Artificial Intelligence and Machine Learning

Giuseppe Valente
Sapienza - University of Rome
Engineering in Computer Science

Contents

Chapter 1

Introduction

This work aims to study and solve Reinforcement Learning (RL) problem, using models like Q-learning and Deep Q-Network (DQN). We will introduce a basic problem called Frozen Lake, and we will study learning strategies that help the agent to solve the environment. The challenges in RL are understanding how an agent can explore and exploit an environment effectively to achieve its goal. Q-learning is a model-free RL algorithm that teaches an agent to assign values to each action it might take, conditioned on the agent being in a particular state. For any finite Markov decision process, Q-learning finds an optimal policy (maximizing the expected value of the total reward over any and all successive steps), starting from the current state. While effective for simple environments, its scalability is limited when dealing with larger or more complex state spaces. To address this limitation, DQN employs deep neural networks to approximate the Q-value function, enabling the application of RL to high-dimensional and complex environments. With this study, we will see the theoretical foundation and the practical implementation of these methods, evaluating their performance in solving the Frozen Lake problem.

Chapter 2

Reinforcement Learning

RL is one of three basic machine learning of machine learning concerned with how an agent should take actions in a dynamic environment to maximize a reward. Finding a balance between exploration (of uncharted environment) and exploitation (of current knowledge) with the goal of maximize the cumulative reward. The environment is typically stated in the form of a Markov decision process (MDP). The purpose of reinforcement learning is for the agent to learn an optimal policy that maximizes the reward function or other user-provided reinforcement signal that accumulates from immediate rewards.

2.1 Agent's learning task

A basic reinforcement learning agent interacts with its environment in discrete time steps. At each time step t, the agent receives the current state S_t and reward R_t. It then chooses an action A_t from the set of available actions, which is subsequently sent to the environment. The environment moves to a new state S_{t+1} and the reward R_{t+1} associated with the transition (S_t,A_t,S_{t+1}) is determined. The goal of a reinforcement learning agent is to learn a policy

2.2 Q-Function

The Q-Function (the state-action) function is a function that helps the agent to decide which action to take in a given state in order to maximize its expected cumulative reward. In general we can write the Q-Function using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \tag{2.1}$$

Q(s, a): Current estimate of the expected cumulative reward for taking action a in state s and following the optimal policy thereafter.
Q(s, a): The current Q-value for the state-action pair (s, a).
α**:** The *learning rate*. It controls how much the new information should override the old information. A higher value means the agent will quickly adapt to new experiences.
r: The *immediate reward* received after taking action a in state s.
γ**:** The *discount factor*. It determines how much the agent values future rewards compared to immediate rewards. A value close to 1 means the agent cares about long-term rewards,

while a value close to 0 means the agent prioritizes immediate rewards.
$\max_{a'} Q(s', a')$**:** The maximum Q-value over all possible actions a' in the next state s'. This represents the best expected reward the agent can get from the next state onward.
Q(s, a): The original Q-value before the update, which is updated based on the new experience.
$Q(s, a) \leftarrow$**:** The leftward arrow indicates that the Q-value is being updated with the new value on the right side of the equation.

2.3 Deterministic Environment

In a *deterministic* environment, the outcome of any action taken by the agent is completely predictable. For a given state s and action a, the next state s' is always the same. The environment follows strict rules, meaning that there is no randomness involved in the transition from one state to another. The reward associated with an action is also deterministic, meaning the agent always receives the same reward for performing the same action in the same state. The agent can rely on complete predictability and plan its actions with full certainty about the consequences. In deterministic environments, the Q-value can be updated directly based on the immediate reward and the known future reward, so α isn't needed. Then the update rule for the Q-function is:

$$Q(s, a) \leftarrow r + \gamma Q(s', a') \tag{2.2}$$

2.4 Non-deterministic Environment

In a *non-deterministic* environment, the outcome of an action is uncertain. For a given state s and action a, the next state s' is probabilistic, and the reward r may vary. The Q-function in a non-deterministic environment represents the expected cumulative reward, taking into account the uncertainty in state transitions and rewards. The agent must learn not only the value of taking an action in a given state but also the probability distribution of possible outcomes. The update rule is 2.1 tcomes can vary each time the same action is taken in the same state.

2.5 Generic Q-Learning algorithm

The following is the pseudocode for the Q-learning algorithm using the ϵ-greedy strategy in both deterministic and non-deterministic environments:

Algorithm 1 Q-learning Algorithm (with ϵ-greedy)

Input: Learning rate α, discount factor γ, exploration rate ϵ, number of episodes N
Initialize: Q-table $Q(s, a)$ for all states s and actions a, set episode count $i = 1$
while $i \leq N$ **do**
    Initialize starting state s_1
    while not terminal state s_t **do**
        Choose action a_t based on the ϵ-greedy policy:
        if random number $r \leq \epsilon$ **then**
            Select a random action a_t
        else
            Select action $a_t = \arg\max_a Q(s_t, a)$ (greedy action)
        end if
        Take action a_t, observe reward r_t and next state s_{t+1}
        Update Q-value:
        if deterministic environment **then**
            $Q(s_t, a_t) \leftarrow r_t + \gamma Q(s_{t+1}, a')$
        else
            $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right)$
        end if
        $s_t \leftarrow s_{t+1}$
    end while
    $i \leftarrow i + 1$
end while

2.6 Hyperparameters

They are parameters set before the learning process begins, play a crucial role in determining the efficiency and effectiveness of the learning process.

- **Learning rate (α)**: Controls how much the Q-values are adjusted during each update. A larger learning rate leads to faster updates, but can also result in instability. A smaller learning rate may lead to slower learning.
- **Discount factor (γ)**: Determines how much the agent values future rewards compared to immediate rewards. A discount factor close to 1 means the agent values long-term rewards, while a value close to 0 means the agent focuses on immediate rewards.
- **Exploration rate (ϵ)**: In the ϵ-greedy strategy, this parameter controls the probability with which the agent chooses a random action (exploration) instead of the action with the highest Q-value (exploitation). It helps balance exploration of new actions with exploiting the known best actions.
- **Number of episodes (N)**: Specifies the number of training episodes or iterations over which the agent will interact with the environment and learn. More episodes typically lead to better learning but require more computation time.

Choosing the right set of hyperparameters is crucial for training an effective model. Poor choices of hyperparameters can lead to slow convergence, overfitting, or even complete failure of the algorithm to learn useful behavior.

2.7 Exploitation and Exploration balance

2.8 Measure the optimal policy

Chapter 3

Deep Q-Learning

Chapter 4

Frozen Lake Project