Robot Learning - Laboratory 2
Tabular Q-Learning and Deep Q-Learning

Andrea Delli (S331998)
Robot Learning course (01HFNOV)
Master's Degree in Computer Engineering
Artificial Intelligence and Data Analytics
Politecnico di Torino

Abstract: This laboratory explores Tabular Q-Learning and Deep Q-Learning (DQL) applied to the CartPole environment, a classic reinforcement learning benchmark. Tabular Q-Learning, employing state-space discretization, is effective for small-scale problems but struggles with scalability. Experiments on ϵ strategies and initialization values reveal their impact on learning performance. DQL overcomes these limitations by using neural networks, enabling direct handling of continuous spaces. Stabilization techniques like target networks and experience replay enhance DQL's efficiency, achieving consistent maximum rewards.

1 Introduction

The Cart-Pole environment is a fundamental and widely-used benchmark in reinforcement learning (RL), designed to test control strategies and decision-making algorithms. The setup consists of a cart that can move along a one-dimensional track and a pole mounted on the cart, free to pivot around its attachment point. The primary goal of the task is to balance the pole upright by applying forces to the cart to move it left or right.

Key components of the environment include a state observation vector s containing four elements: the cart's position x and velocity $\dot{x}$, the pole's angle θ relative to the vertical axis, and the pole's angular velocity $\dot{\theta}$. The agent's objective is to maintain the pole in a vertical position while keeping the cart within the track's boundaries. Any significant deviation, such as the pole falling or the cart moving out of bounds, results in a failure.

$$s = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

In this laboratory we focus on **Q-Learning**, which is an off-policy, model-free reinforcement learning (RL) algorithm used to find the optimal action-value function $Q^*(s, a)$ for a given Markov Decision Process (MDP). The goal is to determine an optimal policy π^* that maximizes the cumulative discounted reward. Q-Learning is based on the Bellman optimality equation:

$$Q^*(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s', a') | s, a\right]$$

where:

- s and a are the current state and action
- s' is the next state
- r is the reward for transitioning from s to s'
- γ is the discount factor

The Q-function is updated iteratively as:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

where α is the learning rate.

2 Tabular Q-Learning

Tabular Q-Learning represents the Q-function as a table, where each state-action pair (s, a) has a corresponding Q-value. This approach works well for environments with discrete and small state-action spaces. For example, in a grid world, the Q-values can be stored as a 2D array.

In the CartPole environment used in this laboratory, the standard space is continuous, so we need to discretize it. In order to do so, we define a specific range of values for each of the state measures, defined as follows:

- Position: $x \in [-2.4, +2.4]$
- Velocity: $\dot{x} \in [-3, +3]$
- Angle: $\theta \in [-0.3, +0.3]$
- Angular velocity: $\dot{\theta} \in [-4, +4]$

Each of these ranges is equally divided in 16 points, so the total state space can have 16^4 possible values and thus we'll have a total of 16^4 elements in the Q-value function matrix. Each space given by the continuous environment is automatically converted to the nearest discrete space observation.

This approach is simple and it's easy adaptable to simple continuous environments, but it's an approximation and it's not scalable to large, complex and continuous state-action spaces due to memory constraints.

In the first experiment we consider an ϵ-greedy policy as strategy to deal with the exploration-exploitation trade-off during training. In particular, we consider 2 different approaches:

1. A constant value of $\epsilon = 0.2$
2. A decreasing value of ϵ, following a GLIE (Greedy in Limit with Infinite Exploration) schedule, where the value of ϵ at iteration k is given by $e_k = \frac{b}{b+k}$

In the second case, we want to have the final value of $e_k = 0.1$ after 20000 episodes ($k = 20000$), so the value of b has to be:

$$e_k = \frac{b}{b+k} \implies e_k(b+k) = b \implies b(1-e_k) = e_k k \implies b = \frac{e_k k}{1-e_k}$$

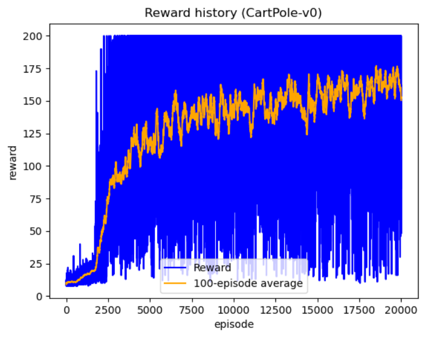
So if we want the final iteration $k = 20000$, for $e_k = 0.1$ we will have:

$$b = \frac{0.1 \times 20000}{1 - 0.1} = \frac{2000}{0.9} \simeq 2222.222$$

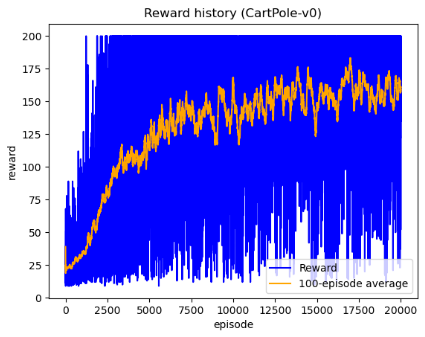
The plots reported in Fig. 1 show the reward history for the two different ϵ schedules: constant $\epsilon = 0.2$ (Fig. 1a) and GLIE (Fig. 1b).

The plots show a high variance in the training episodes, even though by looking at the average reward both schedules have a similar increasing trend, meaning that the agents are learning a better policy when updating the Q-values inside the matrix.

The performance of both models is very similar, and this is also confirmed by repeatedly testing the final models and as shown in Table 1.



(a) Reward history with constant $\epsilon = 0.2$



(b) Reward history with ϵ following a GLIE schedule

Figure 1: Reward history for tabular Q-leaning with different ϵ values

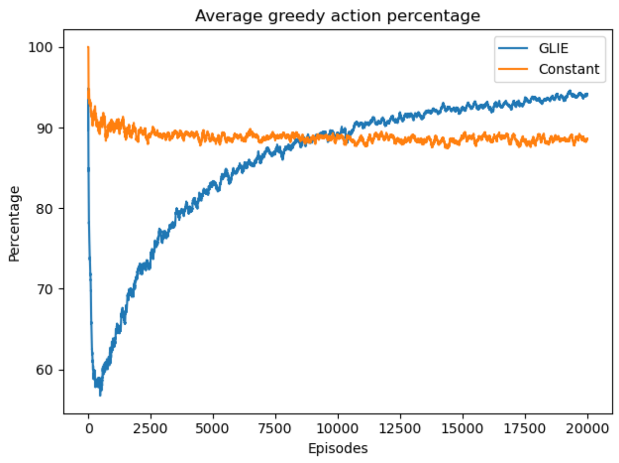


Figure 2: Percentage of times in which the action taken was greedy during training, for both ϵ schedules: GLIE and constant

Another interesting plot is shown in Fig. 2, where each line represent the percentage of times the chosen action was greedy in every episode during training.

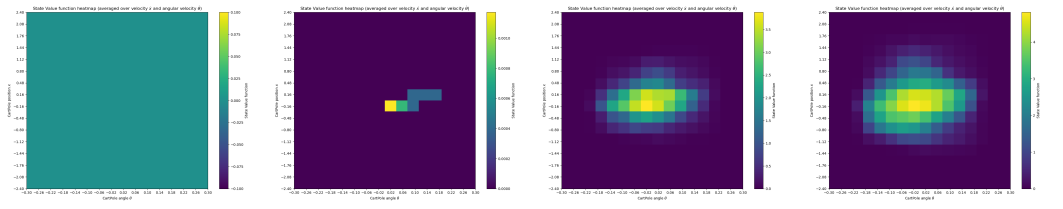
The plot shows that by using a constant $\epsilon = 0.2$ the percentage of times in which the action taken is completely greedy is around 90%. This is expected because 80% of the times the policy acts greedily (by definition of ϵ-greedy), while the other 20% of the time the action is chosen randomly between the two possible actions (left and right), resulting in a total of $\simeq 80\% + \frac{1}{2}(20\%) = 90\%$.

The GLIE schedule instead starts by favoring exploration (with a value of ϵ near to 1), meaning that the chosen action is random (in fact the starting point of the line is around 50%), and then acting more and more greedily until the final iterations, where we have a value of ϵ near to 0.1, so we have choose greedily $\simeq 95\%$ of the times $(90\% + \frac{1}{2}(10\%) = 95\%)$.

Another interesting observation can be made by plotting the heatmaps of the tabular Q-function in terms of x and θ, so averaging the values over $\dot{x}$ and $\dot{\theta}$, as shown in Fig. 3 and 4, for both ϵ schedules. It's important to notice that the scale of the heatmaps changes, and this is done because it allows to see even the slight change (for example from episode 0 to episode 1), that otherwise with a "full" scale (e.g. from 0 to 5) wouldn't be visibly noticeable.

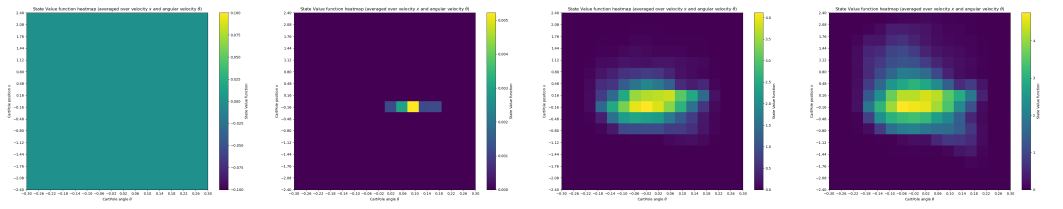
As expected, in episode 0 (Fig. 3a and Fig. 4a) both algorithms show an uniform image where the Q-function value has the initial value of 0 in all possible states. After one episode (Fig. 3b and Fig.

4b) the Q-function slightly increases near the value of 0 for both axis, showing that the function slowly starts to learn the optimal position and angle. This trend continues also after 10000 episodes (Fig. 3c and Fig. 4c), where we clearly see that the policy favors stability states (position and angle near to 0). The trend continues for the full training with 20000 episodes (Fig. 3d and Fig. 4d), and the final heatmaps suggest that the polcy tends to favor positions and angles near to 0, both because it's the agent's starting point and because it's the "safest" spot in the environment, being as further as possible from the episode termination (e.g. for large angles or for exiting the environment [1]).



(a) Episode 0 (b) Episode 1 (c) Episode 10000 (d) Episode 20000

Figure 3: Heatmaps of the Q-function values obtained when training with constant $\epsilon = 0.2$, obtained by averaging the values over $\dot{x}$ and $\dot{\theta}$, for different timesteps (0, 1, 10000, 20000)



(a) Episode 0 (b) Episode 1 (c) Episode 10000 (d) Episode 20000

Figure 4: Heatmaps of the Q-function values obtained when training with ϵ following a GLIE schedule ($e_k = \frac{b}{b+k}$), obtained by averaging the values over $\dot{x}$ and $\dot{\theta}$, for different timesteps (0, 1, 10000, 20000)

Another interesting experiment is to set $\epsilon = 0$, thus using a greedy policy. In this case the policy has a large dependency on the initial values of the Q-function, which has been tested using two different initialization values: 0 and 50.

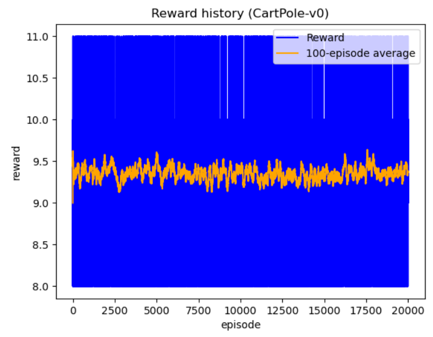
The reward history during training for both approaches is reported in Fig. 5, where we have the left plot (Fig. 5a) initialized with 0 and the right plot initialized with 50 (Fig. 5b).

As explained in [2], the implemented tabular Q-learning is dependent on the initial action-value estimates, i.e. it's biased by the initial estimates. Initial action values can also be used as a simple way to encourage exploration, as it happens for the initial value of 50 (Fig. 6b), which is an over-optimistic view that encourages the agent to explore, because whatever action the agent takes the reward will be less than the starting estimate, and will try another action that has never tried before. This technique for encouraging exploration is also denoted as *optimistic initial values*.

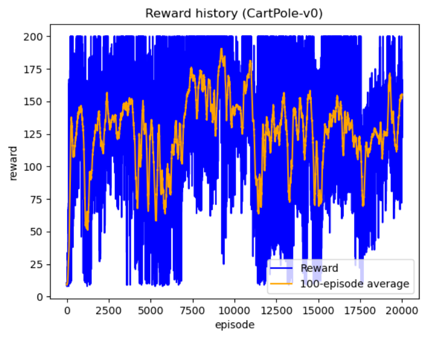
Both plots show that in both cases the policies didn't converge towards an optimal policy, and the case with 0 as initial value performs poorly (Fig. 6a).

The learned Q-function value generated with the greedy approaches is shown in the heatmaps reported in Fig. 6. In the case of initial value set to 0 (Fig. 6a) the learned policy favors values of x and θ near to 0, which are the only values explored as they are the only ones near the initial state. The range of values continues to be near to the initial value, in the range $[0, +0.3]$.

For the Q-function initialized with 50 (Fig. 6b) the model has explored many of the central states, in particular all angle ranges (y axis) and the position ranges between ± 1.8. This highlights the



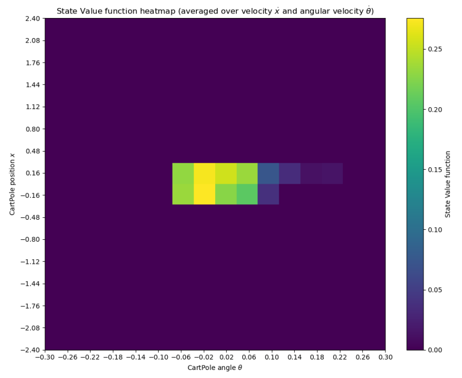
(a) Reward history with greedy policy ($\epsilon = 0$) and initial Q-function values equal to 0



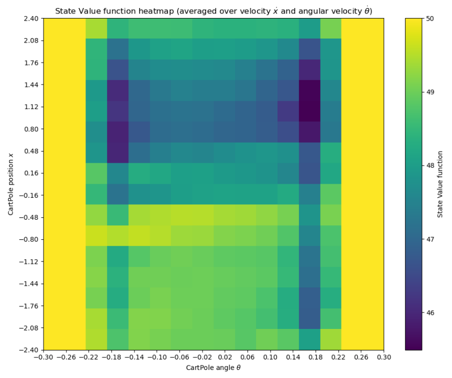
(b) Reward history with greedy policy ($\epsilon = 0$) and initial Q-function values equal to 50

Figure 5: Reward history for tabular Q-leaning with $\epsilon = 0$ and different initial Q-function values

propension of the agent to favor exploration, because it tends to prefer states with higher Q-function values, which are the ones that it has not explored yet.



(a) Heatmap with initial Q-function values equal to 0



(b) Heatmap with initial Q-function values equal to 50

Figure 6: Heatmaps of the Q-function values obtained after training with $\epsilon = 0$, obtained by averaging the values over $\dot{x}$ and $\dot{\theta}$

The average performance of the generated models during testing is reported in Table 1, where the performance is measured as the average number of timesteps for which the model survives (1 timestep $\implies$ +1 reward). By analyzing these results, we can conclude that Q-learning can be adapted for environments with continuous state spaces using approximation techniques like discretization, or using other approximation techniques like neural networks as experimented in Section 3 where the neural network approximates the Q-value for any given state-action pair by learning a mapping between state-action inputs and their corresponding Q-values. The challenge of applying Q-learning directly to continuous action spaces is due to the Q-function update rule, where we need to find the best action for the given state ($max_a Q(s, a)$), which may be challenging to compute.

ϵ schedule	GLIE ϵ	Constant $\epsilon = 0.2$	$\epsilon = 0$ Init Q-values to 0	$\epsilon = 0$ Init Q-values to 50
Average timesteps (max: 200)	181.55	189.76	9.39	160.48

Table 1: Performance comparison between different ϵ schedules, averaged over 100 test episodes

3 Deep Q-Learning (DQN)

Deep Q-Learning extends tabular Q-Learning by using a neural network as a function approximator to estimate the Q-values, and can also be applied to continuous spaces. The neural network takes a state s as input and outputs Q-values for all possible states. This approach is scalable and can be used in continuous and high-dimensional state-action spaces, thus in this laboratory is applied directly in the continuous CartPole environment. The downside is that the network training is computationally intensive and training can be unstable, requiring techniques like experience replay and the usage of a target network and hyperparameter tuning.

The network is composed by three fully-connected layers connected using non-linear ReLU activation functions. The input layer has 4 input values (because we have 4 elements as state, as described in Section 1) and 128 output values, while the output layer has 128 inputs and 2 outputs (left and right), and of course the hidden layer has 128 nodes.

One of the critical elements is the use of an ϵ-greedy exploration strategy in order to balance exploration and exploitation. Initially, epsilon is set high ($\epsilon = 1$) to encourage exploration, and it gradually decreases after each epoch using a decay multiplier as the agent becomes more confident in its learned policy, ensuring more exploitation of the optimal actions.

The stability of training is significantly enhanced by employing a **target network**. This is a separate neural network that mirrors the structure of the main Q-network but is updated less frequently. During training, the target network remains fixed for a specified number of steps or episodes, providing stable target Q-values when computing the loss. This stabilization helps to mitigate the issue of oscillating or diverging Q-value estimates that can arise when the main network is updated frequently. Periodically, the weights of the target network are updated to match those of the main Q-network.

Experience replay is another integral component that has been implemented. It uses a buffer to store past experiences, which are sampled randomly to train the Q-network. In particular, an element inside the buffer is composed by 5 elements:

- state s: current state
- action a: action taken in state s
- reward r: reward obtained by taking action a in state s
- next state s': destination state taking action a in state s
- done: boolean value that indicates whether the episode is terminated

By breaking the temporal correlation between consecutive experiences, experience replay reduces variance in updates and improves training stability. The agent learns from a diverse set of experiences rather than relying solely on the most recent interactions with the environment. To speed up the training, past experiences are sampled in batches of 32 elements processed in parallel.

During training, to compute the loss (MSE loss optimized using Adam with learning rate 0.0001) the Bellman equation is used for computing target Q-values using the target network:

$$Q(s, a) = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')$$

These targets incorporate the immediate reward received and the discounted maximum future Q-value predicted by the target network. The Q-network is then updated by minimizing the mean squared error between its predicted Q-values and these computed targets. This process ensures that the network gradually learns to predict accurate Q-values, aligning with the optimal policy.

The plot in Fig. 7 shows the reward history during training of the Q-network with 1000 epochs. The plot shows a stable increasing of the average reward, meaning that the model is successfully learning a good policy to make the CartPole survive the whole episode. In fact, the average reward of the final model (computed by executing 100 episodes using the trained network) is 200, which is the best score obtainable. The plot also shows that the variance of the models decreases significantly over time, meaning that the ϵ schedule is successfully working.

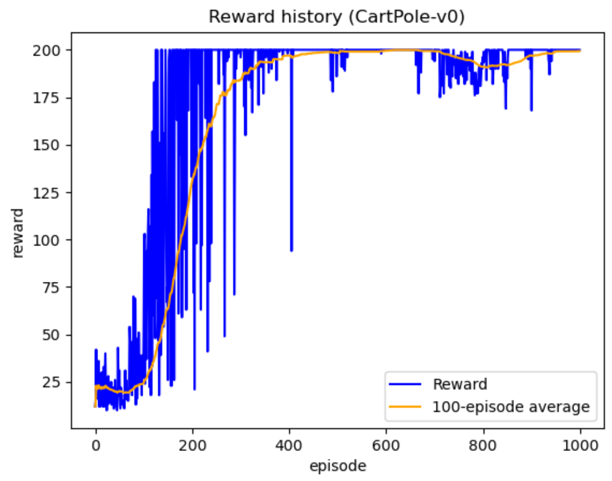


Figure 7: Reward history obtained during the Deep Q-Learning training

The used metrics and hyperparameters used to train the final model are reported in Table 2.

Measure	**Value**
Loss	Mean Square Error
Optimizer	Adam
Learning rate	0.0001
Discount factor γ	0.98
Target network update	every 10 epochs
Batch size	32
Initial ϵ	1
ϵ decay	0.995
Minimum ϵ	0.01
Experience replay buffer size	10000

Table 2: Metrics and hyperparameters used when training the Deep Q-network

4 Conclusion

This laboratory explored the application of Tabular Q-Learning and Deep Q-Learning to the CartPole environment, highlighting the strengths and limitations of each approach. Tabular Q-Learning demonstrated its utility in discrete state-action spaces by discretizing the continuous state space of the environment. While this method is effective for small-scale problems, its scalability is constrained by the exponential growth of the state-action space as complexity increases. Additionally, experiments with varying exploration strategies revealed the importance of epsilon schedules and initial Q-value estimates in guiding learning and promoting exploration.

In contrast, Deep Q-Learning overcame the scalability challenges by employing a neural network as a function approximator, enabling it to handle continuous and high-dimensional state spaces directly. The incorporation of techniques such as experience replay and target networks played a crucial role in stabilizing the training process. The use of the Mean Squared Error loss function, optimized with the Adam optimizer, facilitated the accurate approximation of Q-values. Experimental results confirmed the effectiveness of DQL, with the trained agent consistently achieving the maximum possible reward in the CartPole environment.

Overall, the results demonstrate the progression from traditional tabular methods to more advanced deep learning techniques in reinforcement learning. While Tabular Q-Learning serves as an intuitive introduction to RL concepts, Deep Q-Learning exemplifies the power and flexibility of neural networks in solving complex problems.

References

[1] Gym. Cartpole documentation. URL https://www.gymlibrary.dev/environments/classic_control/cart_pole/.

[2] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.