume a bug in our implementation.

On several occasions, our agent is able to reach the final state, as shown by our train reward. But it does not learn a policy leading to a consistent sucess:

Figure Evolution of train reward	Evolution of test reward
 Episode Reward: Train −5 −6 −7 −8 −9 −10 0 500 1000 1500 2000 2500 3000 Step	 Episode Reward: Test −9.6 −9.8 −10.0 −10.2 −10.4 1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00 Step