Report: Dealing with sparse rewards in the Mountain Car environment

Paolo Giaretta - Stefano Ferioli

CS-456 - Artificial neural networks/reinforcement learning - spring semester 2024

1 Random agent

An agent choosing uniformly at random among the three possible actions at every environment step cannot solve the problem within 200 steps. In Figure 1, all 100 test episodes end at 200 steps due to truncation. Moreover, the agent remains bounded in the valley as the maximal deviations from the starting position remain small (Figure 1).

This preliminary analysis shows that a successful agent must learn a coherent strategy that allows it to choose at each environment step a suitable action as a function of the current state (position and velocity).

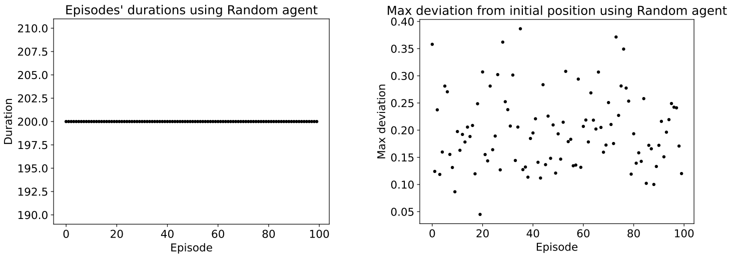


Figure 1: Random agent performance.

2 DQN agent

2.1 No auxiliary reward

The DQN agent trained without auxiliary reward is not capable of solving the task. All episodes are truncated at 200 steps and the cumulative reward remains -200 for all episodes (Figure 2). This behavior is expected given the sparsity of reward. Indeed, the Q-network is randomly initialized and the agent cannot solve the task. This means that at the beginning the agent observes only transitions with a reward of -1. Very quickly the Q-network minimizes the Q-learning loss (Equation 1) by approximating all Q-values as $Q(s)_a \approx -1/(1-\gamma)$ (Figure 2). Once this happens, the agent either moves in a preferred direction or almost randomly, thus failing to move out of the valley and experience rewards different than -1.

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s') \in \mathcal{B}} \left(r + \gamma \max_{a'} SG(Q(s')_{a'}) - Q_\theta(s)_a \right)^2 \tag{1}$$

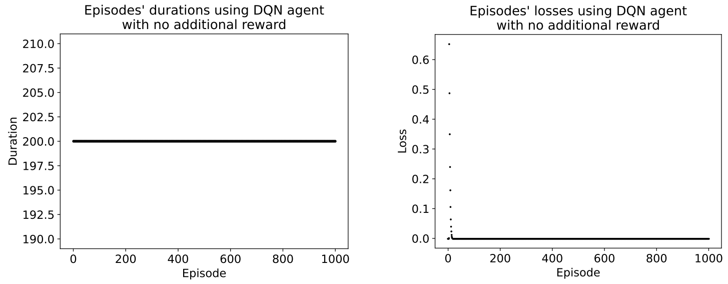


Figure 2: Performance of DQN with no auxiliary reward.

2.2 DQN with heuristic reward

To assist the agent in learning a better behavior we introduce an auxiliary heuristic reward based on the variation of total mechanical energy. We add the heuristic reward to the environment reward. This way, we encourage the agent to increase the cart mechanical energy, allowing it to explore states of higher elevation. The mechanical energy of the car E is estimated as a function of the state variables $s = (x, v)$ as $E(x, v) = \frac{1}{2}v^2 + \frac{sin(3x)}{800}$. When the agent observes a transition (s, a, r, s'), the auxiliary reward is proportional to the increment in mechanical energy $r^{\text{aux}} = \lambda(E(s') - E(s))$ between the old and the new state. The auxiliary reward weight factor λ must be chosen so that the auxiliary reward is neither too large or small:

- If the auxiliary reward is excessively small, the agent does not learn to solve the task. Its behavior becomes similar to DQN with no auxiliary reward, as the learned the Q-values of neighboring states become very similar, even if the mechanical energy increases. For example, this is the case when $\lambda = 1500$ (Figure 3).
- If the auxiliary reward is excessively large there are two problems. First, the training become unstable and could not converge. Second, if the total reward becomes positive for some transition, the agent may still learn solve the task, but with more steps than necessary on average. Indeed, if the total reward becomes positive for some transitions, the agent will learn to extend the episode in order to be rewarded more than for concluding the episode. For example, this is the case when $\lambda = 150000$ (Figure 3).

Unless specified otherwise, in all subsequent plots we use $\lambda = 15000$, as it obtains very good results.

With the addition of this auxiliary reward, the agent learns very quickly. From the cumulative success graph, we can infer that once

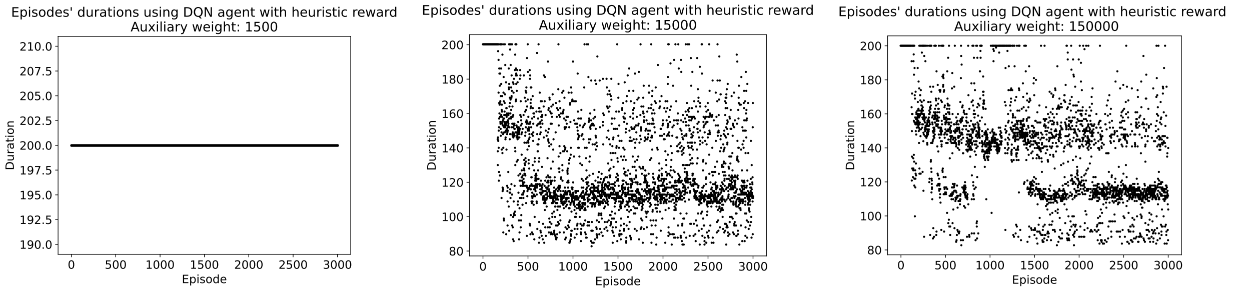


Figure 3: Effect on auxiliary reward weight on training ($\lambda = 1500,\ 15000,\ 150000.$)

the agent solves the task for the first times it remains successful, with very few exceptions. For all later episodes, the graph becomes linear with a unitary slope.

The plot of the step-by-step update losses in Figure 4 shows an initial low loss, followed by a sharp increase when the agent solves the task for the first times, and a final decrease with smaller fluctuations. At the beginning the agent remains closely bound to the bottom of the valley, thus the Q-network learns very well the approximate Q-values for these frequently visited states. When the agent reaches higher elevations, it visits these states for the first time. The Q-network has never seen these states, and the estimated Q-values present high TD error in Equation 1. After numerous successful episodes, the agent has already visited those transitions multiple times, the Q-values become more consistent and the training loss decreases over time.

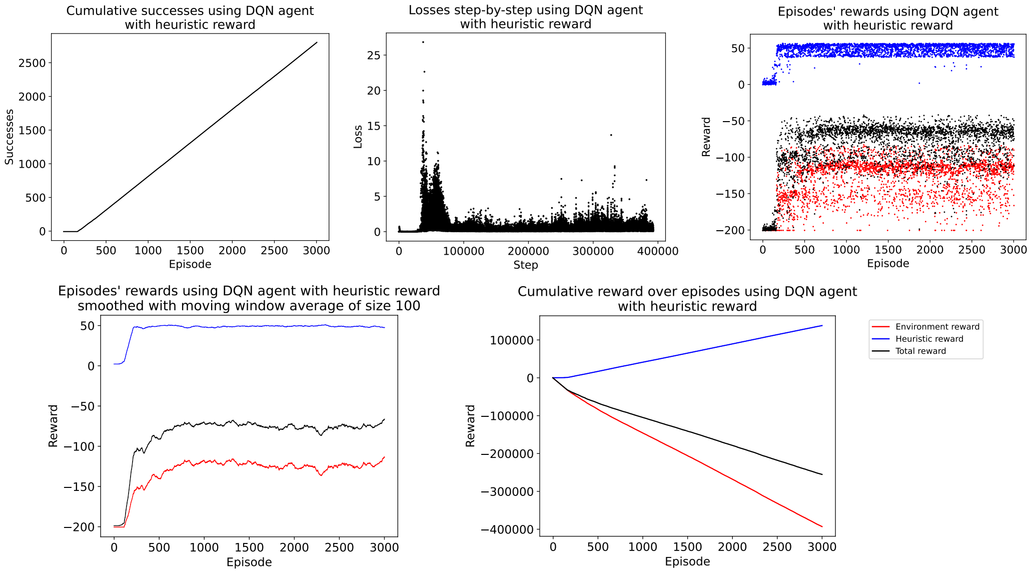


Figure 4: Performance of DQN agent with mechanical energy increment as an auxiliary reward ($\lambda = 15000$).

2.3 Non domain-specific reward

The Random Network Distillation (RND) [1] encourages the agent to explore new parts of the environment by computing the RND intrinsic reward. To do so, it is necessary to normalize the transition to the next state s' before evaluating the target and predictor networks and to normalize the squared difference of target and predictor outputs. These normalizations are necessary to ensure consistency of the reward scale for different inputs and time points. Without normalization, the frozen target network may provide excessively small feedback to the agent or not be sensitive enough to input variations. By subtracting the mean and dividing by the standard deviation, the agent receives an intrinsic reward that depends on how much the outputs of the networks differ relatively from previously received intrinsic rewards. Similarly, by normalizing the inputs the predictor network is trained based on the relative distance of the current state from the previously seen ones. For these reasons, Burda et al. emphasize that it is challenging to find hyperparameters that work in all settings without normalization [1].
The reward factor parameter must be chosen in a sensible range for successful training (Figure 5).

- If the reward factor is excessively small, then the agent is not sufficiently encouraged to explore new states and behaves similarly to standard DQN with no heuristic reward.
- If the reward factor is excessively large, two problems may arise. First, the training may become unstable and not converge (exploding loss). Second, the agent may solve the environment from time to time but it will prioritize other transitions over terminal ones because the RND reward outweighs the environment one. We observed that this can even lead the agent to forget a successful strategy over time.

Given that the RND reward is clipped in the range $[-5,\ 5]$, the reward factor must live in $[0,\ 0.2]$, so that the the overall reward per step is always negative, except for terminal transitions. We choose a reward factor of 0.2.

The heuristic reward and RND reward present different behaviors. When the agent learns with the heuristic reward, the auxiliary reward stabilizes to positive values (Figure 4) as the agent favors transitions increasing the total energy. In the RND case, the additional reward presents a positive spike when the agent starts to explore new states (Figure 6). However, as the agent remains successful for many episodes those states have been explored many times and the RND reward diminishes to negative values. As expected from

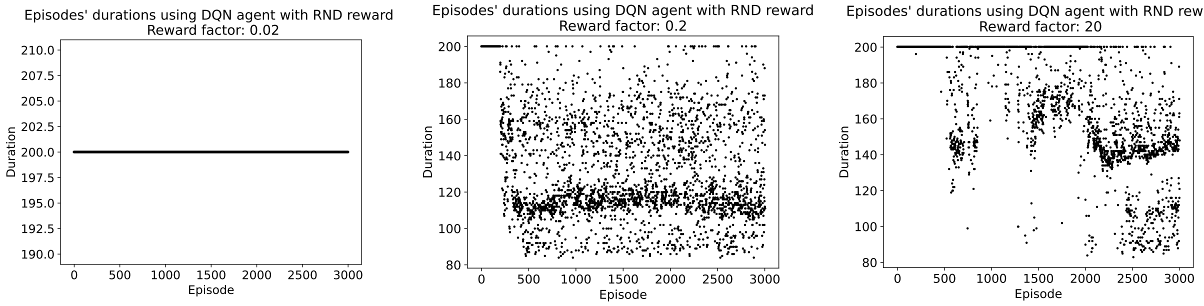


Figure 5: Effect of reward factor on training $(0.02,\, 0.2,\, 5)$.

normalization, the RND reward remains on average closer to zero. Given the random initialization of the frozen target network, the RND reward spans a wider range of value than the ad-hoc heuristic reward. The other plots show similar behavior to DQN with heuristic reward, except for the auxiliary reward that, unlike the heuristic reward, is negative on average.

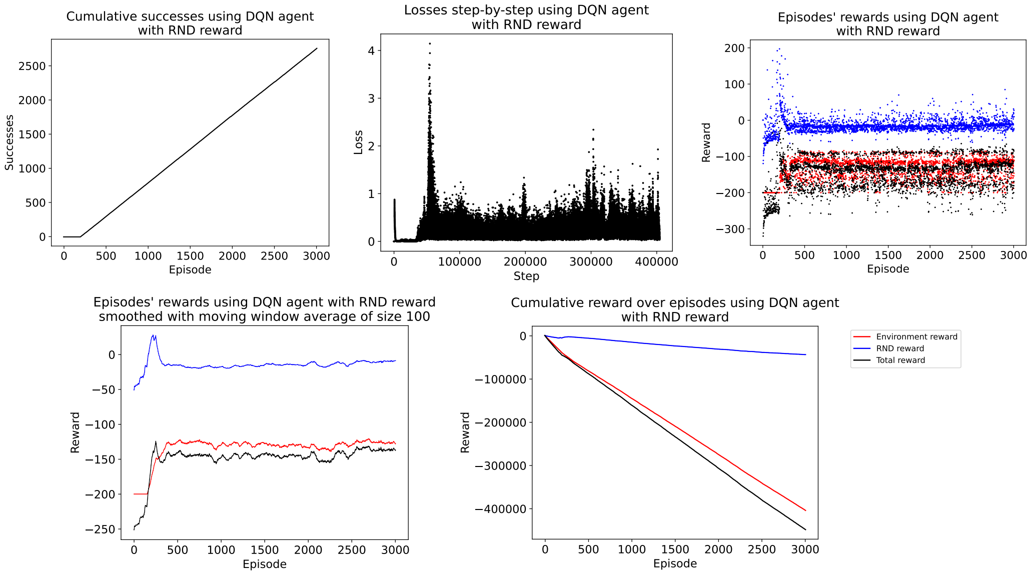


Figure 6: Performace of DQN agent with RND reward for a reward factor of 0.2.

An alternative to the RND reward is to train a predictor that tries to predict the next state s' from the current state-action pair (s, a). For stochastic environments, the target function (the next state) is not deterministic. If the agent is rewarded for incorrect forward dynamics predictions (when it explores new states), then there is the risk it remains attracted to local sources of entropy in the environment [1], i.e. regions where the stochasticity is highest. RND solves this problem by employing a deterministic (even though randomly initialized) frozen network. The target is not stochastic and prediction errors are mostly due to the exploration of new states.

3 Dyna agent

We start by saying that the agent starts solving the task after very few training episodes.

This result strictly depends on the bucket size.

- If the buckets are too big (and too few) the policy that is learned by the agent is of poor quality and does not always solve the problem. Indeed, if the buckets are too big, the action taken in a given bucket is not guaranteed to influence in which bucket the cart will be in the next step, and this makes it very difficult for the agent to estimate what is the effect of every action on the next state. In Figure 7, we see how the number of failures is very high even after 3000 training episodes.
- If the buckets are too small (and too many) the policy that is learned by the agent is of good quality. However, having too many bickets makes the propagation of information among buckets (and thus the learning) very slow. In addition, the movement of the cart is likely to "jump over" some buckets, making it difficult for the agent to keep track of the "continuity" of the states. In addition, too many states make the training very slow, as the data structures used by the agent grow quadratically in size. In Figure 7, we see how the number of failures decays very slowly over the episodes, having failures even after 2000 training episodes.

As for the DQN agent, in Figure 8, we plot the episodes' durations, the episodes' total reward, the cumulative episodes' rewards, and the cumulative number of successes. Instead of the step-by-step loss, we plot the step-by-step Q-values update steps.

As expected, the episode duration decreases rather quickly as the Dyna agent is trained. Since now there is no additional reward, the reward plot is the same as the duration plot with changed signs. After some initial failures, the number of successes increases steadily. On the other hand, the episodes' cumulative reward decreases steadily (necessarily, since all episodes' rewards are negative), however, we are glad to see that the curve is convex, meaning that the episodes' total rewards are overall increasing and the agent is thus learning.

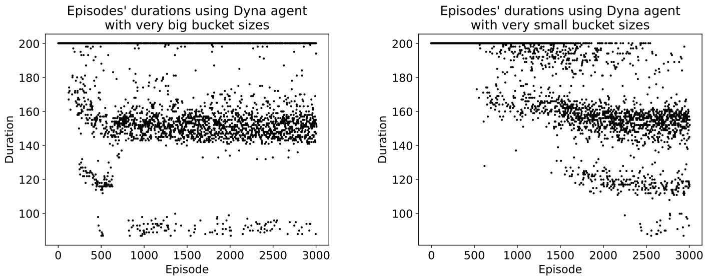
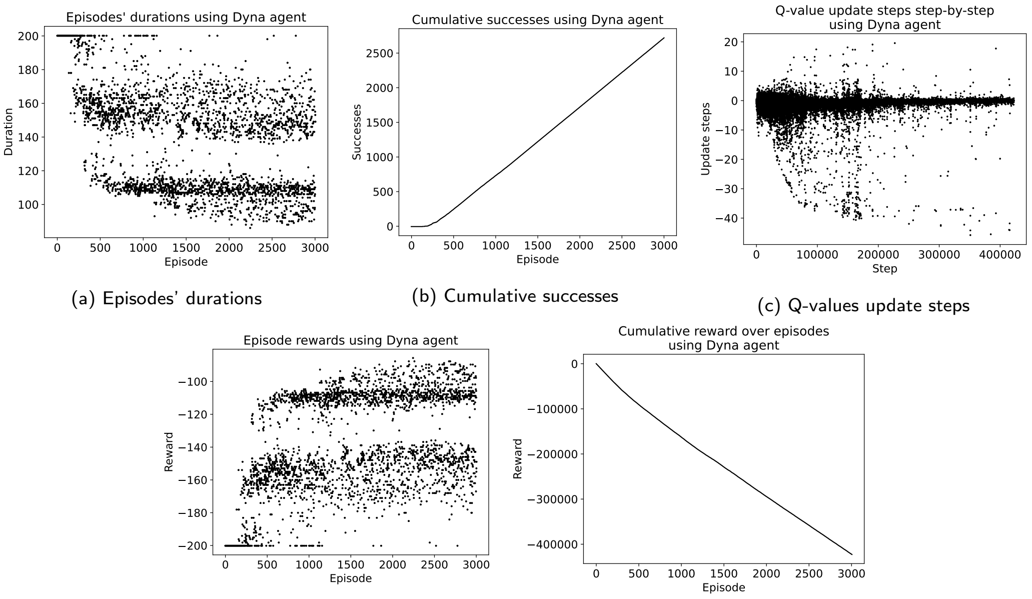


Figure 7: Dyna agent's training performance, respectively, when quadrupling and dividing by 4 the sizes of the buckets



(a) Episodes' durations

(b) Cumulative successes

(c) Q-values update steps

(d) Episodes' environment reward

(e) Episodes' cumulative reward

Figure 8: Dyna agent training statistics

Differently from DQN, the Dyna agent manages to solve the task without the need for additional rewards. This is due to how the Q-values are initialized. Indeed, initializing the Q-values at 0 makes the agent "expect" a reward of 0 when visiting a state that has never been visited before. This is an incentive to exploration that allows the agent to eventually solve the task successfully.

We examine now the distribution pattern in bands that we see in the duration plot, especially for large numbers of episodes. We hypothesize that the bands are determined by the initial position of the cart. To test this hypothesis, we plot again the episodes' durations using a colormap to assign a color to each dot depending on the initial position. In Figure 9, we can see that our hypothesis is indeed founded.

Now we create new plots illustrating the estimation of the highest Q-value in each bucket after training (Figure 10). We notice how the Q-values develop around the central states, which are also the initial states. From the progression of heatmaps in Appendix A, we can see how the information is propagated from the center outwards. The agent quickly learns the expected (negative) reward of all the states that are visited. The Q-values stabilize rather quickly to the final result. Finally, we plot a few episode trajectories (Figure 11). We selected six episodes based on their initial position. These plots easily relate to what we have already observed about the episode's duration. Indeed, for initial positions of -0.52 or less, we have a rather consistent duration and similar trajectories (red and orange dots in Figure 9). For initial positions between -0.48 and -0.44, we have longer durations (yellow and light green dots in Figure 9). Finally, if the initial position is close to -0.4, we suddenly have a very good performance in terms of duration (dark green dots in Figure 9).

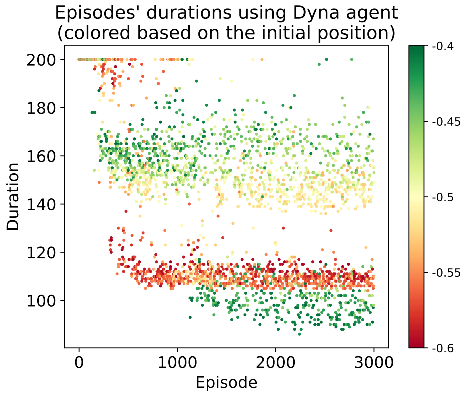


Figure 9: Dyna training episodes' durations colored based on the initial position of each episode

4 comparison

We conclude our report with a comparison of the three agents, DQN with heuristic reward, DQN with RND reward, and Dyna.

We start by comparing the training performance. In Figure 12, we plotted the episodes' environment rewards (smoothed with moving window average) and the cumulative episodes' environment rewards. We observe how DQN with heuristic reward is the first to reach peak performance, on the other hand, DQN with RND reward improves fast at the beginning but stabilizes to a worse performance than DQN with heuristic reward. Finally, Dyna improves rather slowly at the beginning, but in less than 1500 episodes reaches a good level comparable with DQN with heuristic reward. The second plot confirms our previous observations, as Dyna's cumulative reward is the worst among the three because of the higher negative reward accumulated in the first 1500 episodes, and DQN with heuristic

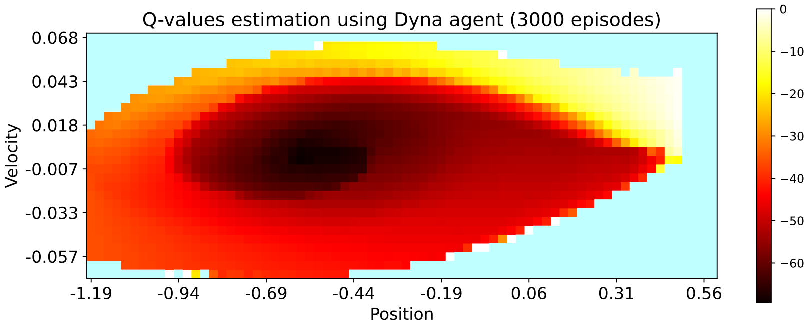


Figure 10: Dyna Q-value estimation after training. In light blue, the states that were not encountered.

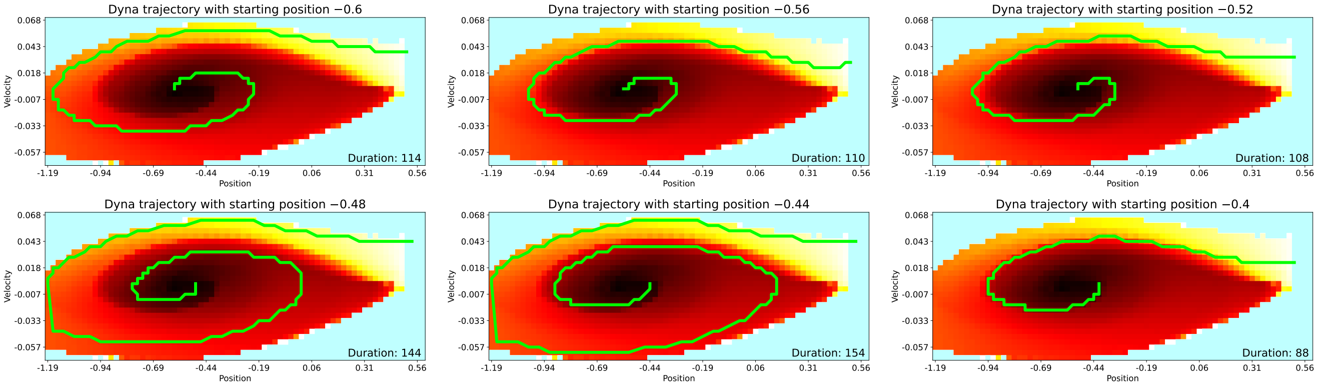


Figure 11: A few training episodes' trajectories for different initial positions

and RND rewards are respectively the best and the second-best as expected.

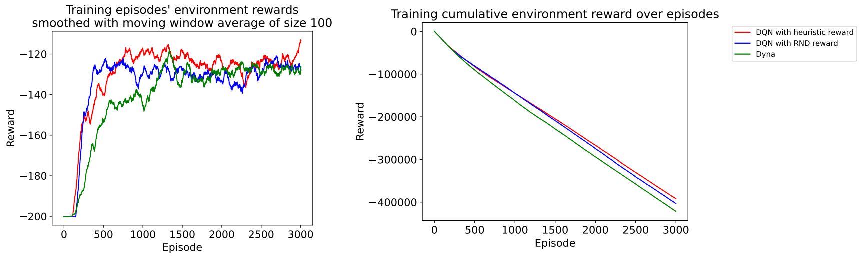


Figure 12: Episodes' environment rewards and cumulative episodes' environment rewards, both during training

We then compare the testing performance of the three agents. To do so, we set the ε parameter to 0, to make the three agents fully greedy. In addition, we do not call the update function after each step. We test the three agents on the same set of environment random seeds, and thus on the same set of initial states. In Figure 13, we plot the episodes' durations (smoothed with moving window average), the cumulative episode durations, and the episodes' durations as a function of the initial states. From the first two plots, we see how DQN with heuristic reward is on average the best-trained agent, DQN with RND reward is the second best and Dyna is the worst. However, if we compare the durations as a function of the initial state, we notice how DQN with heuristic reward is the worst for initial states less than -0.52 and that DQN with RND reward is "bad" only for a small set of initial states. Finally, we observe that Dyna does not always learn the optimal policy. For example, trajectories such as the 4-th and 5-th ones in Figure 11 are sub-optimal.

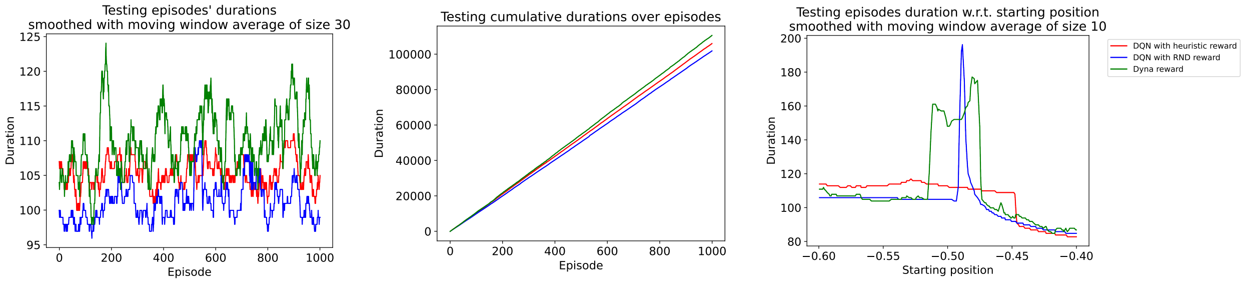
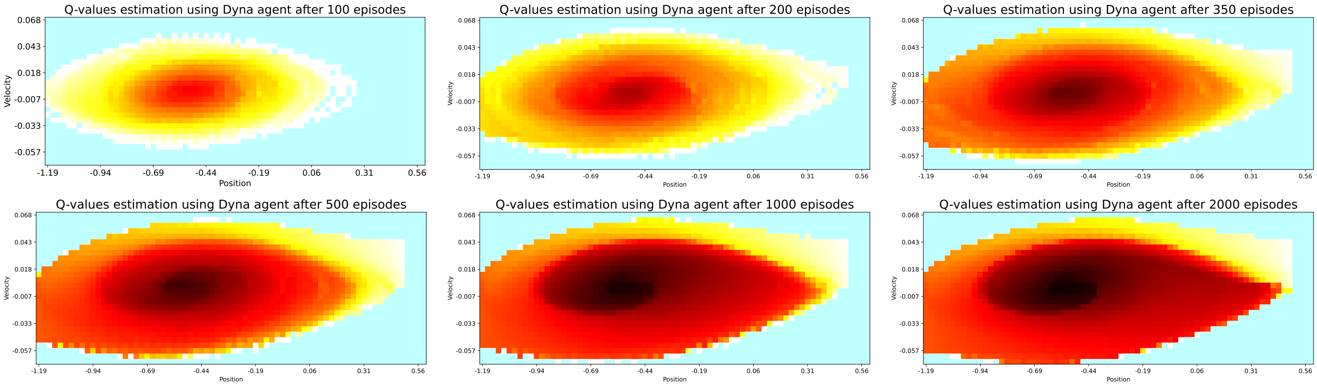


Figure 13: Episodes' durations, cumulative episode durations, and episodes' durations as a function of the initial states, all three during testing

Appendix

A Dyna Q-value estimation during training



Q-values estimation using Dyna agent after 100 episodes

Q-values estimation using Dyna agent after 200 episodes

Q-values estimation using Dyna agent after 350 episodes

Q-values estimation using Dyna agent after 500 episodes

Q-values estimation using Dyna agent after 1000 episodes

Q-values estimation using Dyna agent after 2000 episodes

Figure 14: Dyna Q-value estimation during training. In light blue, the states that were not encountered.

References

[1] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, *Exploration by random network distillation*, 2018.