# Frozen lack

“Selected topic in ai project”

Team members:

Marwan eslam

Omar Mohamed

Farid Mohamed

Karim osama

Bodour Mohamed

Nada kamal

# Under supervision of

Eng/ Mariam Mohamed

# Table of content

* Overview
* Objective
* Problem definition
* Reward system
* Project 1: Q-Learning for Frozen Lake (8x8)
* Project 2: DQL-Learning for Frozen Lake (4x4)
* Performance Comparison: Q-Learning vs. DQN
* conclusion

# Overview:

This documentation provides a comprehensive guide to two **Reinforcement Learning (RL) projects** that use **Q-Learning** and **Deep Q-Learning (DQN)** to train an agent for autonomous decision-making in the **FrozenLake-v1** environment from **Gymnasium**.

Both projects are designed to develop an agent that learns to navigate a frozen lake **without falling into holes** and **reaching the goal** using RL techniques.

# Project Objectives:

* Implement **Q-Learning** and **Deep Q-Learning (DQN)** to train an RL agent.
* Compare **tabular Q-learning** with a **neural network-based approach (DQN)**.
* Analyze agent performance through **reward tracking, epsilon decay, and convergence analysis**.
* Demonstrate agent behavior in **deterministic (non-slippery)** and **stochastic (slippery)** environments.

# Problem Definition:

The **Frozen Lake environment** consists of a **grid-based map** where an agent starts at the **bottom-left corner** and must reach the **goal** at the top-right corner while avoiding holes.

* The environment can be either **deterministic (is\_slippery=False)** or **stochastic (is\_slippery=True)**.
* The agent interacts with the environment by selecting one of four possible actions:
  + **Left (L)**
  + **Down (D)**
  + **Right (R)**
  + **Up (U)**

The agent receives **rewards** as follows:

* **+1 for reaching the goal**
* **0 for falling into a hole or stepping on an empty tile**

The agent's goal is to **maximize cumulative rewards** by learning an **optimal policy** for reaching the goal efficiently.

# Reward System:

The agent receives **rewards** based on its movement:

* **+1 for reaching the goal**
* **0 for stepping on an empty tile or falling into a hole**
* **No penalties for incorrect moves**, but the goal is to find the most optimal path.

The objective is to **maximize cumulative rewards** by learning an **optimal policy** that minimizes the number of steps to reach the goal while avoiding hazards.

# Project 1: Q-Learning for Frozen Lake (8x8)

**Overview of Q-Learning**

A screenshot of a video game

Description automatically generated

Q-learning is a **model-free, off-policy reinforcement learning algorithm** that enables an agent to learn an optimal policy through **trial and error**.

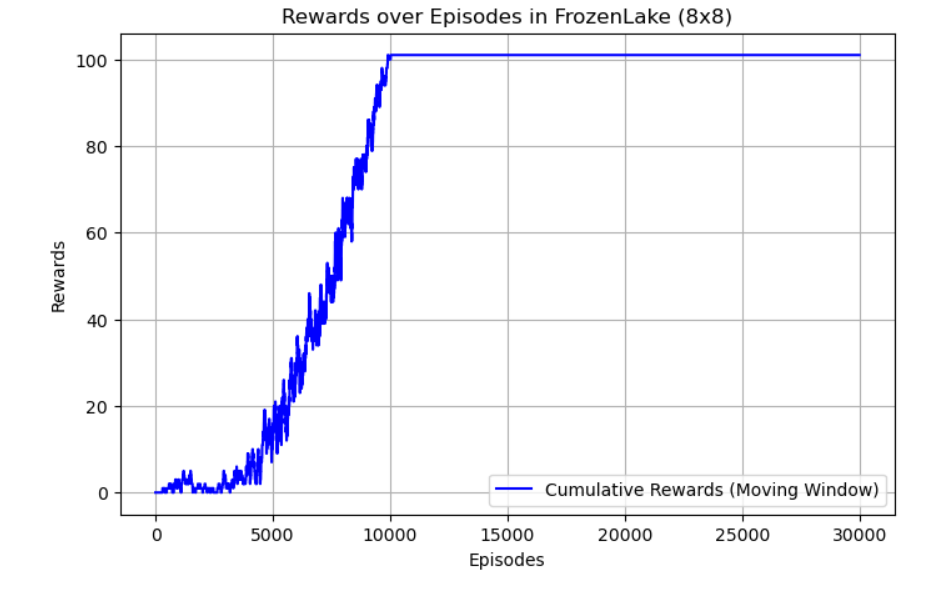
**Core Concept**

* The agent maintains a **Q-table** where each state-action pair has an associated Q-value.
* The Q-value represents the **expected future reward** for taking an action in a given state.
* The agent updates the Q-values iteratively using the **Bellman equation**.

**Steps in Q-Learning:**

1. **Initialize the Q-table**:
   * Create a **2D table** where rows represent **states** and columns represent **actions**.
   * Initialize all Q-values to **zero**.
2. **Choose an action using an epsilon-greedy strategy**:
   * With probability **ε (epsilon)**, choose a **random action** (exploration).
   * With probability **1 - ε**, choose the **best known action** based on Q-values (exploitation).
3. **Take the action and observe the result**:
   * The environment returns **a new state, a reward, and a termination flag**.
4. **Update the Q-table** using the Bellman equation\*\*:
   * Adjust the Q-value based on **the immediate reward and the highest future Q-value**.
5. **Repeat the process for multiple episodes**:
   * Gradually **reduce epsilon** to shift from exploration to exploitation.
   * Train until the Q-table stabilizes and the policy converges.

**Training performance:**



# **Project 2: Deep Q-Learning (DQN) for Frozen Lake (4x4):**

**Overview of Deep Q-Learning (DQN):**

**A screenshot of a game

Description automatically generated**

Deep Q-Learning (DQN) is an extension of Q-learning that **uses a neural network** instead of a Q-table.

* Suitable for **large or continuous state spaces** where tabular Q-learning fails.
* Uses a **Deep Neural Network (DNN)** to approximate Q-values.

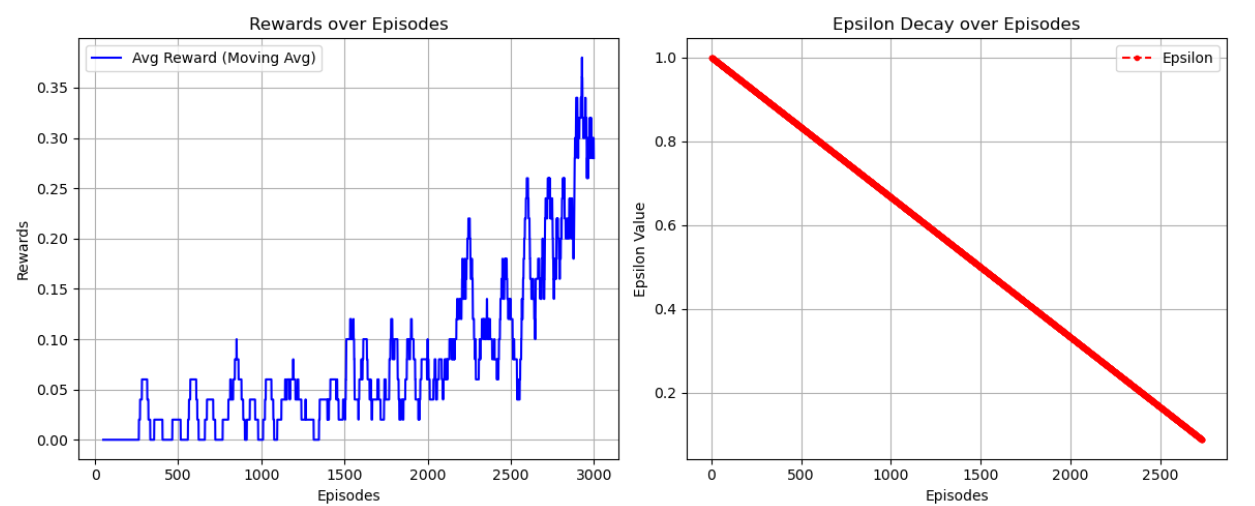
**Core Concept:**

1. **State Representation**:
   * Instead of storing Q-values in a table, the state is encoded as an **input vector**.
2. **Neural Network**:
   * The network takes the state as input and **outputs Q-values** for each action.
3. **Training using Backpropagation**:
   * The network updates Q-values using **Mean Squared Error (MSE)** between predicted and target Q-values.
4. **Experience Replay**:
   * Stores past experiences in a **replay buffer** and samples random batches for training.
5. **Target Network**:
   * A separate **target network** is used to stabilize training by reducing oscillations.

**Steps in DQN:**

1. **Initialize the Deep Q-Network (DQN)** with random weights.
2. **Use an epsilon-greedy strategy** to balance exploration and exploitation.
3. **Take an action and store the experience in memory**.
4. **Train the neural network using mini-batches from the experience replay buffer**.
5. **Periodically update the target network** to stabilize learning.
6. **Repeat the process until convergence**.

**Training performance:**



# Performance Comparison: Q-Learning vs. DQN:

A screenshot of a computer

Description automatically generated

# Conclusion:

Both **Q-Learning and DQN** are effective reinforcement learning methods, but their suitability depends on **environment size and complexity**.

* **For small, discrete environments** → **Q-learning** is simple and efficient.
* **For large environments** → **DQN** provides better scalability.