11. Research Analysis | Isaac Mobe | HDB 212-C002-0027/2022

Question:

Pick one basic AI paper (on **Q-learning**, **Transformers**, or **GANs**). Try reproducing the main idea using simplified code or available notebooks. Summarize the idea, run basic experiments, and explain what your group learned.

Chosen paper/concept: *Q-Learning* (Watkins, 1989) – is a simple way for agents to learn how to act optimally in controlled **Markovian domains**. Markovian domains refer to environments that satisfy the **Markov property** which is a foundational concept in reinforcement learning and dynamic programming. Some of the **Markovian Domains** include the **Markov property** that states that, the future state of the system depends only on the current state and action not on the sequence of events that preceded it. **Controlled Markovian Domains** which are environments where an agent can influence transitions between states through its actions, and those transitions follow probabilistic rules defined by a Markov Decision Process. (MDP).

Q-learning is designed to operate in **controlled Markovian domains**, meaning it assumes the environment behaves like an MDP. The algorithm incrementally learns the optimal action-value function by evaluating and updating estimates of the expected rewards for taking certain action in certain states.

It amounts to an incremental method for dynamic programming which imposes limited computational demands. It works by successively improving its evaluations of the quality of particular actions at particular states.

Understanding Q-learning means grasping one of the most elegant ideas in artificial intelligence: how an agent can learn optimal behavior through trial and error, without any teacher telling it what to do. Let me guide you through reproducing this foundational algorithm that launched modern reinforcement learning.

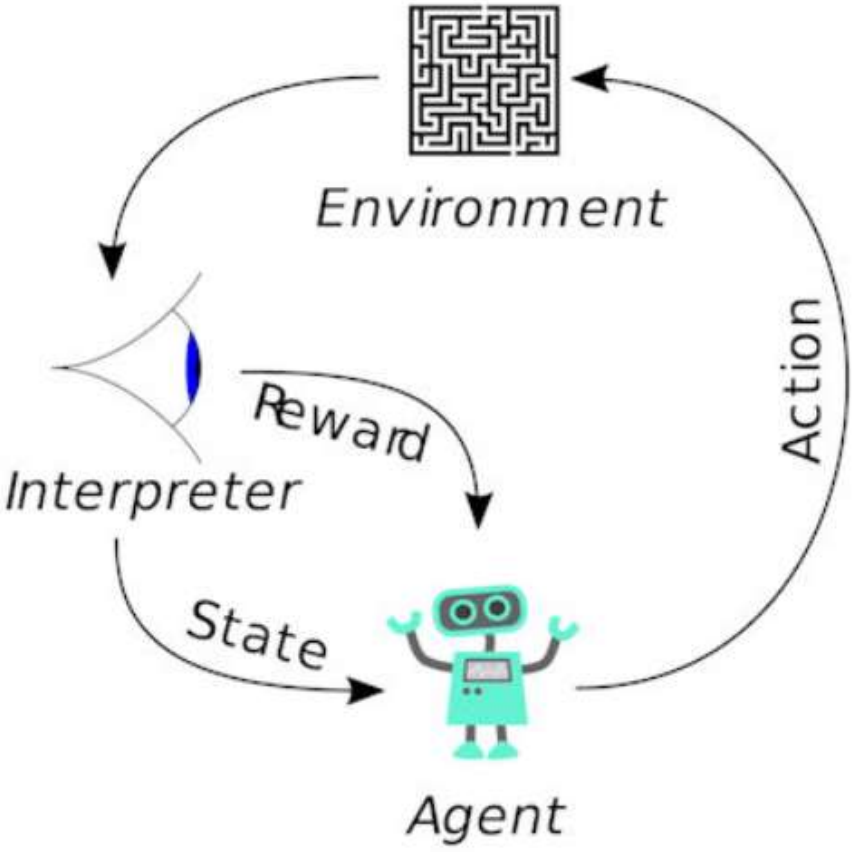
The core insight behind Q-learning, developed by Chris Watkins in 1989, addresses a fundamental question: how can an agent learn the value of taking specific actions in specific situations when the consequences of those actions only become clear over time? Think of learning to play chess - the value of moving your queen isn't immediately obvious; it depends on how the entire game plays out.

Q-learning solves this through a brilliant mathematical framework. The "Q" stands for "quality" - specifically, the quality or expected future reward of taking action 'a' in state 's'. The algorithm learns a Q-table that maps every possible state-action pair to its expected long-term value. The magic happens in the update rule that learns from temporal difference errors.

More concept analysis:

Q-learning is a reinforcement learning technique used in machine learning. The goal of Q-Learning is to learn a policy, which tells an agent which action to take under which circumstances. It does not require a model of the environment and can handle problems with stochastic transitions and rewards, without requiring adaptations.

Q-learning was first invented in Prof. Watkins' Ph.D. thesis "Learning from Delayed Rewards", which introduced a model of reinforcement learning (learning from rewards and punishments) as incrementally optimizing control of a Markov Decision Process (MDP), and proposed a new algorithm – which was dubbed "Q-learning" – that could in principle learn optimal control directly without modelling the transition probabilities or expected rewards of the MDP. The first rigorous proof of convergence was in Watkins and Dayan (1992). These innovations helped to stimulate much subsequent research in reinforcement learning. The notion that animals, or 'learning agents' inhabit a MDP, or a POMDP, and that learning consists of finding an optimal policy, has been dominant in reinforcement learning research since, and perhaps these basic assumptions have not been sufficiently examined.



Reproduction:

Q-Learning is like teaching a robot to navigate by letting it learn from experience. The robot doesn't know the rules initially, but it learns by trying different actions and seeing what happens.

Agent: The learner (blue square)

Goal: What we want to reach (+10 reward)

Obstacles: Things to avoid (-5 reward)

The Q-Learning Formula:

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

Q (s, a): Quality of action 'a' in state 's'

α: Learning rate (how fast we learn)

γ: Discount factor (how much we care about future rewards)

r: Immediate reward

“Rat in a Maze” scenario,

```python
import random

# Maze layout: 4x4 grid
# 0 = empty cell, 1 = wall, 2 = cheese (goal)
MAZE = [
    [0, 0, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 2],
]

# Actions: up, down, left, right mapped to coordinate changes
ACTIONS = {
    0: (-1, 0),  # UP
    1: (1, 0),   # DOWN
    2: (0, -1),  # LEFT
    3: (0, 1),   # RIGHT
}

START = (0, 0)  # Rat starts top-left
GOAL = (3, 3)   # Cheese at bottom-right

# Q-learning parameters
alpha = 0.1    # learning rate
gamma = 0.9    # discount factor
epsilon = 0.9  # exploration rate (start high)
episodes = 500 # number of training episodes

# Initialize Q-table: for each cell, store Q-values for 4 actions
Q = { (r, c): [0.0, 0.0, 0.0, 0.0] for r in range(4) for c in range(4) }

def get_valid_actions(pos):
    """Return list of actions that don't hit walls or go outside maze."""
    valid = []
    r, c = pos
    for action, (dr, dc) in ACTIONS.items():
        nr, nc = r + dr, c + dc
        if 0 <= nr < 4 and 0 <= nc < 4 and MAZE[nr][nc] != 1:
```

```
            valid.append(action)
    return valid

def step(pos, action):
    """
    Move rat according to action.
    Returns: (new_position, reward, done)
    Reward: -1 per step, +10 if cheese found.
    """
    dr, dc = ACTIONS[action]
    new_pos = (pos[0] + dr, pos[1] + dc)
    if MAZE[new_pos[0]][new_pos[1]] == 2:
        return new_pos, 10, True  # Found cheese!
    else:
        return new_pos, -1, False  # Normal step cost

# Training loop
for ep in range(episodes):
    state = START
    done = False
    # Decay epsilon linearly over episodes (explore less over time)
    eps = epsilon * (1 - ep / episodes)

    while not done:
        # Choose action: explore or exploit
        if random.random() < eps:
            action = random.choice(get_valid_actions(state))  # Explore
        else:
            # Exploit: pick best Q-value action among valid moves
            q_vals = Q[state]
            valid_actions = get_valid_actions(state)
            max_q = max(q_vals[a] for a in valid_actions)
            best_actions = [a for a in valid_actions if q_vals[a] == max_q]
            action = random.choice(best_actions)

        # Take action, observe reward and next state
        next_state, reward, done = step(state, action)

        # Q-learning update rule
        future_q = 0 if done else max(Q[next_state][a] for a in
get_valid_actions(next_state))
        td_target = reward + gamma * future_q
        td_error = td_target - Q[state][action]
        Q[state][action] += alpha * td_error
```

```
        # Move to next state
        state = next_state

# Print learned policy as arrows
print("Learned policy (↑↓←→) for each cell:")
for r in range(4):
    line = ""
    for c in range(4):
        if MAZE[r][c] == 1:
            line += " #  "  # Wall
        elif (r, c) == GOAL:
            line += " C  "  # Cheese
        else:
            best_action = max(range(4), key=lambda a: Q[(r, c)][a])
            symbols = ['↑', '↓', '←', '→']
            line += f" {symbols[best_action]}  "
    print(line)
```

So, what's Happening

1. **Environment Setup:**
    - The maze is a 4×4 grid with empty cells (0), walls (1), and a goal cell with cheese (2).
    - The rat starts at the top-left corner (0,0) and wants to reach the cheese at (3,3).
    - The rat can move up, down, left, or right, but cannot move into walls or outside the grid.
2. **Q-Table Initialization:**
    - We create a dictionary Q where each key is a cell (row, col).
    - Each value is a list of 4 Q-values, one for each action (up, down, left, right), all initialized to 0.
    - These Q-values represent the "quality" or expected future reward of taking that action from that state.
3. **Training Loop (Over Episodes):**
    - For each episode, the rat starts at the start position.
    - We decay the exploration rate epsilon linearly so the rat explores a lot early on and exploits learned knowledge later.
    - The rat repeatedly:
        - Chooses an action using an **ε-greedy policy**:
            - With probability ε, it picks a random valid action (exploration).
            - Otherwise, it picks the action with the highest Q-value among valid moves (exploitation).
        - Takes the action, observes the reward and next state.
        - Updates the Q-value for the (state, action) pair using the **Q-learning update rule:**

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

where:

- **Alpha**: is the learning rate (how much new info overrides old),
- **Gamma**: is the discount factor (importance of future rewards),
- **r**: is the immediate reward,
- **s'**: is the next state,
- **a'**: ranges over possible actions in **s'**.

- Moves to the next state.

- The episode ends when the rat reaches the cheese.

4. **After Training:**
   - We print the learned policy by showing arrows indicating the best action (highest Q-value) from each cell.
   - Walls are shown as #, the cheese as C, and arrows show the rat's learned path.

How Q-Learning Works in This Example

- **Q-values start at zero:** The rat initially has no knowledge of which moves are good.
- **Exploration vs. Exploitation:** Early on, the rat tries many random moves to learn about the maze. Over time, it uses what it learned to pick the best moves.
- **Temporal Difference Learning:** After each move, the rat updates its estimate of the value of that move based on the immediate reward and the best future reward it expects.
- **Value Propagation:** The reward of +10 at the cheese "propagates backward" through the maze as the rat updates Q-values, so states closer to the cheese get higher Q-values.
- **Convergence:** Given enough episodes and proper parameters, the Q-values converge to the true expected rewards, and the rat learns an optimal or near-optimal path to the cheese.

Conclusion:

In this simple "Rat in a Maze" Q-learning example, we saw how an agent with no prior knowledge—can gradually discover an efficient path to its goal purely by trial and error. Through the ε-greedy strategy it balances exploration and exploitation, and the temporal-difference updates allow reward information to ripple backward from the goal, shaping the Q-values across the grid. Over many episodes, these Q-values converge so that the rat reliably follows the learned

policy represented by arrows to the cheese. This minimal reproduction captures the core of Watkins' Q-learning: a lightweight, model-free algorithm that requires only observed transitions and rewards to learn optimal behavior in any Markovian environment.

References:

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards* (Doctoral dissertation). University of Cambridge.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3–4), 279–292.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.