The City College of New York

CSC 36000:
Modern Distributed Computing *with AI Agents*

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

Today's Lecture

Single-Agent Q-Learning

Multi-Agent Q-Learning

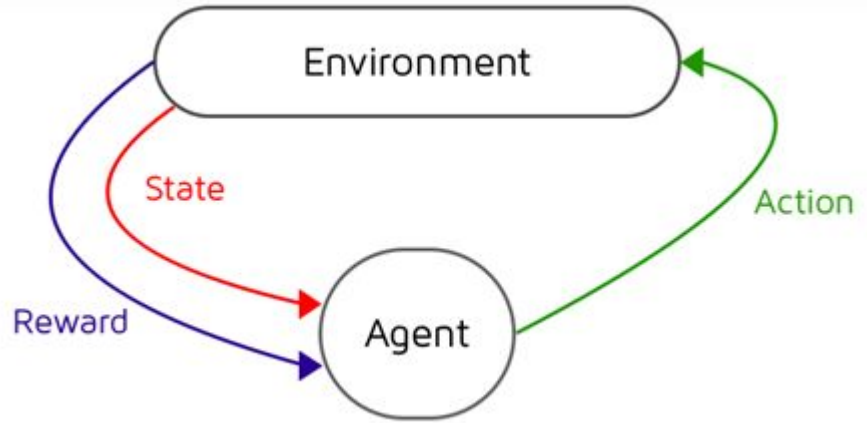
- Independent Q-Learning
- Value Decomposition Networks
- QMIX

Single AI Agents

We usually model the world (“environment”) of an Agent as something called a **Markov Decision Process** (MDP).

An MDP Consists of:

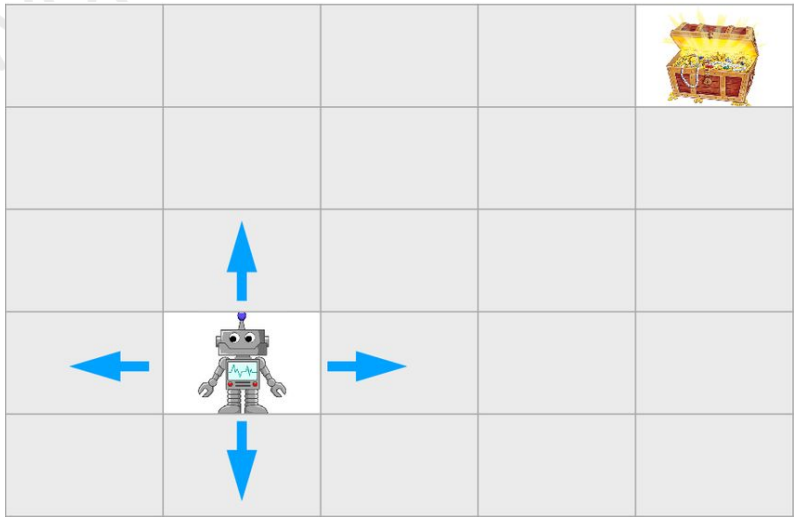
- States **S**
- Actions **A**
- Transitions **T**
- Rewards **R**



GridWorld Example

How would you model this environment as an MDP?

- States?
  - {Robot: (1,1). Treasure: (5,5)}
- Actions?
  - {Up, Down, Left, Right}
- Transitions?
  - {The “go right action” will take the robot one square to the right 100% of the time}
- Rewards?
  - {+100 when the robot reaches the treasure}



Learning what Actions to Take

How does an agent learn what actions to take to get the highest expected cumulative reward?

Many, many methods

- Q-Learning
- REINFORCE
- PPO
- GRPO
- …

We'll focus on a single algorithm called **Q-Learning.**

Q-Learning

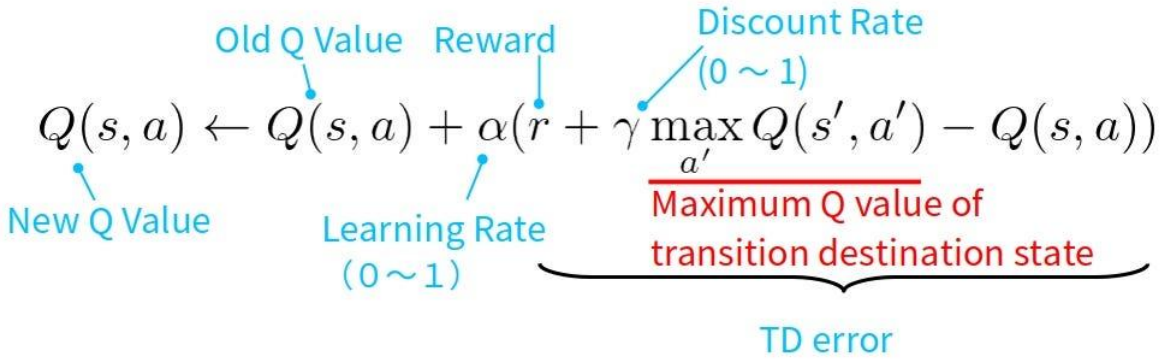
A **Q-value** is how valuable it is to take a particular **action** at a particular **state**, and usually consists of the two paired together like (s,a)

Example:

- Good action at state s_1 : (s_1, a_1) = 5
- Bad action at state s_2: (s_1, a_2) = -1

Q-learning is the process of learning Q-values for different state-action pairs, so that we can eventually make the best decisions by taking the actions with the highest Q-values!

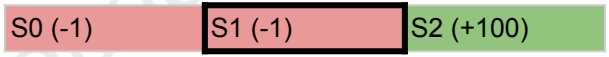
Bellman Equation



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:



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

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

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.

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

Example:

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

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

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

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

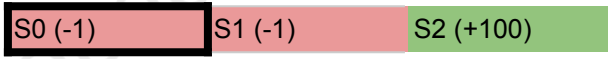
State	Action: Left	Action: Right
S0	0	0
S1	0	**10**
S2	0	0

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:



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

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

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

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.

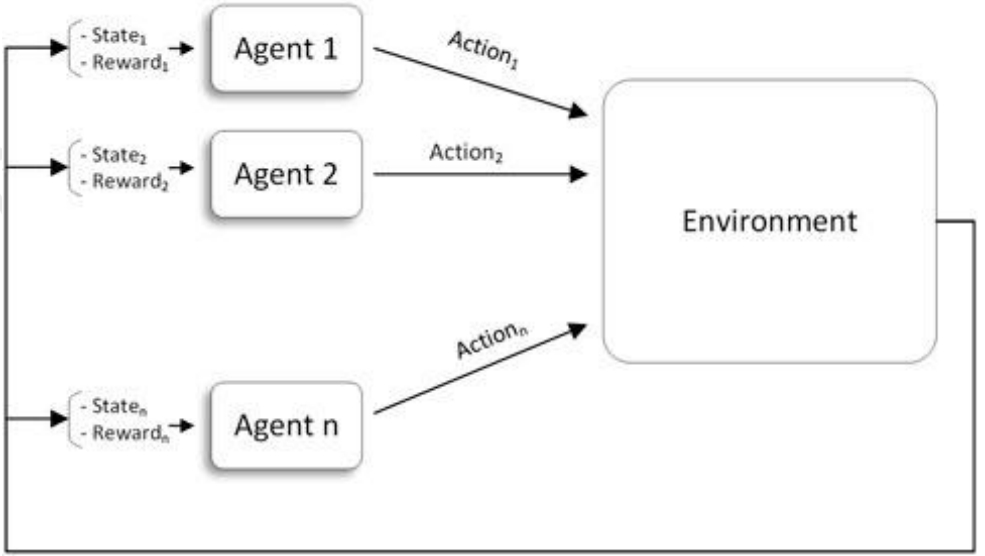
Note: In the real world, we would probably use a neural network instead of a table to calculate Q-values! (**Deep Q-Learning**)

Multi-Agent Q-Learning

From Single-Agent to Multi-Agent

Adding more agents significantly complicates the picture, even if the agents have a common goal!

Each agent has its own actions, and maybe even its own rewards.

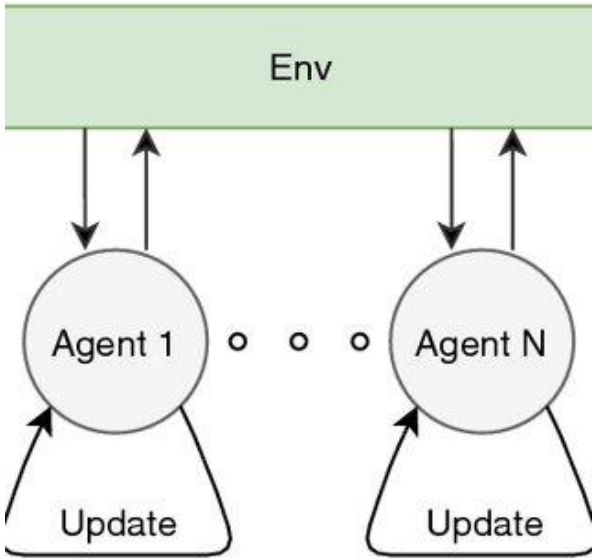


Independent Q-Learning (IQL)

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.

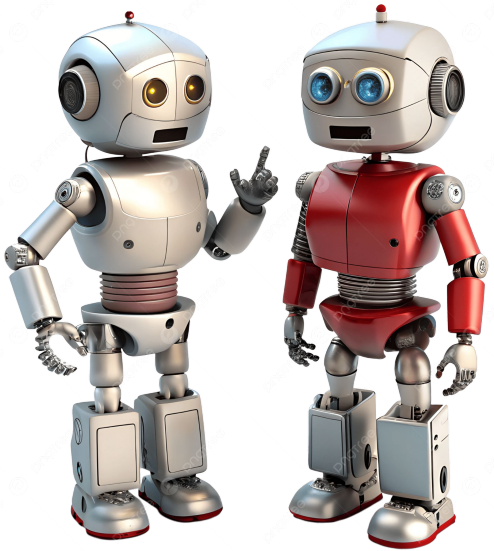
Each agent uses normal Q-learning *independently*, considering the other agents simply just as part of the state.



How do we get agents to coordinate?

We need to do more in order to get the agents to work together!

What if we make it so we don't just try to maximize the *individual* cumulative rewards, but also the *total* cumulative reward?



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.**

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

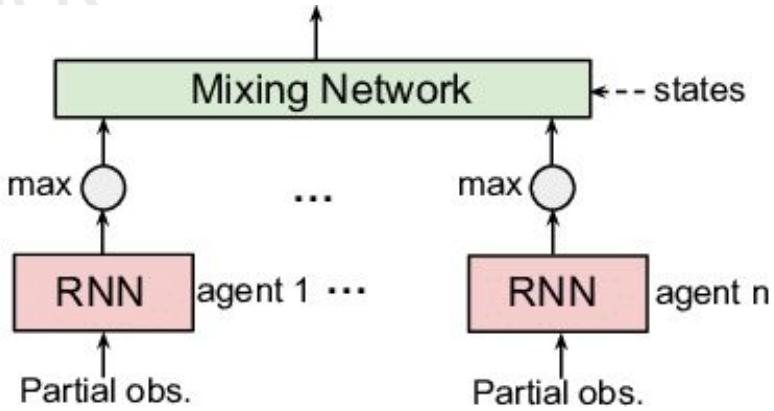
This is a big step up for coordination compared to IQL, but it is only possible to create rewards that are *linear* with the agents.

When the agents are different for each other, this doesn't work as well!

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

Questions?