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

March 18, 2026 CSC 59866

Evolutionary Learning, Mean-Field Learning, Self-play for self-improvement and Continual Learning: Communication paradigms for AI Agents to Adapt in Real-Time Decentralized AI: When and How to Scale AI Agents.

Logistics and Motivation

Recall Lecture 14: We discussed Game Theory, Nash Equilibria, and how algorithms like CFR help small groups of agents find optimal competitive or cooperative strategies.

The New Problem: What happens when we scale from 2 agents playing Poker to 10,000 autonomous vehicles navigating a smart city? The math from Lecture 14 completely breaks down.

Today's Agenda

When to Scale: The Curse of Dimensionality in massive multi-agent systems.

Mean-Field Learning: Approximating the crowd mathematically, Subsampling a Mean-Field environment.

Evolutionary Learning: Using Meta-Black-Box optimization to discover algorithms.

Continual Learning & Self-Play: Adapting in real-time without forgetting.

Communication Paradigms: How decentralized agents share data.

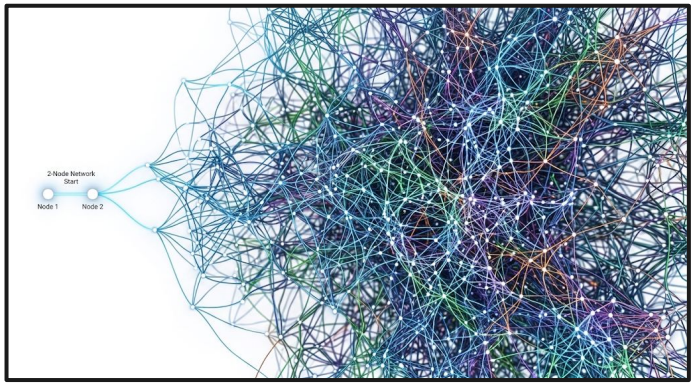
Mean-Field Learning

The Curse of Dimensionality (Revisited)

The Scaling Problem: In Multi-Agent Reinforcement Learning (MARL), the joint state and action spaces grow *exponentially* with the number of agents (n).

If 100 agents each have 4 possible actions, the joint action space is $|A| = 4^{100}$. You cannot calculate a traditional Q-Value for this.

The Reality: Standard algorithms (like MADDPG from Lecture 9 or CFR from Lecture 14) are mathematically and computationally impossible to run on massive swarms.



$\bar{a}$

Introduction to Mean-Field Reinforcement Learning

Borrowed from statistical physics. Instead of modeling the exact interaction between Agent i and every other Agent j, we model the interaction between Agent i and the **average behavior of the population** (the Mean-Field).

The Assumption: As the number of agents $N \to \infty$, the impact of any single agent on the system approaches zero.

The Math: The Q-function simplifies from evaluating all joint actions $Q_i(s, a_1, a_2, \ldots, a_N)$ to evaluating the agent's action against the mean action distribution :

$$Q_i(s, a_i, \bar{a})$$

SUBSAMPLE-MFQ (Anand et. al. 2025)

The Problem with Standard Mean-Field: Calculating the exact continuous mean action $\bar{a}$ is hard in dynamic settings.

SUBSAMPLE-MFQ: Recent work suggests learning a decentralized policy by randomly subsampling only k agents from n total agents.

The Theoretical Guarantee: The algorithm learns a policy in polynomial time relative to k. Crucially, the learned policy converges to the optimal policy with an error bound of:

$$\tilde{O}\left(\frac{1}{\sqrt{k}}\right)$$

This error bound is independent of n. Whether n is 1,000 or 1,000,000, subsampling k agents provides a guaranteed approximation.

Example: Autonomous Traffic Jam

The Scenario: You are an autonomous vehicle (Agent i) approaching a mega-intersection. Calculating exact trajectories for $n = 10,000$ cars is impossible. You decide to use a Mean-Field Subsampling approach with $k = 5$.

The Sensor Data: Your sensors randomly sample 5 nearby cars. Their intended actions are:

$$A_{sub} = \{\text{Brake}, \text{Accelerate}, \text{Brake}, \text{Brake}, \text{Maintain}\}$$

The Dynamics Rule: The system aggregates the mean-field policy using a simple **Majority Vote** mechanism to determine the crowd's momentum vector:

$$a_g(t) = \text{majority}(\{a^g_{h_j}(t) : j \in [k]\})$$

Your Task: 1. Calculate the mean-field action. 2. If your reward function heavily penalizes going *against* the mean-field momentum to avoid collisions, what should your policy output?

Example: Autonomous Traffic Jam (Solution)

Step 1: Apply the Aggregation Rule. Count the actions in our subset A_{sub} :

- Brake = 3
- Accelerate = 1
- Maintain = 1

Step 2: Calculate the Majority ($a_g(t)$). The majority action of the subsample is **Brake**. We assume the entire population of 10,000 cars is, on average, braking.

Step 3: Update Local Policy. To maximize the reward function (avoiding collisions by matching crowd momentum), Agent i must also choose to **Brake**.

The Takeaway: By only analyzing 5 agents, you made an optimal, safe decision in $O(1)$time!

Evolutionary and Meta-Learning

Beyond Gradients: Evolutionary Learning

Backpropagation and Gradient Descent require estimating the gradient of the environment. But what if your environment has discrete, non-differentiable rules (like routing packets in a network)?

Genetic Algorithms (GAs): A class of Black-Box Optimization.

- Initialize a population of random agent policies (neural network weights).
- Evaluate their "Fitness" (Total Reward) in the environment.
- Select the top performers and apply **Mutation** (random noise) and **Crossover** (mixing weights).

The Advantage: Highly parallelizable across thousands of CPUs and immune to local minima that trap gradient descent.

Meta-Black-Box Optimization (Discovering GAs) (Lu et. al. 2022)

Can an AI *learn* how to evolve other AIs?

- **Attention-Based Genetic Algorithms:** Recent research replaces the manual crossover/mutation steps with a **Self-Attention Mechanism** (a Transformer).
- **How it works:** The Transformer takes the entire population's current weights and fitness scores as input. It uses cross-attention to identify which "traits" (weights) led to high fitness, and automatically outputs the next generation's weights!
- **The Meta-Learning Result:** The model mathematically *discovers* optimal heuristic search algorithms completely on its own.

Continual Learning and Self-Play

The Need for Continual Learning

The Real World is Dynamic: Once deployed, decentralized agents face shifting environments (seasons changing, new traffic laws, different user behaviors).

The Problem: If an agent learns a new task using standard RL, it overwrites its neural network weights and suffers from **Catastrophic Forgetting** of past tasks.

Continual Learning: The paradigm of an agent learning continuously over its lifetime *without* forgetting previous knowledge.

Elastic Weight Consolidation (EWC)

The Math of Remembering: How do we prevent forgetting? We add a regularization penalty to the loss function during real-time adaptation.

EWC Equation:

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta_{A,i})^2$$

- $\mathcal{L}_B(\theta)$: Loss for the *new* task.
- $\theta_{A,i}$: The network weights from the *old* task.
- F_i : The Fisher Information Matrix. It measures how "important" a weight is to the old task.

The Result: The agent is allowed to heavily alter "unimportant" weights to learn the new task, but the math rigidly protects the weights crucial to its previous survival!

Self-Play for Self-Improvement

Where does the data come from? In pure decentralized settings, you can't rely on humans to constantly provide new training data.

The Auto-Curriculum: By copying an agent and forcing it to play against past versions of itself, the environment naturally increases in difficulty at the exact pace of the agent's learning.

Examples: AlphaGo Zero, OpenAI Five.

Combined with Evolutionary Learning, you can maintain a population of diverse past agents to play against, ensuring the agent is robust against multiple strategies, not just its own.

Decentralized Communication Topologies

If agents are scaling and adapting in real-time, how do they share knowledge?

Centralized (Cloud): All agents send gradients to a central server. (Single point of failure).

Fully Decentralized (Mesh): Agents only communicate with neighbors within a physical radius.

The Math of Consensus: Decentralized networks use **Gossip Algorithms** to reach consensus on the global state. Agent i updates its belief x_i by averaging with its neighbors $\mathcal{N}_i$:

$$x_i(t+1) = \sum_{j \in \mathcal{N}_i \cup \{i\}} W_{ij} x_j(t)$$

(Where W is a doubly stochastic weight matrix).

Emergent Communication

Do we need to program the network protocols manually? No.

Emergent Communication: We can treat the communication channel as an "Action Space" in MARL.

Agent 1 outputs a continuous vector (a message) alongside its physical action. Agent 2 receives that vector as part of its state observation.

Over time, via backpropagation, the agents *invent their own language* optimized strictly for solving the task efficiently, minimizing bandwidth usage automatically!

Conclusion

When and How to Scale

When to Scale: If the environment has homogeneous entities (traffic, drones, financial markets), you must scale to accurately simulate reality.

How to Scale (Rule of Thumb, May Vary Depending on Problem):

- $N < 10$**:** Use standard MARL (MADDPG, CTDE).
- $N \to \infty$**:** Use Mean-Field Learning or Subsampling.
- **Non-Differentiable / Hardware limits:** Use Evolutionary/Genetic Algorithms.
- **Real-time Adaptation:** Use Continual Learning and local decentralized consensus.

Summary and Relevance to Projects

Mean-Field RL allows us to solve massive swarm systems by interacting with the statistical average of the population rather than individuals.

Evolutionary Meta-Learning proves that AI can discover its own optimization algorithms via Self-Attention.

Continual Learning ensures agents adapt in real-time without catastrophic forgetting.

For your Senior Projects: If your project involves Swarm Robotics, Traffic Optimization, or Edge IoT Networks, you must explicitly state your communication topology and whether you are using a Mean-Field approximation!

Questions?