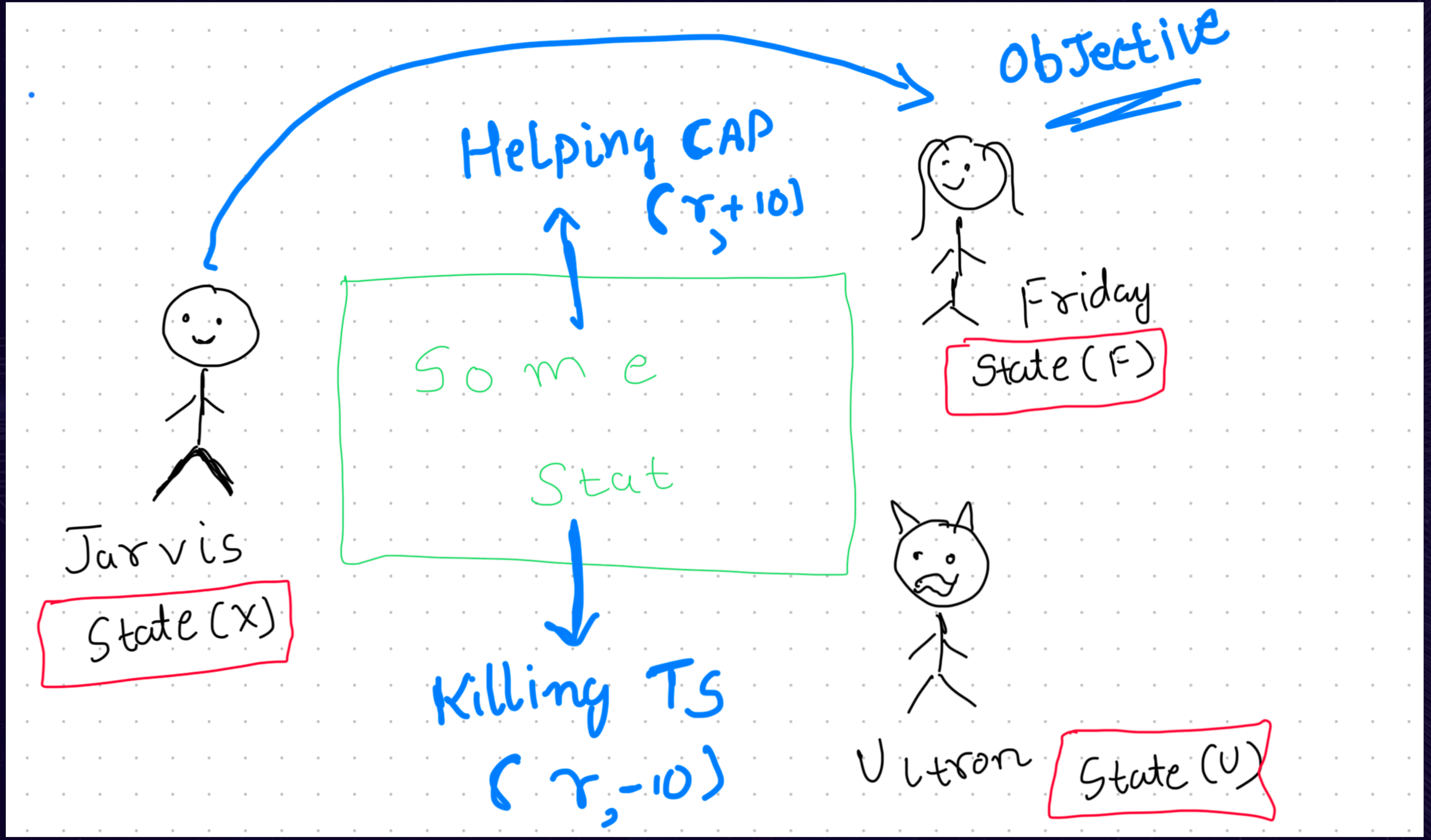
Q - Learning - CHRISTOPHER J.C.H. WATKINS, PETER DAYAN

A Research Paper Presentation

By
Pranathi Etta - 302282085
Sariyu Madagoni - 302291211
Om Nai - 30228559


Objective
Helping CAP
(r3+10)
Some
Stat
Killing TS
(r3-10)
Jarvis
State (X)
Friday
State (F)
Ultron
State (U)

What Is Q-Learning ?

- Q-learning is a model free RL approach (basically works on trial and error approach).
- Providing agent to act optimally in Markovian Domain by virtue of effects of the action taken, regardless of building maps of the domain.
- It can be also refred as an async way of Dynamic Programming.

Key Concepts

Markovian Domains:A Markovian domain is an environment where the future state depends only on the current state and action, adhering to the Markov property.

Reinforcement Learning Basics: Agents learn optimal actions by interacting with the environment and receiving feedback through rewards.

Q-Learning Features: A model-free reinforcement learning method that evaluates actions using immediate rewards and future discounted rewards.

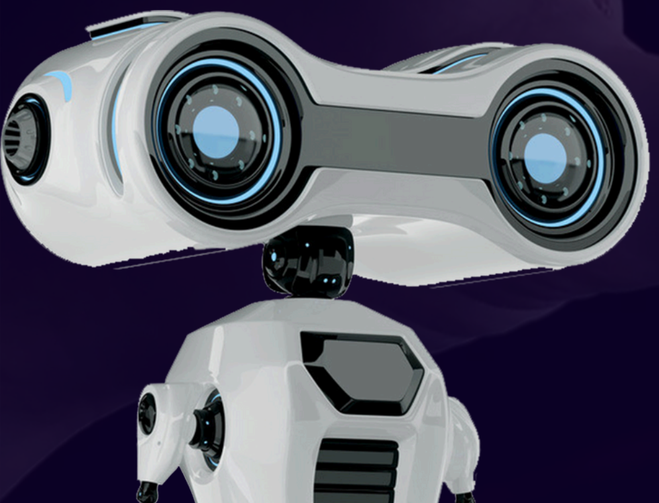
Methodology Adopted

- An agent throughout its lifetime starts from a start state, and makes several transitions from its current state to a next state based on its choice of action and also the environment the agent is interacting in.
- At every step of transition, the agent from a state takes an action, observes a reward from the environment, and then transits to another state. If at any point in time, the agent ends up in one of the terminating states that means there are no further transitions possible.
- This is said to be the completion of an episode.

Methodology Adopted

$$\text{Prob}\,[y_n = y | x_n, a_n] = P_{x_n y}[a_n].$$

Consider a computational agent moving around some discrete, finite world, choosing one from a finite collection of actions at every time step. The world constitutes a **controlled Markov** process with the agent as a controller. At step **n**, the agent is equipped to register the state **x n (belongs to X)** of the world, a can choose its action **an** (belongs to Q) accordingly. The agent receives a probabilistic reward **rn**, whose mean value **Rxn (an)** depends only on the state and action, and the state of the world changes probabilistically to yn according to the law



Convergence Proof

- The convergence proof of Q-Learning ensures that the algorithm reliably learns the optimal Q-values (Q*) for all state-action pairs under certain conditions.
- The proof is based on the Action Replay Process (AFIP), a theoretical framework that simulates the learning process as a controlled Markov process.
- The Action Relay Process in RL refers to the sequence of choosing an action, executing it, and passing this action through systems (such as Q-values or environmentt interactions) to receive feedback and update accordingly.

Convergence Proof

The key steps include:

Conditions for Convergence:

- All state-action pairs are visited infinitely often.
- Learning rates (alpha) decay appropriately over time.

Core Theorem:

- Under the above conditions, the Q-values updated using the Q-Learning formula converge to the true optimal Q-values with probability 1.

Convergence Proof

Lemmas Supporting the Proof:

- Lemma A: Q-values in the AFIP are optimal for each level of learning.
- Lemma B: The AFIP process mimics the real Markov process, and its rewards and transitions converge to the true environment dynamics.

Proof Highlights:

- Uses backward induction and stochastic convergence to show that errors in Q-values diminish over time.
- The presence of a discount factor (γ) ensures bounded value estimation and stability in the learning process.

Convergence Proof

Outcome:

Guarantees that Q-Learning identifies the optimal policy (π^*) for maximizing cumulative discounted rewards.

The Convergence of Q-Learning

- **Importance of Action Relay in Convergence:**
  a. State-Action Pair Updates: Continuous relaying of actions and state-reward outcomes ensures that the Q-values are refined over time, leading to convergence.
  b. Exploration vs. Exploitation: The balance of choosing exploratory and exploitative actions through the relay process influences how effectively Q-values are learned, ensuring convergence as exploration decreases over time.

Methods for Determining Q-Values

There are two methods for determining Q-values:

- Temporal Difference: Temporal Difference (TD) Learning is a model-free reinforcement learning (RL) algorithm that learns by bootstrapping, meaning it updates estimates of future rewards based on incomplete information, using the difference between predicted and actual rewards over time.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \gamma a' \max Q(s',a') - Q(s,a)]$$

Bellman’s Equation

The Bellman Equation is a fundamental recursive equation in dynamic programming and reinforcement learning that expresses the relationship between the value of a state and the values of its successor states. It is used to compute the optimal value function, which represents the maximum expected reward that can be achieved from any given state, following a certain policy or optimal strategy.

$$Q(s,a)=R(s,a)+\gamma \max_a Q(s',a)$$

Psuedo Code

1. Initialize Q-Learning table with all zeros or with small random values
   - Size of Q-Lerning Table <- [number of states] x [number of actions]
   - Q[state][action] represents the value of taking action in that state

2. For each episode(epoch):
   - Initialize the starting state (i.e. usually current state)

3. Repeat until the episode(epoch) is completed:
   a. Choose an action in the current state:
      - Use an exploration strategy, like (here ε-greedy):
        - With probability ε, choose a random action (exploration)
        - With probability 1-ε, choose the action with the highest Q-value (exploitation)

   b. Take the chosen action and observe the reward (reward) and the new state (new_state)

Psuedo Code

1. c. Update the Q-value for the current state and action:
   - Q[current_state][action] = Q[current_state][action] + α * (reward + γ * max(Q[new_state][a]) - Q[current_state][action])
   where:
   - α is the learning rate (controls how much the Q-values are updated)
   - γ is the discount factor (controls the importance of future rewards)
   - max(Q[new_state][a]) is the maximum Q-value for the next state across all actions a
   d. Update the current_state to the new_state

4. End of episode
5. Repeat until convergence or a stopping criterion is met (e.g., a fixed number of episodes or no further significant updates in Q-values)

Implementation

Step 1: Define the Environment

```
import numpy as np

# Define the environment
n_states = 16 # Number of states in the grid world
n_actions = 4 # Number of possible actions (up, down, left,
right)
goal_state = 15 # Goal state

# Initialize Q-table with zeros
Q_table = np.zeros((n_states, n_actions))
```

Implementation

Step 2: Set Hyperparameters

```
# Define parameters
learning_rate = 0.8
discount_factor = 0.95
exploration_prob = 0.2
epochs = 1000
```

Implementation

Step 3: Implement the Q-Learning Algorithm

```
# Q-learning algorithm
for epoch in range(epochs):
 current_state = np.random.randint(0, n_states) # Start from a
random state

 while current_state != goal_state:
 # Choose action with epsilon-greedy strategy
 if np.random.rand() < exploration_prob:
 action = np.random.randint(0, n_actions) # Explore
 else:
 action = np.argmax(Q_table[current_state]) # Exploit
```

Implementation

Step 3: Implement the Q-Learning Algorithm

```
# Simulate the environment (move to the next state)
 # For simplicity, move to the next state
 next_state = (current_state + 1) % n_states

 # Define a simple reward function (1 if the goal state is reached, 0
otherwise)
 reward = 1 if next_state == goal_state else 0

 # Update Q-value using the Q-learning update rule
 Q_table[current_state, action] += learning_rate * \
 (reward + discount_factor *
 np.max(Q_table[next_state]) - Q_table[current_state, action])

 current_state = next_state # Move to the next state
```

Implementation

Step 4: Output the Learned Q-Tablet

```
# After training, the Q-table represents the learned Q-values
print("Learned Q-table:")
print(Q_table)
```

Extensions

Non-Discounted Markov Processes:
For γ=1\gamma = 1γ=1, Q-Learning adapts to environments with absorbing goal states, ensuring long-term rewards are maximized.
Applications: Maze-solving and goal-oriented tasks.
Updating Multiple Q-Values:
Allows batch updates or parallel Q-value adjustments, improving learning efficiency, especially in replay-based methods like Deep Q-Learning.
Generalized Algorithms:
Combines Q-Learning with methods like Temporal Difference (TD(λ)TD(\lambda)TD(λ)) for smoother updates and faster convergence.
Memory-Based Enhancements:
Experience replay reuses past episodes to improve sample efficiency and balance exploration vs. exploitation.

Practical Applications:

- Robotics: Learning behaviors for automated systems without pre-programmed rules.
- Game AI: Strategy learning for agents in competitive games.
- Industrial Automation: Optimizing decision-making in dynamic systems like traffic control or inventory management.
- Finance: Portfolio optimization and real-time decision-making in trading systems.

Challenges in Q-Learning

Requires all state-action pairs to be visited infinitely often for convergence.

Learning rates (an\alpha_nan) must decay properly to balance new learning and past knowledge.

Initial Q-values can bias learning speed and trajectory.

Limited to discrete state-action spaces; struggles with large or continuous domains.

Effective exploration vs. exploitation remains challenging.

Difficulties arise in dynamic environments with non-stationary transitions or rewards.

Sparse or delayed rewards can slow learning significantly.

References

Slides Reference :-

- GFG :- https://www.geeksforgeeks.org/q-learning-in-python/
- Wekepedia :- https://en.wikipedia.org/wiki/Q-learning
- Daniel Russo :- https://djrusso.github.io/RLCourse/slides/week4.pdf

Design Slides Used :-

- Canva free ppt presentation


Thank You!