Reinforcement Learning Coursework 2 DQN

Department of Computing, MSc Advanced Computing

Belfiore Asia, *CID*: 02129867

Q1) DQN Tuning

1.1) Hyperparameter Tuning

The final hyperparameters of the model are listed below in Table 1. All these were chosen as their combination gave the best-performing model in terms of learning stability, velocity, and total return out of all the other tested values.

Label	Hyperparameter	Value
A	Size of (each) hidden layer in DQN	5
B	Number of DQN hidden layers	2
C	DQN Learning Rate*	0.01 (used adaptive learning rate: see Section 1.1.*)
D	Size of Replay buffer	30000
E	Number of Training Episodes	300
F	ϵ value for ϵ-greedy policy*	0.2 (used ϵ-decay instead: see DECAY_RATE and Section 1.1.*)
G	Reward Discount	1
H	Batch Size of sampled trace (from replay buffer)	100
I	Frequency of Target network updates	50
DECAY_RATE	Decay Rate for ϵ-greedy policy	0.95
"Adam"	DQN Optimiser	-

Table 1: Optimized DQN Hyperparameters. Additional parameters are highlighted in grey.

1.1.) Exploration-Exploitation Strategy*

I implemented ϵ-decay for the ϵ-greedy policy with decay rate of 0.95, which gave the best model performance out of the values tested (0.995, 0.98, 0.95, 0.9). This allows for more exploration at the beginning of training while prioritising exploitation as the number of episodes increases.
Without epsilon decay, the same model (with same hyperparameters as in Table 1), had a lower average return and more unstable slower leanin, with return values dipping below the threshold throughout the episodes.
Furthermore, I set the Adam optimizer (instead of Stochastic Gradient Descend) as model optimiser, which led to a much better model performance due to the introduced adaptive learning rate (i.e. non-constant value stated in Table 1) which varies during the training episodes. This change alone had by

far the most significant impact on the learning velocity and quality of the agent, and allowed even the unoptimised starting agent to reach decent average training returns.

1.2) Learning Curve

The final DQN, with the hyperparameters listed in Table 1, has a mean learning rate and variance shown below in Figure 1. The agent consistently achieves a mean return that surpasses the target threshold for over 200 episodes. However, the sequential dips and spikes in the mean returns throughout the episodes and the high return variance throughout the episodes signal that the agent's learning is unstable.

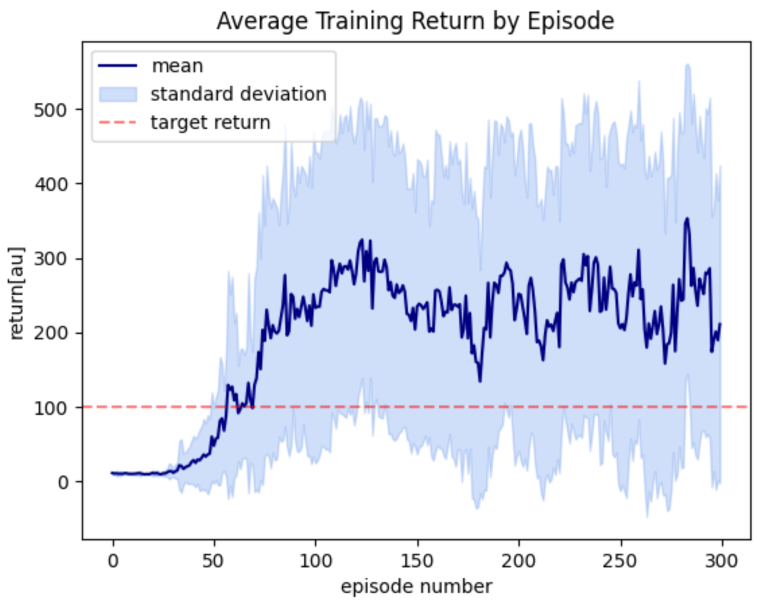


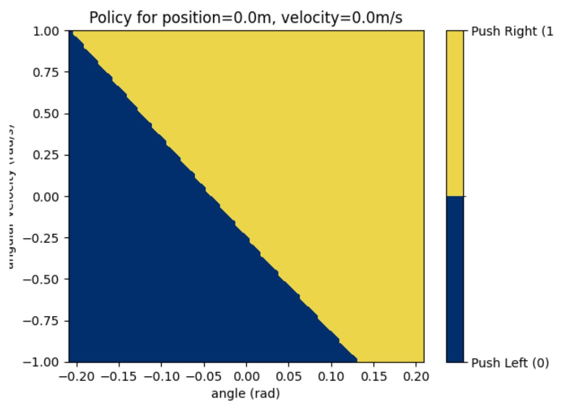
Figure 1: Learning of the DQN agent with hyperparameters shown in Table 1 for 10 Training runs of 300 episodes each.

Q2) Policy Visualization

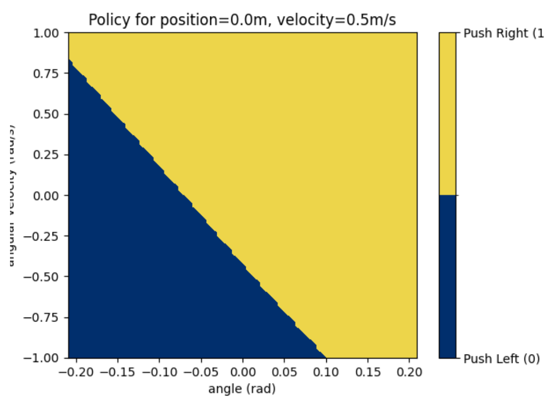
2.1) Slices of Greedy Policy Action

The DQN's learnt policy is shown below in Figure 2 as a function of pole angle and angular velocity. In general, for an optimal agent the graphed area of the (optimal) policy would be split into two sections by a negative sloped line, representing the action decision border, passing through the centre of the angle-angular velocity space ((0,0) for cart velocity = 0). Everything above the line would represent the cases where the agent chooses to push the cart right (action 1, yellow region). Everything below represents the cases where the agent pushes left (action 0, blue area). The slope of the border depends on the dynamics of the environment (see points below). In particular:

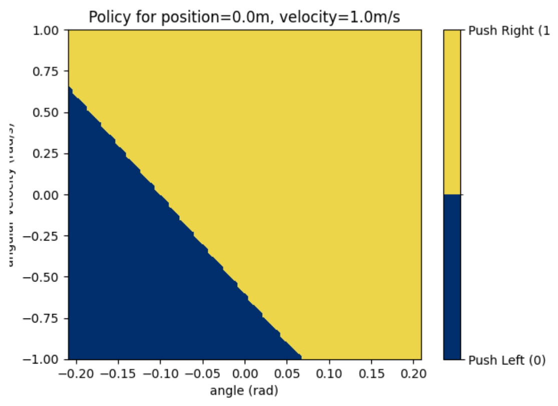
- *The regions of the plot where the agent chooses to push left or right:*
  a) *Optimal Agent* → In any scenario in which the pole is tilted left (negative angle) or has negative angular velocity (left rotation), an optimal agent will push left (blue region); similarly, anytime the pole is tilted right (positive angle) or has a positive angular velocity (right rotation), the agent will push right (yellow region). This makes intuitive sense, as the agent tries to balance the pole whenever it is tilting and rotating in the same direction, by pushing in the same leaning direction to counterbalance the angular velocity and bring the pole back up. The greater the angular velocity (i.e. the faster the pole rotates), the more the agent will have to push.
  b) *Implemented DQN* → The policy learnt by the agent generally matches the optimal behaviour.



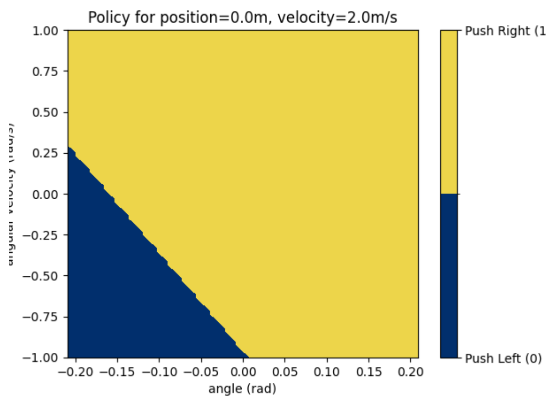
a) DQN Policy with cart position = 0m and velocity = 0m/s



b) DQN Policy with cart position = 0m and velocity = 0.5m/s



c) DQN Policy with cart position = 0m and velocity = 1m/s



d) DQN Policy with cart position = 0m and velocity = 2m/s

Figure 2: Learnt policy by DQN agent with hyperparameters shown in Table 1. The blue zone represent the cases where the agent pushes left, the yellow zones where it pushes right.

- *The general shape of the action decision boundary:*
  a) *Optimal Agent* → As stated above, the boundary between the two action decisions would be a sloped diagonal line starting from the top-left and ending at the bottom-right of the graphed region, and centred somewhere in the graph depending on the cart velocity (see points below). This follows the behaviour described in the previous point, as the decision border can be intuitively interpreted as the cases when the action chosen by the agent is 'invariant' from the angle and angular velocity because the angular velocity alone is *somewhat* sufficient to keep the pole stable. The slope of the border is defined by which angular velocity values counterbalance each tilting angle and depend on the dynamics of the environment: the further down the pole is tilting (higher $|\theta|$), the greater the angular velocity needs to be to balance it out. This means that the boundary does not necessarily start (and end) at the top-left and bottom-right corners of the graph, but can be less or more sloped based on these balances.
  b) *Implemented DQN* →The policy learnt by the DQN generally follows this behaviour, however, the slope of the action border is more sloped (compared to the main diagonal).
- *The symmetries of the action decision boundary when the cart velocity is 0 (Figure 2a):*
  a) *Optimal Agent* → If the cart velocity is 0, the boundary of the action decision would be centred at the centre of the graph (i.e. pass through (angle, angular velocity) = (0, 0)). This reflects that the action decision is 'at the border' when the pole is upright and not moving (0,0) and

when it's tilting to one direction with (balancing) opposite angular velocity. Thus the two action areas are symmetric with respect to the action border.

b) *Implemented DQN* → The policy learnt by the DQN follows this behaviour, however, the action border is not centred at (0,0), which means that the agent tends to push the cart right more often, even when the pole is balanced (optimal policy border cases).

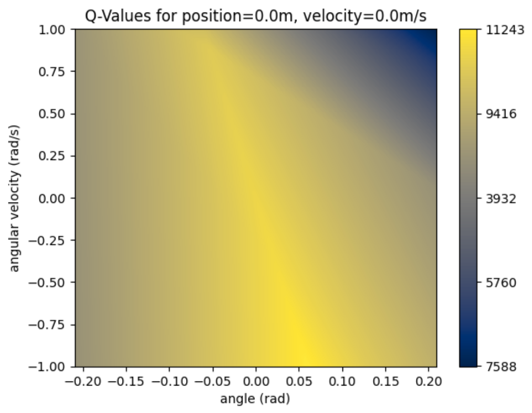
- *How the action decision boundary shifts as cart velocity increases (Figure 2b, 2c, 2d):*

a) *Optimal Agent* → With higher cart velocities, the boundary of the action decision shifts towards the bottom-left of the graph: this reflects the fact as the velocity of the cart increases (moving faster to the right), the balance states observed with the stationary cart do not hold anymore (intuitively, at (0,0) the cart will be naturally moving to the right which will in turn make the pole tilt to the left with a negative angular velocity, which stronger as the cart velocity increases); these balance points are now shifted to lower values of both angle and angular velocity, causing the agent to move the cart right in order to maintain inertia (assumes that there is non acceleration and the cart velocity is constant).

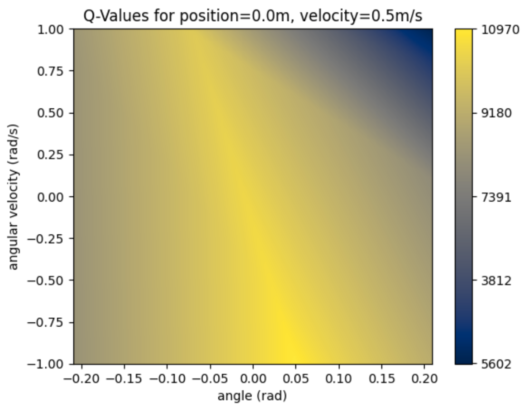
b) *Implemented DQN* → The policy learnt by the DQN generally follows this behaviour.

2.2) Slices of Q-Function

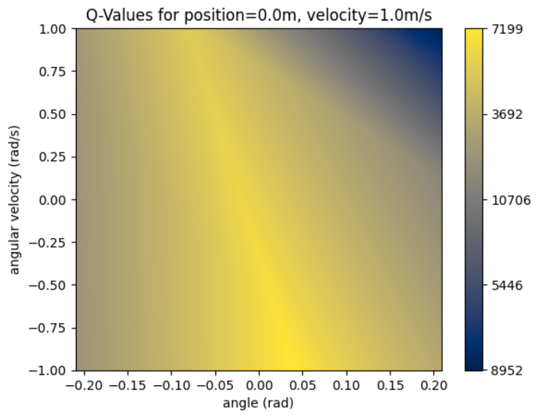
The DQN's Q-Function is shown, as a function of pole angle (x-axis) and pole angular velocity (y-axis), below in Figure 3.



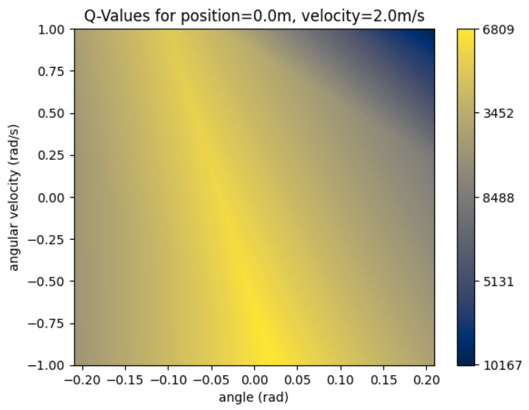
a) DQN Q-Function with cart position = 0m and velocity = 0m/s



b) DQN Q-Function with cart position = 0m and velocity = 0.5m/s



c) DQN Q-Function with cart position = 0m and velocity = 1m/s



d) DQN Q-Function with cart position = 0m and velocity = 2m/s

Figure 3: Cumulative Q-Values learnt by DQN agent with hyperparameters shown in Table 1 for 300 episodes. The yellow area shows the highest Q-Values, the blue area shows the lowest Q-values.

The Q-Values should be high when the pole is (or tends to be) upright and stable, i.e. when either both the angle and angular velocity of the pole are zero or when they're (balanced) opposite of each other, meaning either the angle is positive and the angular velocity is negative (pole is tilted towards the right but it is being pushed back towards the centre) or vice versa. If, otherwise, the angle and angular velocity have the same sign, then the pole is likely to fall (i.e. tilted and falling towards the same direction), and the Q-values should be low. In general, the Q-Values should follow the same symmetry described in Q2.1 for the policy. In particular:

- *The regions of the plot where values are relatively higher or lower:*
  a) *Optimal Agent* → Following the reasoning above, the highest values (yellow region) should be concentrated along the policy action decision border (Figure 2). These should (somewhat steadily) decrease as they approach the top-right (positive angular velocity and angle) and bottom-left (negative angular velocity and angle) corners of the graph, where the lowest Q-values (blue region) appear: these areas represent the cases where the pole is most unstable, i.e. tilting towards one direction with same-sign angular velocity (i.e. positive angle and positive angular velocity, and negative angle and negative angular velocity) and where the agent is more likely to make the pole fall. An optimal agent is able to always correct the tilting pole for any angle or angular velocity.
  b) *Implemented DQN* → The Q-Function learnt by the implemented agent follows this. However, the lowest Q-values only appear where both the angle and angular velocity are positive (top-right corner), and generally, the values don't decrease uniformly (they take mostly similar middle values).
- *The range of values the agent has learned, both close and far from the edge of the episode termination region:*
  a) *Optimal Agent* → An optimal agent would always have a return of 500 per episode because it is able to balance the pole for as many steps as allowed, independently from the starting position of the cart (since it is technically able to keep the pole upright forever or for the maximum allowed number of steps, here equal to 500). It follows that the values will be equally distributed at each point of the region (even at the edges of $\pm 2.4\ m$), from the fact that the optimal policy can 'go on forever', thus avoiding falling off the allowed position ranges. In terms of angle and angular velocity, the Q-values steadily decrease as they both get positively higher or negatively lower ((-0.2 rad, -1 rad/s) and (0.2 rad, 1 rad/s), top right and bottom-left corner).
  b) *Implemented DQN* → The maximum value of the learnt Q-Function (in Figure 3 for all training episodes) is lower than the optimal one.

With any cart velocity, the Q-values are symmetrical with respect to the policy action decision border for that velocity (Figure 2).

- *The symmetries of the learned values when the cart velocity is 0 (Figure 3a):*
  a) *Optimal Agent* → With velocity = 0, the Q-values are maximal along the policy's action decision border and slowly decrease (symmetrically with respect to this border) as they approach higher positive (top-right) or negative (bottom-left) angles and angular velocities.
  b) *Implemented DQN* → The Q-Function learnt by the implemented agent follows this.
- *How the values change as cart velocity increases (Figure 3b, 3c, 3d):*
  a) *Optimal Agent* → As the velocity increases, the Q values generally take lower values when compared to a stationary cart of velocity zero their distribution shifts similarly to the policy action boundary. This is because the pole is naturally more unstable due to the resting velocity of the cart, and it is harder for the agent to balance it. Furthermore, it becomes much easier for the cart to fall off the edge of the region when both the angle and angular velocity take high positive values (tilting and rotating right, top-right corner), as any right push the agent makes to balance the pole further increases the cart velocity thus making it reach the right edge faster.
  b) *Implemented DQN* → The Q-Function learnt by the implemented agent follows this.