The City College of New York

CSC 59866-E: Senior Project I

AI Agents for Decision Making in the Real World

By Saptarashmi Bandyopadhyay
Email: sbandyopadhyay@ccny.cuny.edu, sbandyopadhyay@gc.cuny.edu
Assistant Professor of Computer Science
City College of New York and Graduate Center at the City University of New York

February 4, 2026 CSC 59866

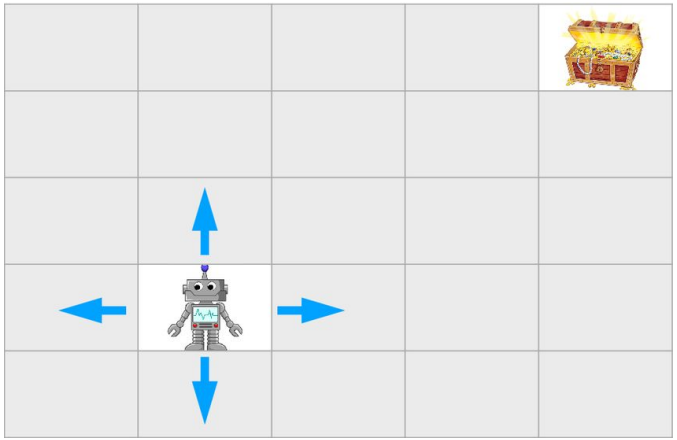
Single-Agent AI, Multi-Agent AI, and Complexity

Single-Agent AI

GridWorld Example

How would you model this environment for an AI Agent trying to find the treasure? You need:

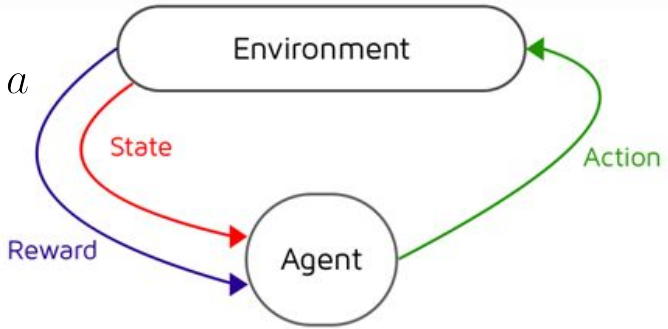
- The possible *states* of the world
- *Actions* the agent can take
- How the state *changes* when an action is taken
- What the *reward* is for reaching / getting closer to the treasure
- How much value is in getting the treasure *now* versus far in the future



Single AI Agents

Definition: A Markov Decision Process (MDP) is a tuple (S, A, T, R, γ)

- **State (S):** The agent's location on the Grid (x, y) .
- **Action (A):** Up, Down, Left, Right governed by $\pi(s) \rightarrow a$
- **Transition (T):** . (e.g., If I move "Right", 90% chance I move right, 10% chance I slip) $P(s'|s, a)$
- **Reward (R):** . (e.g., +100 for Treasure).
- **Discount Factor** (γ) : A number between 0 and 1 that makes short-term rewards matter more than long-term rewards.



The Goal: Find a policy that maximizes expected return $R(s, a)$

Learning what Actions to Take

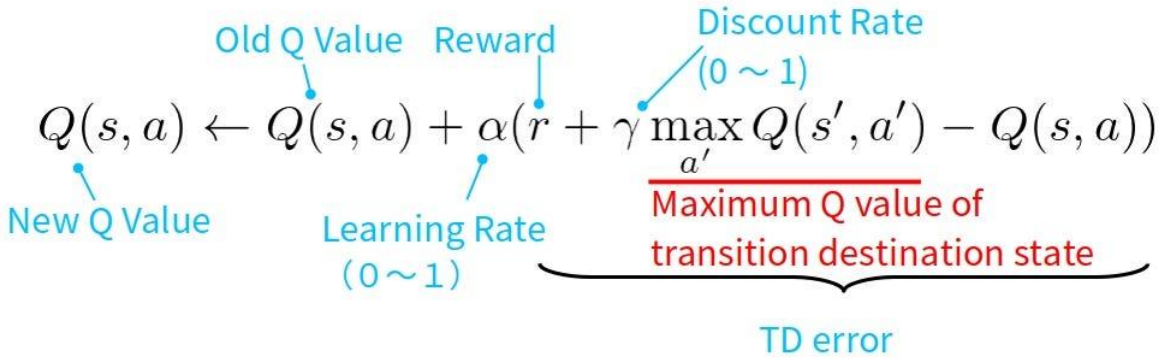
In simple environments like our GridWorld example, we can memorize the optimal actions to take at each state, assigning each state-action pair a number which we call the **Q-value**.

Complexity Check:

- Size of Table = $|S| \times |A|$
- *GridWorld Example:* 10×10 grid = 100 states. 4 actions. Table size = 400 entries.

A 100 x 100 GridWorld example could trivially be solved.

Bellman Equation

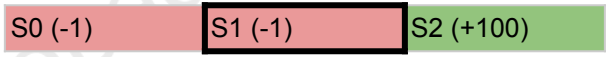


The Bellman Equation describes how the agent updates the Q-values it learns for each state-action pair

- Learning rate α describes the size of the change
- Discount factor γ is a number between 0 and 1, and is used to make immediate rewards matter more than future rewards.

Example:

Three States (S0, S1, S2), Two Actions (Move Left, Move Right)



State	Action: Left	Action: Right
S0	0	0
S1	0	0
S2	0	0

Our Learning Parameters

- Learning Rate (α) = 0.1: How much we trust the new information.
- Discount Factor (γ) = 0.9: How much we value future rewards.

Suppose the Agent starts at S1 and has to make a decision on whether it goes to S0 or S2 from S1!

The table indicates initial Q values before Bellman equation is applied

How do our Q-values change if we move right?

Example:

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

$$\text{New } Q(S_1, \text{Right}) \leftarrow 0 + 0.1 * [100 + 0.9 * 0 - 0] = 10$$

S0 (-1)	S1 (-1)	S2 (+100)

State	Action: Left	Action: Right
S0	0	0
S1	0	0
S2	0	0

State	Action: Left	Action: Right
S0	0	0
S1	0	**10**
S2	0	0

Our Learning Parameters

- Learning Rate (α) = 0.1: How much we trust the new information.
- Discount Factor (γ) = 0.9: How much we value future rewards.

Reset to S1. What if we move left this time?

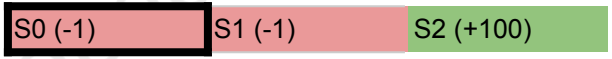
Example:

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

$$\text{New } Q(S_1, \text{Left}) \leftarrow 0 + 0.1 * [-1 + 0.9 * 0 - 0] = -0.1$$

Our Learning Parameters

- Learning Rate (α) = 0.1: How much we trust the new information.
- Discount Factor (γ) = 0.9: How much we value future rewards.
- Q value in going right is higher, favoring that decision



State	Action: Left	Action: Right
S0	0	0
S1	0	10
S2	0	0

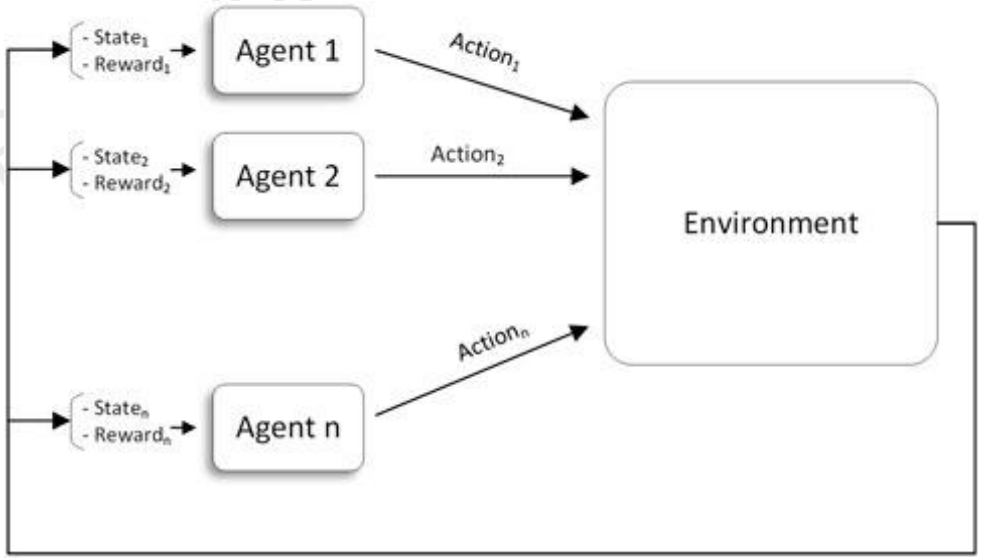
State	Action: Left	Action: Right
S0	0	0
S1	**-0.1**	10
S2	0	0

Note: In the real world, we would probably have a much more complex environment, and use a neural network instead of a table to calculate Q-values! (**Deep Q-Learning**)

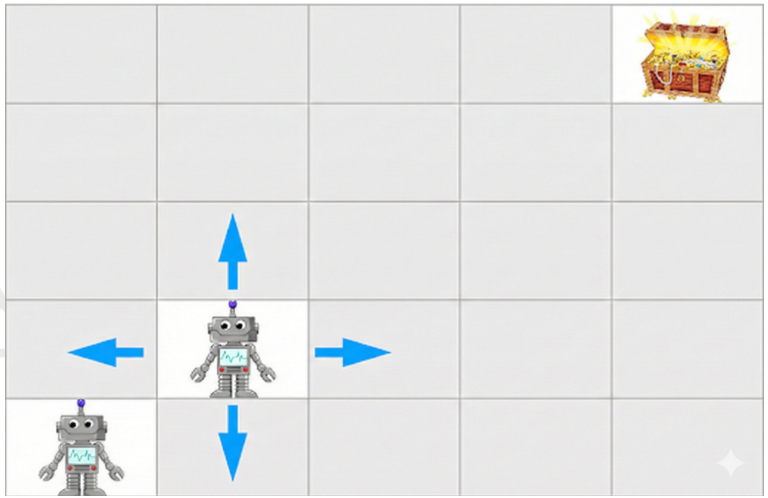
Multi-Agent AI

From Single-Agent to Multi-Agent

- In Single Agent AI, the world only changes based on *my* action (and random noise).
- **In Multi-Agent AI:** The world changes based on *my* action AND *your* action.
- **The Non-Stationarity Problem:**
  - If Agent B changes its policy, the environment dynamics change for Agent A.



Multi-Agent AI



The Conflict:

- Robot A wants to go to $(5, 5)$
- Robot B wants to go to $(5, 5)$
- They cannot occupy the same square!

New Definition: A Markov Game (or Stochastic Game) is $(N, S, A_1 ... A_n, T, R_1 ... R_n)$

- N: Number of agents
- **State (S):** The *Joint* State. Now describes (x_A, y_A) AND (x_B, y_B).
- **Action (A):** The *Joint* Action $\mathbf{u} = (a_1, ...a_n)$.

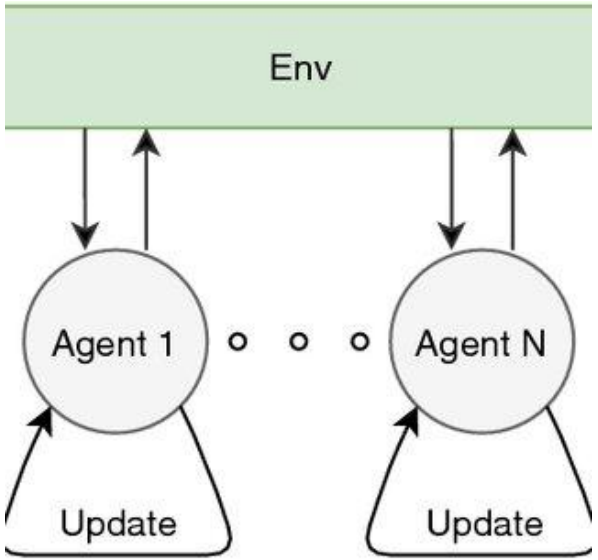
In this system, we need *new* algorithms to help agents coordinate and interact with each other!

Independent Q-Learning (IQL)

In **Independent Q-Learning**, each agent makes decisions as if it were the sole decision-maker.

This method is very easy to implement, but it lacks when it comes to true coordination.

Each agent uses normal Q-learning *independently*, considering the other agents simply just as part of the state.



Value Decomposition Networks (VDN)

A simple way to encourage coordination among agents is for each state, take the actions for each agent, and sum up their Q-values!

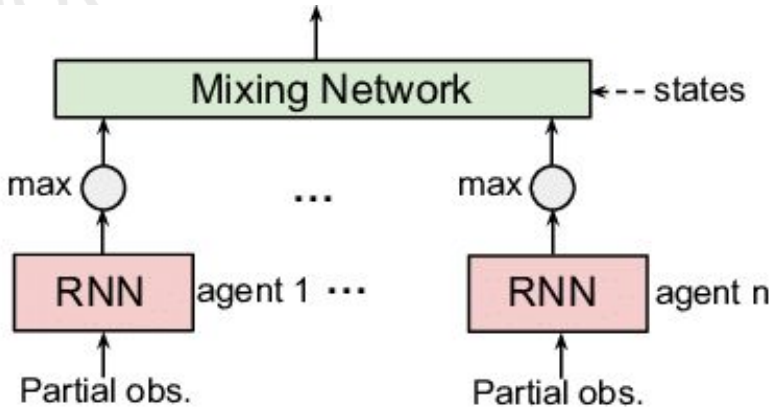
Architectures that employ this are known as **Value Decomposition Networks.**

$$Q_{\text{tot}}(s, a) = \sum_i Q_i(s, a_i)$$

This is a big step up for coordination compared to IQL, but it is only possible to create rewards that are *linear* with the agents.

QMIX

In **QMIX**, we create more possibilities for cooperation by using a *mixing network*, which learns the best way to combine each agent's Q-values.



This allows different agent types to cooperate with each other more flexibly!

Joint Action Vector

$$Q_{\text{tot}}(s, \mathbf{a}) = f_s(Q_1(s^1, a^1), \ldots, Q_n(s^n, a^n))$$

Total Q-value

Monotonic Mixing Network

Complexity Analysis

State-Action Complexity

If we try to build a Q-Table for Multi-Agent GridWorld, how big is it?

Single Agent (10 x 10 grid):

- States: 100.
- Table Size: 100 x 4 = 400.

Multi-Agent (2 Agents, 10 x 10 grid):

- State is now (x_1, y_1, x_2, y_2) .
- States: 100 x 100 = 10,000.
- Actions: Agent 1 has 4, Agent 2 has 4. Joint Actions = 4 x 4 = 16.
- Table Size: 10,000 x 16 = 160,000.

Space Complexity $\approx O(|S|^N \times |A|^N)$.
It is **Exponential** with respect to the number of agents N.

The Curse of Dimensionality in Real-World Agents

Connected Autonomous Vehicles

- Imagine a 4-way intersection.
- $N = 20$ cars.
- $S =$ (Position, Velocity) for each car.
- $A =$ (Accelerate, Brake, Turn) for each car.

The Math: $|A|^N = 3^{20} \approx 3.4$ Billion joint actions *per time step*.
The Research Gap: We cannot solve this with tables. We cannot even solve this with basic independent neural networks (due to non-stationarity).

We can, however, solve it with clever training methods, such as MAPPO!

Time Complexity: Single vs. Multi-Agent

Single Agent: Q-learning converges to the optimal policy (proven).

Multi-Agent:

- Agents are chasing a moving target.
- Often fails to converge (oscillates) or converges to a suboptimal Nash Equilibrium (a state where no individual agent can improve if other agents remain the same).

When evaluating your projects, Convergence Speed (e.g. Sample Efficiency) is a critical metric (as seen in PPO and MAPPO).

Questions?