Reinforcement Learning

Deep Q Networks

Coursework 2

18/11/2022 - 02/12/2022

MSc Computing in Artificial Intelligence & Machine Learning

Author :

Louis BERTHIER

Email : ldb22@ic.ac.uk

CID : 02285087

Contents

List of Figures

List of Tables

Chapter A

Tuning the DQN

A.1 Hyperparameters' Selection

The hyperparameters are presented in table A.1.1 with the evolution and justification of the values.
Many **independent simulations** were performed in order to find the hyperparameters one by one even if a grid search would have been more suitable to really retrieve the optimal parameters without adjustment but it took far **too much time and computational resources** to be suitable (too many hyperparameters and values). Once these optimal hyperparameters were found one by one, **the simulations allowed to adjust them** if the combination did not give an optimal result.
However, the first parameter that was searched for was **the way in which ε should decay**. Moreover, other hyperparameters have not been explored, such as the **activation functions of neurons** (ReLu here) or the **update frequency** of the target and the main policy (each episode here).

A.2 Exploration-Exploitation Tradeoff

As stated in table A.1.1, the important point for the ε parameter is **its decay** as the agent trains. It is possible to reason in the same way as with the Q-Learning method of the previous coursework.
There is an important **tradeoff between exploration** (discovery of the space, "random actions") and **exploitation** (selection of the best actions according to the policy). The agent needs to **explore as much as possible at the beginning before focusing on the best actions** to take to maximize the reward.
Figure 1 shows the need to vary ε. Indeed, for a low ε (exploitation), the agent selects what it considers to be the **best actions** (with the Q network without exploration) which give a more or less high reward according to the accuracy of the network with **strong variations between episodes without improving the policy**. On the contrary, with a high ε (exploration), the agent will not try to obtain the best results by selecting the best actions but **only discover the environment**.

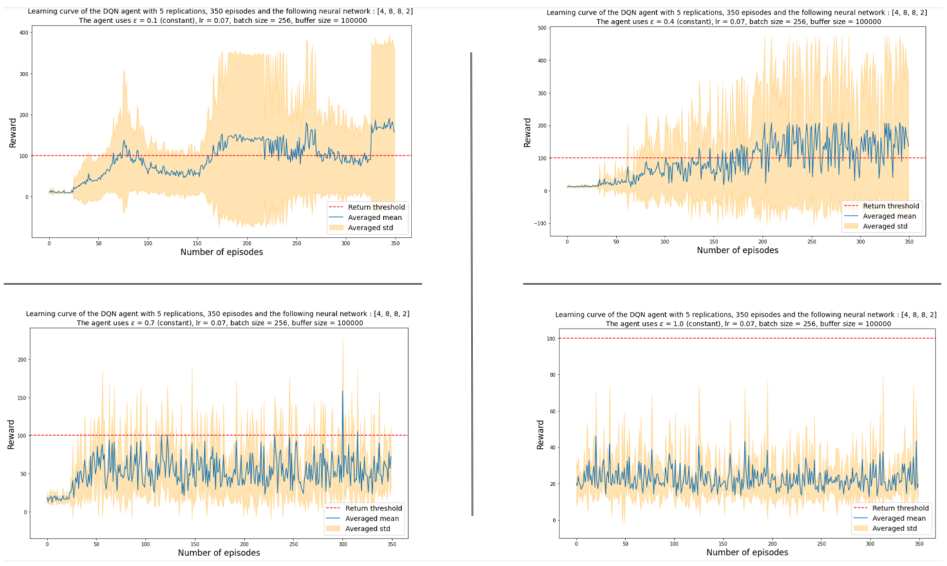
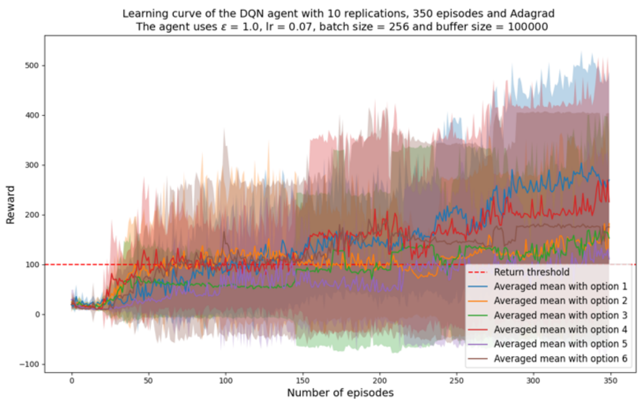


Figure 1: Multiple learning curves for different values of constant ε. **Upper left:** Learning curve with $\varepsilon = 0.1$ **Upper right:** Learning curve with $\varepsilon = 0.4$ **Lower left:**Learning curve with $\varepsilon = 0.7$ **Lower left:** Learning curve with $\varepsilon = 1$

For this, several exponential decay methods (at the end of each episode) have been implemented and converge exponentially to a low ε value (< 0.3) after 20 to 120 episodes as shown in figure 2.
Different learning curves were studied in order to **select the most interesting decay method** and finally the **method 4 will be used** as shown in figure 2.



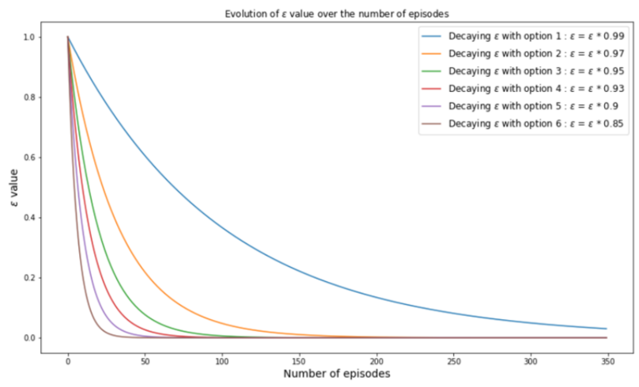


Figure 2: **Left:** Learning curves of the DQN according to the 6 decay methods **Right:** Representation of the 6 decay methods of ε

A.3 DQN's Learning Curve

Finally, after searching for and adjusting optimal hyperparameters (presented in table A.1.1), the DQN simulation was repeated **10 times** to obtain an average behavior and to **take variability into account**. The behavior of the agent is represented by the learning curve shown in figure 3.

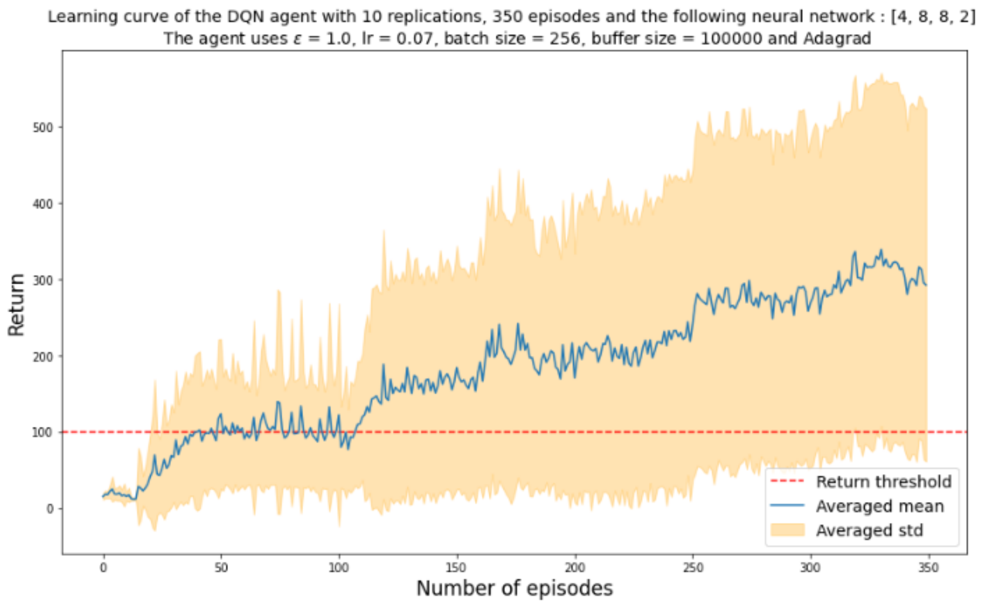


Figure 3: This graph depicts the learning curve of the DQN with the optimal hyperparameters.

The model seems to converge with a **fairly high reward compared to the performance condition**. However, the model is **not really stable** with respect to the large standard deviation. Moreover, although the reward respects the performance condition imposed by the coursework, the model is **far from the maximum reward, i.e. 500**.
Therefore it is possible to **express doubts about the selection and combination of *optimal* hyperparameters**. This is probably due to the combination of independent searches and not to the grid search which simulates all possible cases with regard to the parameters and their values.
Furthermore, it is remarkable that the model has a **large standard deviation**, which could potentially have been reduced by introducing a **decay method for the learning rate**.

Hyperparameters	First Choice	New Choice	Justification
ε (exploration parameter)	1	1 (Decay method 4)	This parameter has not been changed from an initial value point of view. However, it is modified with an **exponential decay** which is discussed in section A.2.
α (learning rate)	1	$7 * 10^{-2}$	This low learning rate allows the information provided during the update to be considered weakly. The **order of magnitude is justified by the state of the art** even though it was **determined empirically** by comparing learning curves obtained for several values of α.
Batch size (set of tuples used to train/update)	1	256	The **state of the art agrees that the optimal batch size for a DQN is between 32 and 2048**. Beyond the proposed experimentally verified value, the model becomes unstable and below it, the model will train faster but learn less. So there is a **trade-off between time and performance**.
Buffer size (memory)	1	10^5	A large buffer (compared to the state-action space) increases the standard deviation and the number of operations before leaving with a new empty buffer (and subsequently with relevant elements in memory). However, a small buffer is clearly not sufficient in terms of performance. **Experimentally this value minimizes the standard deviation** for adequate performance while limiting the possibility of the DQN forgetting due to the size of this memory.
Neural Network's dimensions	[4, 2]	[4, 8, 8, 2]	The addition of hidden layers allows the agent to better grasp the complexity of the input data and relate it to develop itself. The **shallowness of this architecture limits the learning slowdown while allowing the DQN to learn properly**.
Number of episodes	300	350	The various **simulations showed that the agent continued to improve after 300 episodes**. To maximize the return, the number of episodes was increased slightly to **improve the return or to show the convergence of the model** (stagnant curve).
Optimizer	SGD	Adagrad	Adagrad makes it possible to limit the update of a feature the more it is updated by decreasing aggressively and quickly the learning rate. The **simulations showed better results with Adagrad** and it is believed that the learning rate decay with Adagrad **allows not to include a learning rate decay method** during the episodes, unlike ε.

Table A.1.1: **Table of selection of the hyperparameters**

Chapter B

Visualise the DQN policy

For analysis and interpretation of the results, the following is a description of the parameters associated with the system:

- Parameter 1 (x): Horizontal position of the chart. Positive means the cart is on the right.
- Parameter 2 (v): Velocity of the chart. Positive means the cart is moving to the right.
- Parameter 3 (θ): Angle between the pole and the vertical position. Positive means the stick is tilted to the right.
- Parameter 4 (ω): Angular velocity of the pole. Positive means the stick rotates/falls clockwise.

B.1 Greedy Policy Action

The map of the DQN greedy policy for 4 different cart velocity values is shown in figure 4.

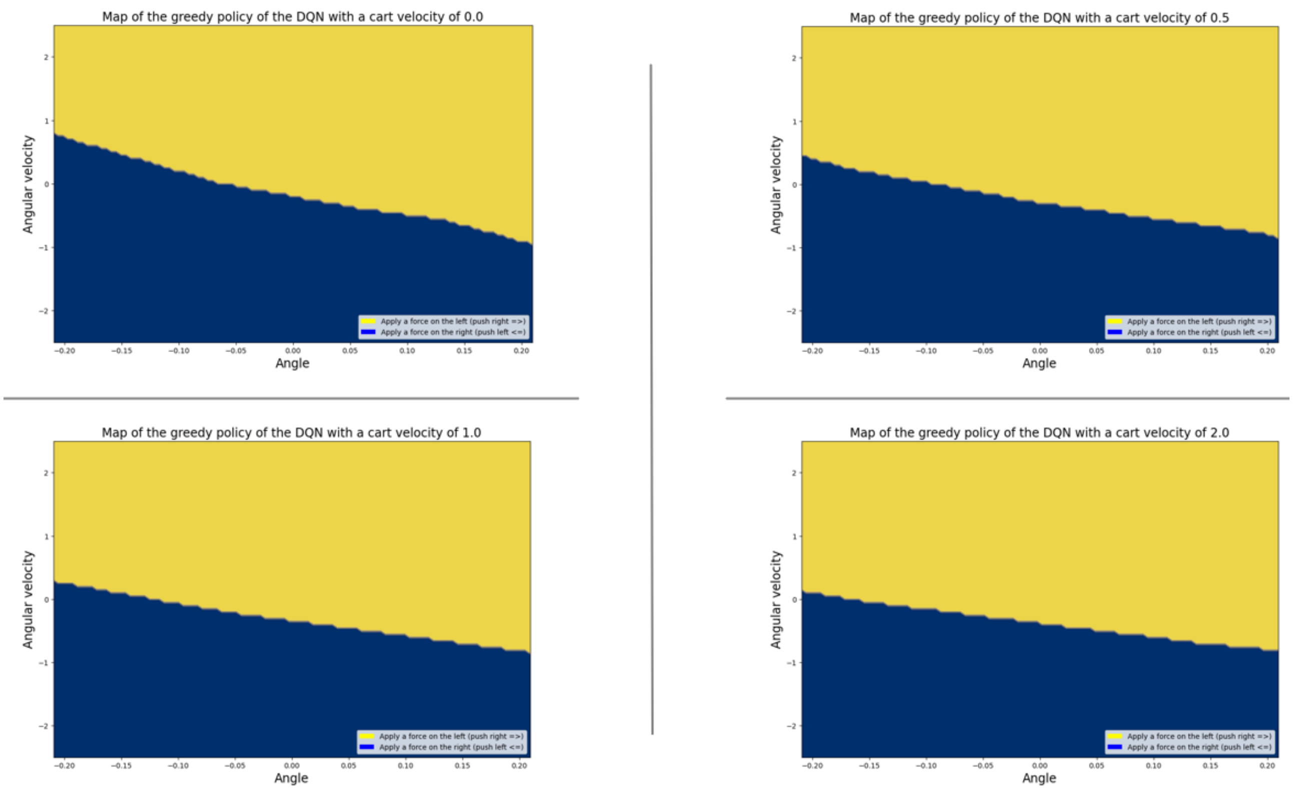


Figure 4: This figure shows slices of the agent's greedy policy for different cart speeds. **Upper left:** Cart speed of 0 **Upper right:** Cart speed of 0.5 **Lower left:** Cart speed of 1 **Lower left:** Cart speed of 2

It is possible to observe that the **agent has learnt the behavior to adopt with regard to the situation**, i.e. to push the cart to the right or to the left depending on the position of the stick and the cart and their respective speed. If the stick has a high positive angular velocity and therefore **falls clockwise**, the DQN has to **move the carriage from its initial position to the right** (yellow part) to make the stick **return to the vertical** and vice versa if the stick falls the other way.

The two actions are **separated by a diagonal** resulting from the learning of the neural network which, depending on the weight of the 4 parameters, will adjust this diagonal. This diagonal is **defined by the value and the weights associated with the angle and the angular velocity of the baton**.
The diagonal is close to a horizontal line, which means that it is the **angular velocity (y-axis) that has the most influence on the decision of the action to take** in comparison with the angle.

It is important to note that **the higher the velocity of the cart, the further down the yellow part extends** so that the diagonal remains more or less the same and **shifts downwards**. This shift can be explained by the **notion of inertia**. The higher the speed of the carriage, the more effective the backlash of the force applied to rebalance the stick.

B.2 Q values

The map of the DQN greedy policy for 4 different cart velocity values is shown in figure 5.

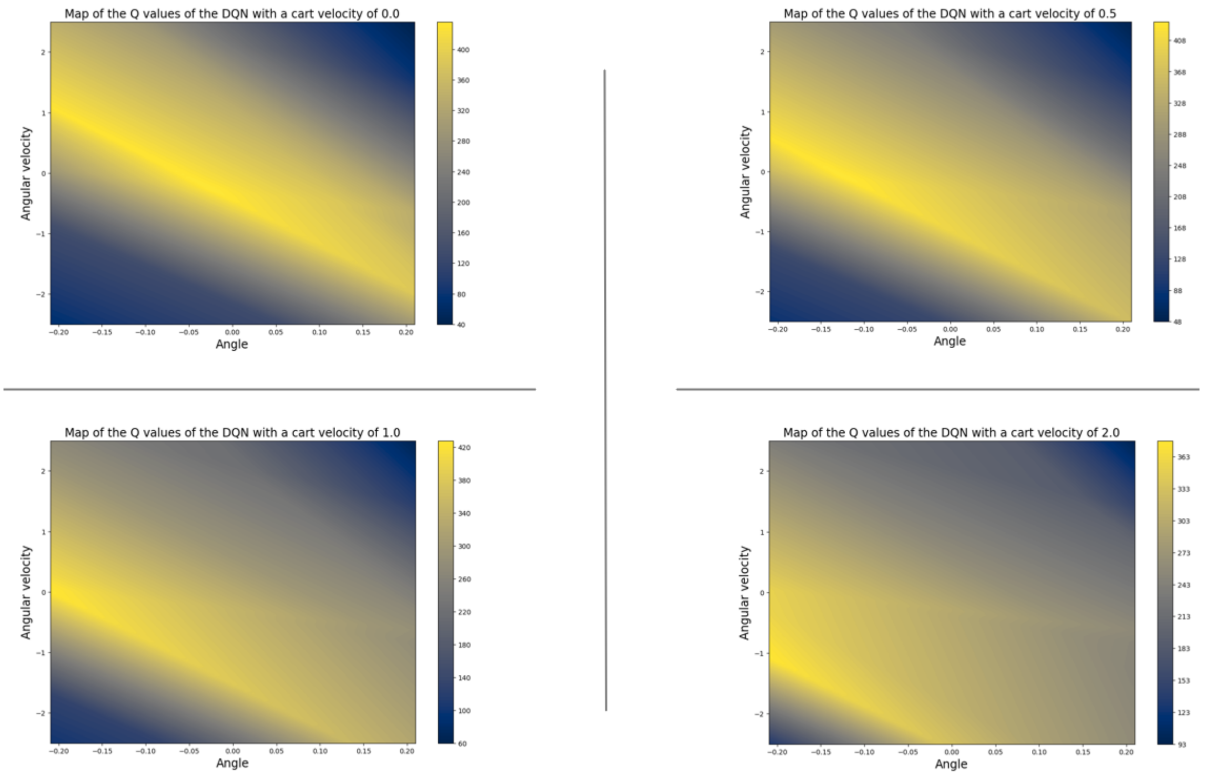


Figure 5: This figure shows slices of the agent's Q values for different cart speeds. **Upper left:** Cart speed of 0 **Upper right:** Cart speed of 0.5 **Lower left:** Cart speed of 1 **Lower left:** Cart speed of 2

With these Q values graphs, the focus is on the **states that are favorable according to the DQN and not on the action that needs to be taken** as in figure 4. The best states (giving the best reward) are in yellow, the worst in blue, and the explanations given in section B.1 about the shift down and the diagonal are also applicable to these charts.

For a velocity cart 0 and 0.5, the diagonals have **almost the same slope** as in figure 4. However, for a higher velocity, this is not the case, whereas a **similar slope should be expected**. This difference can probably be explained by **insufficient learning on the part of the DQN** (average reward between 200 and 300 and not 400 and 500).

Also the lower left corner and the upper right corner have **very low values** since these are extreme states that must not have been explored many times by the agent and for which it is **difficult to take an action that would allow him to rebalance the baton** and thus have a positive reward.

Chapter C

Transform the DQN into DDQN

To transform the DQN into a DDQN, the ***loss* function** in the utils.py file must be modified.

```
def loss(policy_dqn:DQN, target_dqn:DQN,
         states:torch.Tensor, actions:torch.Tensor,
         rewards:torch.Tensor, next_states:torch.Tensor, dones:torch.Tensor)->torch.Tensor:

    bellman_targets = (~dones).reshape(-1)*(target_dqn(next_states)).max(1).values +
                      rewards.reshape(-1)
    q_values = policy_dqn(states).gather(1, actions).reshape(-1)

    return ((q_values - bellman_targets)**2).mean()
```

The new function used to calculate the Bellman error loss and have a DDQN is the following:

```
1  def loss_DDQN(policy_dqn:DQN, target_dqn:DQN,
2           states:torch.Tensor, actions:torch.Tensor,
3           rewards:torch.Tensor, next_states:torch.Tensor, dones:torch.Tensor)->torch.Tensor:
4
5      best_index = policy_dqn(next_states).max(1).indices.reshape([-1, 1])
6      bellman_targets = (~dones).reshape(-1)*(target_dqn(next_states)).gather(1, best_index).
7                        reshape(-1) + rewards.reshape(-1)
8      q_values = policy_dqn(states).gather(1, actions).reshape(-1)
9
10 return ((q_values - bellman_targets)**2).mean()
```

The main idea behind the DDQN is to **reduce the overestimates of the DQN** by **separating the selection of actions and the evaluation of these actions** (prediction of Q-values), and thus to have **unbiased estimates**. For this purpose, the DDQN uses **two Q-networks also called estimators**.
In the second code with the DDQN implementation, the policy network is used to select the best action *(lines 5 and 8)* while the target network is used to evaluate the best action *chosen (lines 6 and 7)* at each episode.

Using the optimal parameters defined in table A.1.1, simulations were performed for the DDQN and again for the DQN. These are shown in figure 6.

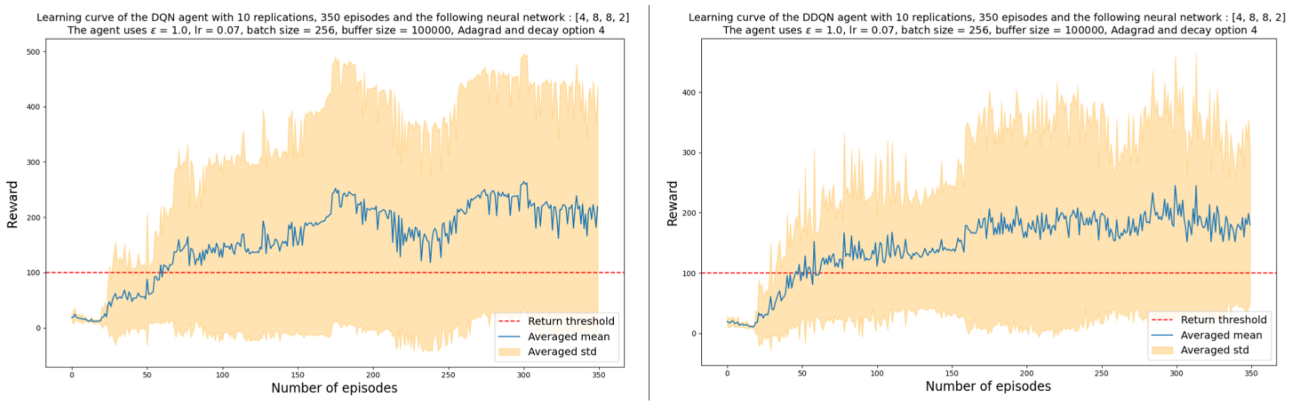


Figure 6: This figure depicts the learning curves of the two agents obtained with the optimal hyper-parameters. **Left:** DQN **Right:** DDQN

It is worth noting that the **DDQN does not show better results when talking purely about reward**. It is the **same order of magnitude** for this simulation with an average reward of 150 which seems to converge towards 200.
On the other hand, it is important to note that the **variance has decreased** through the use of the DDQN (although it is still too high). Also, the **curve seems much smoother with much smaller variations in amplitude between each episode** than with the DQN. This reduction in variance and inter-episode amplitude variation is **evidence of the reduction in bias when predicting the Q-values**.