Autonomous Taxi Agent Using Reinforcement Learning

1st ILLAHIBUCCUS SONA Ishfaaq
ishfaaq.illahibuccus-sona@unice.fr

2nd GHOSH Prabal
prabal.ghosh@unice.fr

3rd HAMMED Habeeb Olawale
habeeb.hammed@unice.fr

***Abstract*—This report presents our implementation and comparison of four Reinforcement Learning (RL) algorithms—Q-Learning, SARSA, Deep Q-Network (DQN), and Value Iteration—to train an autonomous taxi agent in the Taxi-v3 environment from OpenAI Gym. Our results highlight their strengths and weaknesses in optimizing agent performance.**

I. Introduction

Reinforcement Learning (RL) is a type of machine learning where an agent (AI or program) learns by interacting with an environment. The goal is to take actions that maximize rewards over time. The agent learns through trial and error and improves its decisions over time. There exist different algorithms that can be used to compute optimal policies depending on the objectives:

II. Project Overview

This project implements and compares four Reinforcement Learning (RL) algorithms—**Q-Learning, SARSA, Value Iteration and Deep Q-Network (DQN)** to train an **autonomous taxi agent** in the `Taxi-v3` environment from OpenAI Gym. The agent must efficiently **pick-up and drop-off passengers** while optimizing rewards.

A. Objectives

1) Train an autonomous taxi using **Q-Learning, SARSA, Value Iteration, and DQN**.
2) Compare performance across these algorithms based on **reward progression, learning efficiency, and policy convergence**.
3) Visualize learning behavior through **reward plots and agent animations**.
4) Analyze exploration-exploitation trade-offs in RL policies.

B. Environment: OpenAI Gym Taxi-v3

The `Taxi-v3` environment consists of a 5×5 **grid** where:

- The taxi must **pick up and drop off passengers** at designated locations.
- The action space includes **moving up, down, left, right, picking up, and dropping off passengers**.
- Rewards:
  - $+20$ for successful passenger drop-off.
  - -1 for each step taken.
  - -10 for incorrect drop-off.

III. Algorithms

The backbone of RL algorithms revolves around estimating the **Q-value function**, which represents the expected cumulative reward for taking an action in a given state and following an optimal policy thereafter. Mathematically, we can express the the Q-value function as:

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{a} Q(s',a') - Q(s,a) \right] \quad (1)$$

where:

- s is the current state, and a is the action taken.
- $Q(s,a)$ is the estimated Q-value for state-action pair (s,a).
- α is the learning rate, which contains how much new information overrides previous knowledge.
- r is the immediate reward received after taking action a.
- γ is our discount factor, representing the balance between future rewards and immediate ones.
- s' is the next state, and a' is the next action.

A. Q-Learning

Q-Learning is an **off-policy** algorithm that updates Q-values using:

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \quad (2)$$

It uses ϵ**-greedy policy** for action selection and updates Q-values based on the maximum expected future reward while ϵ decreases exponentially.

The ϵ**-greedy policy** balances exploration and exploitation by selecting:

- A random action with probability ϵ (exploration).
- The action with the highest Q-value otherwise (exploitation).

The value of ϵ decreases with time, shifting the agent from exploration to exploitation.

B. SARSA

SARSA (State-Action-Reward-State-Action) is an **on-policy** algorithm that updates Q-values using:

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right] \quad (3)$$

Unlike Q-Learning, SARSA uses the next action selected by the policy, rather than the optimal action. It still follows the -greedy policy, but since it learns from the executed

policy instead of the best possible action, it is generally more conservative and stable. It uses the action the agent actually takes to update, rather than the optimal action.

C. Deep Q-Network (DQN)

DQN replaces the Q-table with a **neural network**. In the previous algorithms, we used tabular methods for RL. i.e., we stored values in a table for each state or state-action pair. Instead of storing a table of values, we use a neural network to estimate Q(s,a). Hence, Q(s,a) becomes Q(s,w) where w is a set of learnable parameters.

D. Value Iteration

Value Iteration is a model-based algorithm that follows the principles of Dynamic Programming (DP). It updates state values iteratively using the Bellman Optimality Equation:

$$V(s) = \max_a \sum_{s'} P(s'|s,a)[r + \gamma V(s')] \tag{4}$$

where $V(s)$ is the value function and $P(s'|s,a)$ represents the transition probability. Unlike temporal-difference (TD) methods above, Value Iteration does not require interaction with the environment, which makes it significantly efficient for small state spaces.

IV. Experimental Setup:

For each algorithm, we define a set of hyperparameters that control the learning process. Q-Learning and SARSA follow the same approach but differ in how it updates Q-values. The value chosen for each parameter is based on experimentation to ensure that the algorithm converges optimally while maintaining efficiency. The main parameters include the following:

- **Number of Episodes**:-training iterations the agent undergoes.
- **Maximum Steps per Episode:** Max exploration step the agent takes in an episode before termination.
- **Learning Rate (α):** How much new information overrides old knowledge.
- **Discount Factor (γ):** Balances the importance of immediate rewards versus long-term rewards.
- **Exploration Rate (ϵ):** Balance between exploration (trying new actions) and exploitation (leveraging past knowledge)

For DQN, key components include:

- Input: One-hot encoded state representation.
- Fully connected layers with RELU activation.
- Experience Replay: Stores past experiences in a deque buffer.
- Target Network: A second neural network that stabilizes training.
- Loss Function & Optimizer: MSE & Batch Size.

We monitor the performance of each algorithm using reward tracking, decay of exploration rate, and visualization.

V. Observations and Result

Expectedly, Q-learning converges faster due to its aggressive or greedy approach by taking the max value. The model reaches a stable policy with an average reward around 7.5. We noticed some initial high negative rewards that indicate that the agent explored inefficient policies before it stabilized, which is common in off-policy learning methods.

Conversely, SARSA converges slightly slower than Q-Learning but exhibits more stability. The smooth increase noticed in the reward graph shows that it takes fewer risky actions, which means a more consistent learning process. The average reward per thousand episodes follows a similar trend to Q-Learning and settles around 7.3-7.5.

For DQN, it requires less episodes to converge but it gives significantly lower average rewards. Its struggles may be due to function approximation inefficiencies in small, discrete environments. In addition, it requires more computational resources and hyperparameter tuning.

Finally, the Value Iteration algorithm gives rapid convergence and optimal policy computation. It guarantees the best possible performance in our case because of the simple environment.

VI. Comparison of Algorithms

Algorithm	Type	Policy	Convergence	Stability
Q-Learning	Model-free	Greedy	Fast	Less Stable
SARSA	Model-free	ϵ-greedy	Slower	More Stable
DQN	Deep Learning	ϵ-greedy	Slowest	Unstable
Value Iteration	Model-based	Deterministic	Fastest	Most Stable

TABLE I
Comparison of RL Algorithms in the Taxi-v3 Environment.

VII. Conclusion

With this project, we successfully implement the four most popular reinforcement learning algorithms to train an autonomous taxi agent in OpenAI's Taxi-v3 environment. Our results show the strengths and weaknesses of each approach, and we are able to compare and contrast these methods. This provides a solid foundation for future work in deep reinforcement learning for autonomous navigation.