DPRL Assignment 4

Rohan Zonneveld & Joost Driessen

December 2023

1 Tabular Q-learning

The ϵ-greedy Q-learning algorithm works by applying Bellman iterations based on observations. Q-values are initialised at zero and updated with every observation using Equation 1.

$$Q(x_t, a_t) = Q(x_t, a_t) + \alpha_t \left[r_t + \gamma \max_{a' \in A} Q_t(x_{t+1}, a') - Q_t(x_t, a_t) \right] \tag{1}$$

Q-values The Q-values for the 3x3-grid are depicted in Table 1. The Q-values are the same for tabular Q-learning (dynamic programming case) and the ϵ-greedy.

Hyperparameters Since Q^* is the only solution that satisfies the Bellman equation it is guaranteed that the optimal solution will eventually be found if three conditions are met. The learning rate must satisfy $\sum_t \alpha_t = \infty$, because when the Q-values have not converged to Q^* it must be possible to update them. Secondly, $\sum_t \alpha_t^2 < \infty$, because it guarantees that the learning rate diminishes appropriately (to a finite number), preventing excessive updates that might hinder convergence. Finally, the exploration policy must be such that there is some probability an action is taken in state x for all actions and all states. This makes sure every action is taken and the Q-value, corresponding to this action, can be updated. Moreover, ϵ must be sufficiently high, since we only update the Q-values along the trajectory that is taken. If ϵ is too low, a policy leading to the reward will be followed and therefore some other Q-values will not be updated. Since the Q-values will not get updated they look converged while in fact they are not updated.

2 Deep Q Network

To deal with large or continuous state spaces we can use function approximators to represent the Q-values and subsequently learn the correct function. In this

assignment, we used a neural network, with two input states (the x- and y-coordinates), two hidden layers with 24 neurons each and four output neurons (one for each possible action) to model the Q-value function. The algorithm works by using two models in parallel, a learning model and a target model. Both models are initialised with the same weights. After each step in the environment the state, action, reward, next state tuple is added to a replay buffer and a mini batch is sampled from this buffer. This mini batch is then used to train the learning model. The targets are the predictions by the target model, which weights are kept the same between multiple episodes, plus the direct reward, which is added to the Q-value corresponding to the action taken to get this reward. Every C iterations, the target model's weights are set to the current weights of the learning model.

Q-values In the Appendix, Figure 1 is depicted of the Q-values learned by the DQN. The Q-values to the left of the target state (5,5) are highest for the action right, the same pattern is observed for states lower than the target states where the Q-values for 'up' are highest. Furthermore, the Q-values seem to get smaller with a factor of 0.9 every step away from the goal state. This aligns with $\gamma = 0.9$, indicating that the Q-values were successfully approximated. This can also be seen in Figure 2, where the optimal policy is depicted color coded with the V-value function. The value is higher for states closer to the end state and the policy points to higher states.

Hyperparameters Two hidden layers with 24 neurons each were chosen for the neural network architecture as this a standard form. While testing the algorithm the Q-values were learned pretty fast. This neither indicates over- or underfitting of the model, thus the complexity of the model fits the problem at hand. The ϵ, the exploration parameter in ϵ-greedy used to choose the next action, was set to 1. An additional parameter, ϵ-decay $= 0.995$, gradually loweres the exploration to allow for more exploitation while the model approximated the optimal policy more closely. The batch size was set to 8, as a smaller batch size allows for more exploration compared to a larger batch size due to every update being more noisy. The update frequency of the target model was set to five, which means the weights of the target model get replaced by the weights of the learning model every five episodes.

Convergence To assess the convergence of the algorithm, the obvious choice is to check if the Q-values are stable. In this case the learned Q-values match the actual Q-values when the reward is effectively predicted for each state transition. After every update of the target model, the Q-values are compared to the ones from the previous target model. The difference between these vectors can be quantified by subtracting one from the other and calculating the length of that vector. If the length of that vector is smaller than 0.01 we define the Q-values to be converged to the optimal solution. The convergence is visualised in Figure 3.

Appendix

State (x)	Up	Down	Left	Right
1,1	0.73	0.66	0.66	0.73
2,1	0.81	0.66	0.73	0.81
3,1	0.81	0.73	0.81	0.90
1,2	0.81	0.73	0.66	0.81
2,2	0.90	0.73	0.73	0.90
3,2	0.90	0.81	0.81	1.00
1,3	0.90	0.73	0.73	0.81
2,3	1.00	0.81	0.81	0.90
3,3	0.00	0.00	0.00	0.00

Table 1: Values for Q^*

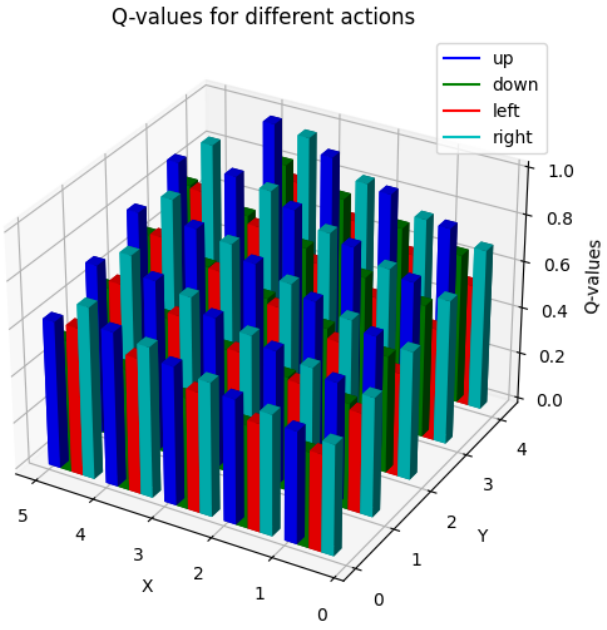

Figure 1: Q-values for every state in the mini maze environment

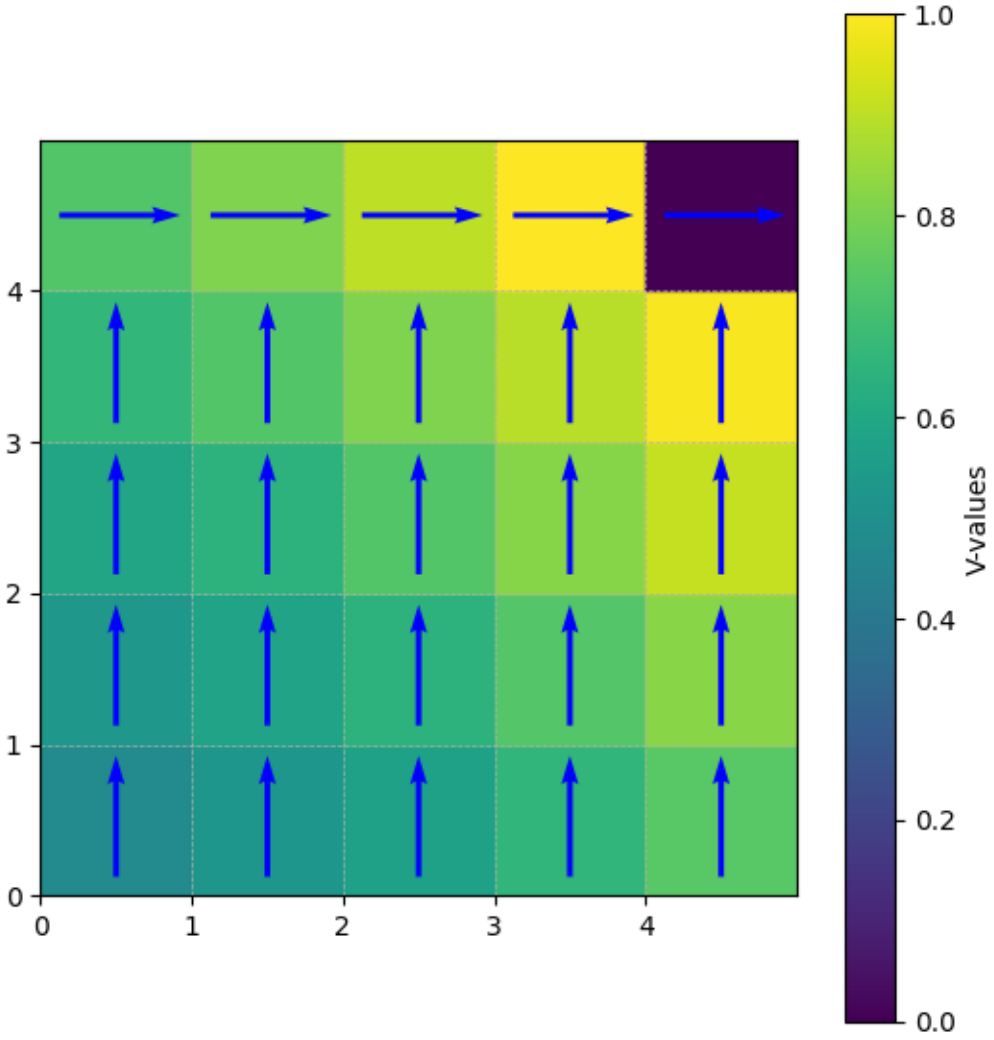

Figure 2: Optimal policy color coded with the value function

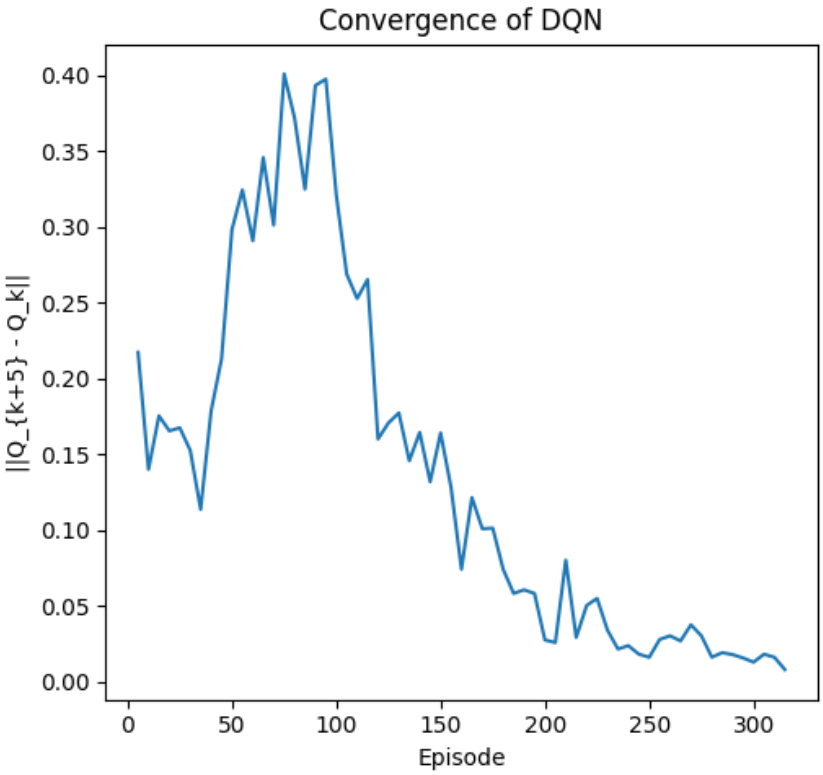

Figure 3: Convergence of the DQN