# Lab 3 - Reinforcement Learning

David Padilla Orenga
Ignacio Pastore Benaim

November 14, 2024

## Previous work:

- `computeActionFromValues(state)`: This method computes the best action to perform from a given state according to the current value function (self.values). Mathematically, it selects the action that maximizes the expected value, considering all possible transitions from that state:

$$\pi(s) = \arg\max_a Q(s,a)$$

  where $Q(s,a)$ represents the action-value function, calculated from the value function $V(s)$.

- `computeQValueFromValues(state, action)`: This function calculates the $Q$-value for a specific state-action pair, representing the expected utility of taking action $a$ in state $s$. The formula is given by:

$$Q(s,a) = \sum_{s'} P(s' \mid s,a)\,[R(s,a,s') + \gamma V(s')]$$

Both equations are related to the Value Iteration Equation that automatically calculates the optimum value for each state using the Bellman optimality equation:

$$V_k(s) = \max_a \sum_{s'} P(s' \mid s,a)\,[R(s,a,s') + \gamma V_{k-1}(s')]$$

## 1 Value Iteration

This implementation uses the batch version of Value Iteration, where the state values are updated using the results of the previous iteration, instead of modifying the values directly during the calculation. The getStates() function lists all states in the environment, while getPossibleActions(state) indicates the available actions from a given state. On the other hand, computeQValueFromValues(state, action) computes the Q value considering immediate rewards and discounted values. The results are stored in self.values, and the generated policy with depth values represents the rewards up to steps.

## 2 Bridge Crossing Analysis

Gamma is high enough to guide the agent to the future highest reward terminal state. In this case, the noise is causing the agent to fall in a non-desirable state, giving the high negative rewards beside the bridge. Consequently, the agent tends to prefer the low reward terminal state.

In order to reach the high reward terminal state we must reduce the noise, giving gamma the prevalence to guide the agent to the desired terminal state. In our case we set the noise to 0.

## 3 Policies

On the one hand, **Discount Factor** controls how much the agent values future rewards. A low Discount favors policies that prefer immediate rewards, such as near exit, while a high Discount (Gamma $\times$ $0.9\times$) encourages exploring long-term options, such as far exit. On the other hand, **Noise** introduces uncertainty in the agent's actions. If the noise is low (`Noise` = 0)), the agent follows a predictable path,

| Question | Discount | Noise | Living Reward |
|---|---|---|---|
| 3a | 0.1 | 0 | 0 |
| 3b | 0.1 | 0.1 | 0 |
| 3c | 0.9 | 0 | 0 |
| 3d | 0.9 | 0.1 | 0 |
| 3e | 1 | 0 | 0 |

Table 1: Parameter adjusted for each policy.

even if it is risky, such as approaching the cliff. If the noise increases (`Noise ¿ 0`)), the agent prefers safer paths and avoids risk, such as moving away from the cliff.

## 4 Q-Learning

In this part Q-Learning is implemented. In real cases, it is not always known how the environment works, so the agent has to learn from experience. The Q-values are stored in a table organized by state and action, and from them the value of each state is calculated and the policy that the agent will follow is decided.

## 5 Epsilon Greedy

In this part, it was observed that, after a sufficient number of iterations in which the agent learns how to act correctly, should decrease from its initial value. At first, a high value of is useful for the agent to explore different options and discover the best strategies. However, once he has learned enough, it is better to reduce so that the agent takes advantage of what he already knows and maximizes rewards rather than continuing to try new actions unnecessarily. This ensures that the agent is more efficient in its learned behavior.

## 6 Bridge Crossing Revisited

Our conclusion is that it is **NOT possible** to guarantee that the agent learns the optimal policy with a probability greater than 99% in only 50 iterations due to several limitations of the environment and Q-learning:

1. First, to learn the optimal path crossing the bridge, the agent needs to explore dangerous routes such as edges with negative rewards. If the $\epsilon$ (the exploration rate) is high, the agent will explore more, but it will also receive many negative rewards, demotivating it. If $\epsilon$ is low, the agent will not explore enough to find the bridge in so few iterations.

2. Very limited learning rate. In 50 iterations, even if the agent finds the bridge, the updates to the Q values will not be enough to clearly reflect that this is the best route.

3. The last problem we encounter is convergence. Learning the optimal policy depends on exploring, updating the Q values, and repeating this many times. With only 50 steps, the agent does not have time to explore, learn and enforce the best policy.

In short, the problem is that 50 iterations are too few for the agent to explore enough and update its Q-values consistently. Therefore, it is **not possible** to guarantee that it learns the optimal policy with a probability greater than 99%.

## Q-Learning Pacman

The $\epsilon$-greedy Q-learning agent performed well on the small grid, successfully learning an optimal policy after sufficient training. However, on the medium grid, the agent failed to achieve a good policy. This is because the medium grid introduces a much larger state space, making it harder for the agent to explore all relevant states within a limited number of iterations. Additionally, the lack of generalization in the basic Q-learning approach means that the agent treats each state individually, requiring significantly more time and data to learn effective Q-values for such a large environment.