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

Embodied Learning from a Distributed Perspective

- ALOHA
- Mobile ALOHA

Distributed Imitation Learning

- Behavioral Cloning
- DAgger

Model Predictive Control

Self-Supervised Learning

- Data Augmentation
- Contrastive Learning

Evolutionary Learning

Computational Game Theory

- Nash Equilibrium

Class Announcements

Class Research Project Assignment

- In-person Mandatory Mid-Term Class Project Check-In on November 26
- 4 minute presentations during Mid-Term Project Check-In
- Create your github repositories per team by November 19 which you share on Brightspace
- You have to give a code overview (if you want to do slides, then 20 bonus points)
- Class performance on project is impressive. Keep up the fast learning of new coding skills on modern distributed computing concepts
- **See Brightspace for the Detailed Announcement and for Detailed Feedbacks**

Embodied Learning

Overview

Sometimes it's not enough for our agents to make decisions that affect the *virtual* environment, we need to interact with the real world!

Embodied Learning takes place when our agent occupies a physical *body*.

Examples:

- Self-Driving Car
- Home Robot
- Mechanical Arms
- Humanoid Robots

Embodied Agents are almost always multimodal because they often need to interface with the world as we do:

- Sight
- Sounds
- Touch

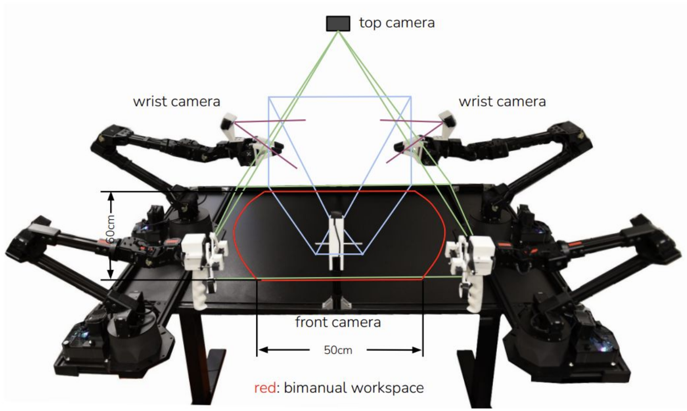
Example: ALOHA

The "Body" (ALOHA): A low-cost, open-source bimanual robot system. Human demonstrations are collected via teleoperation.

The "Brain" (ACT): An imitation learning policy that learns from the human data. It predicts "action chunks" (action sequences) to reduce compounding errors in the physical world.

Embodied Sensing: The policy is trained end-to-end, mapping raw pixels from four webcams directly to the robot's joint commands (a "pixel-to-action" model)

Complex Interaction: This system learns fine-grained, contact-rich tasks (like slotting a battery) with high success (80-90%) from minimal data (~50 demonstrations)



https://www.youtube.com/watch?v=HaaZ8ss-HP4

Limitations of Embodied Learning from a Distributed Perspective

Embodied learning, even today, requires lots of human supervision via *teleoperation* because the physical presence of a robot causes many safety concerns.

We can think of this as having *central hub* creating a bottleneck for our AI system.

If we structured this as a distributed system where a decentralized network of AI Agents could monitor the embodied agent, Embodied Learning could be scaled significantly.

This does, however, assume you have a method to automate the monitoring safely!

Imitation Learning - DAgger

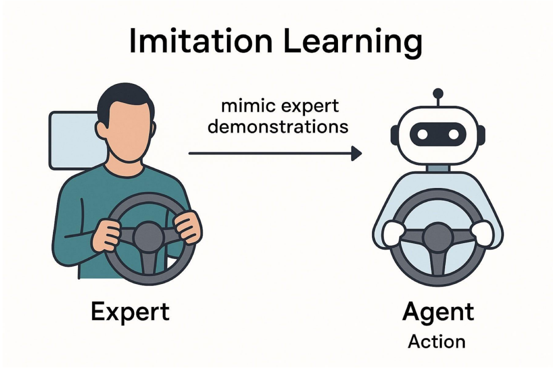
Imitation Learning

We saw that ALOHA uses Imitation Learning, not Reinforcement Learning. But what is Imitation Learning?

With real-world embodied agents, it is often extremely difficult or sometimes even impossible to train in the environment purely using traditional Reinforcement Learning.

Imitation Learning gives us a way to learn from *expert demonstrations*:

- Collect data from rollouts of an *expert policy*
- *Imitate* the expert as best you can!



Distributed Imitation Learning

Problem: Standard IL is sequential. A model must learn Task A, then Task B, then Task C, and needs to remember the full "context" of all tasks.

Distributed Solution: Learn from multiple, diverse experts (e.g., different cooking styles, different surgery techniques) in parallel.

Benefit: This solves the "context window" problem:

- Instead of one model, multiple models learn tasks A, B, and C simultaneously on different agents.
- The final policy is an "agglomeration" of these parallel-learned skills, creating a more robust model that doesn't need to remember a long, sequential history.

Application: This is the opportunity for systems like ALOHA distributing the learning to make it faster and more scalable.

Behavioral Cloning

The simplest algorithm for Imitation learning is **Behavioral Cloning:**

$\xi \in \Xi$	trajectory in expert demonstrations
$x \in \xi$	state along a trajectory
L	loss function (e.g., Euclidean norm, KL divergence, Cross-Entropy)

$$\hat{\pi}^* = \arg\min_{\pi} \sum_{\xi \in \Xi} \sum_{\boldsymbol{x} \in \xi} L(\pi(\boldsymbol{x}), \pi^*(\boldsymbol{x})).$$

Behavioral Cloning

The simplest algorithm for Imitation learning is **Behavioral Cloning:**

With this setup, we can think of Imitation Learning as a *supervised learning* problem!

- Input: x state encountered along expert trajectories
- Output: a action chosen by our policy network
- Target: a* action chosen by the expert for x

$$\hat{\pi}^* = \arg\min_{\pi} \sum_{\xi \in \Xi} \sum_{\boldsymbol{x} \in \xi} L(\pi(\boldsymbol{x}), \pi^*(\boldsymbol{x})).$$

Limitation: Out-of-distribution generalization

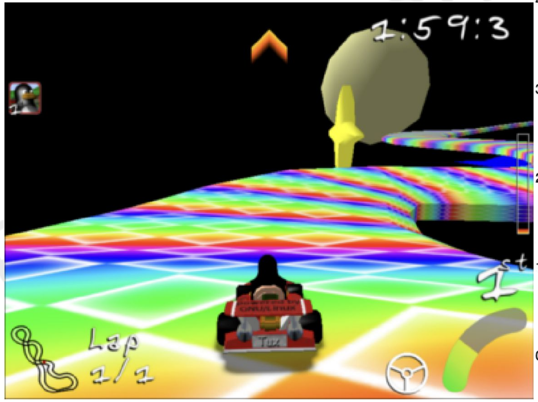
- If the expert doesn't explore the entire state space, our learned policy will have blind spots!

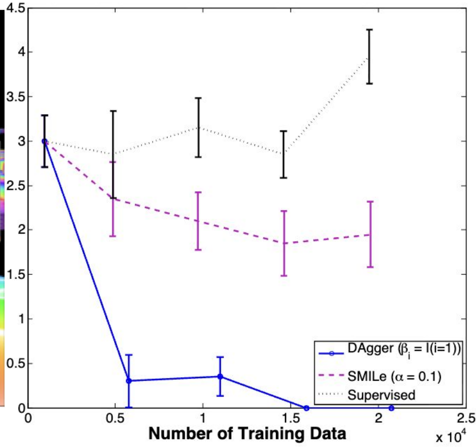
Dataset Aggregation

DAgger is an algorithm for training ***deterministic stationary policies*** in an imitation learning setting.

1. **Initialize the policy with the expert's decisions and do a rollout**
2. **Add encountered states never seen by the expert to the training dataset**
3. **Train the policy on the new, aggregated dataset**

This results in a policy robust to deviations from the expert's choices.





DAgger performs very well against SMILe (stochastic SoTA algorithm) and Supervised Learning on Super Tux Kart

Online no-regret reduction to fix covariate shift in IL

Strengths

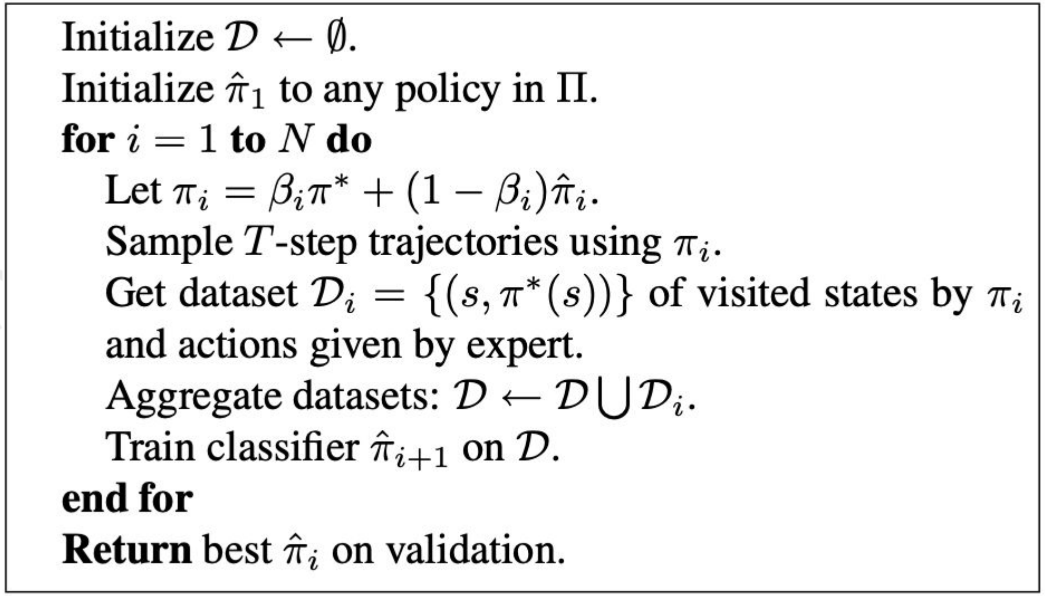
1. Strong theoretical guarantees for efficiency
2. Augmenting the data means you can still use any training procedure you want, and it will still improve your policy
3. Simple to understand and implement

Weaknesses

1. Over-reliance on querying the expert, which for many problems is impractical
2. In order to learn, the policy needs to interact with the environment directly, which could be unsafe for real-world problems (e.g. autonomous vehicles)

Algorithm

- Dataset D
- Policy pi
- State s
- Hyperparameter Beta - Controls how much the expert policy is imitated



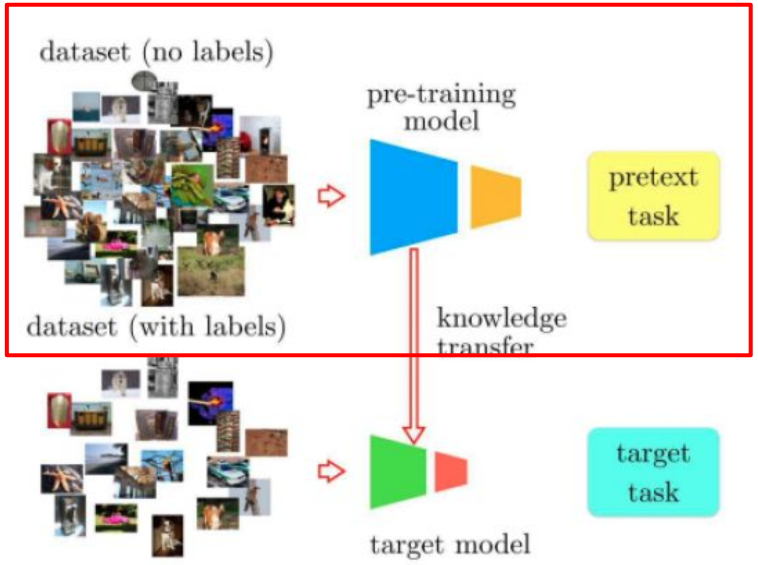
```
Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in $\Pi$.
for $i = 1$ to $N$ do
    Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
    Sample $T$-step trajectories using $\pi_i$.
    Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by $\pi_i$
    and actions given by expert.
    Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
    Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.
```

Algorithm 3.1: DAGGER Algorithm.

Self-Supervised Learning

Self-supervised learning

- Using a supervision signal that can be derived from the original signal.
    - We form a pretext task. Example, colorization, rotation prediction, jigsaw.
- Differs from unsupervised (eg., autoencoders), seems more effective.
- Important for scaling pre-training data!



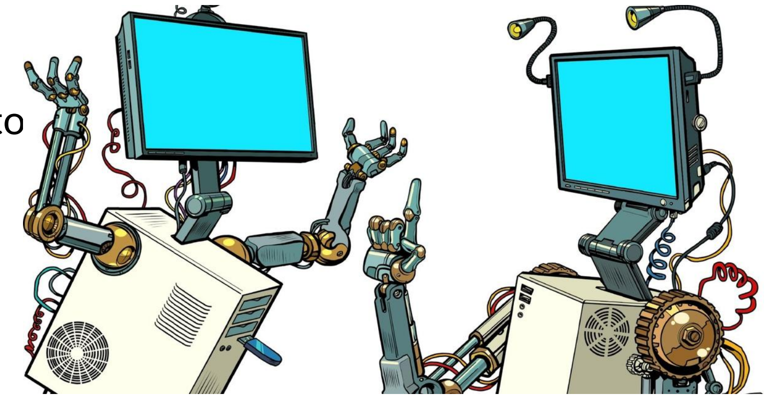
SSL in Distributed Systems

How do we know the pretext task (e.g., segmentation) is actually helpful? Who monitors the learning?

We can use a network of distributed AI agents to monito the self-supervision process in real-time.

These agents can "correct and adapt" the learning. For example, if a model's loss starts increasing, a monitor agent can automatically:

1. Stop the training.
2. Adjust hyperparameters (like fixing the loss function).
3. Resume the training.



Learning Segmentations (DINO)

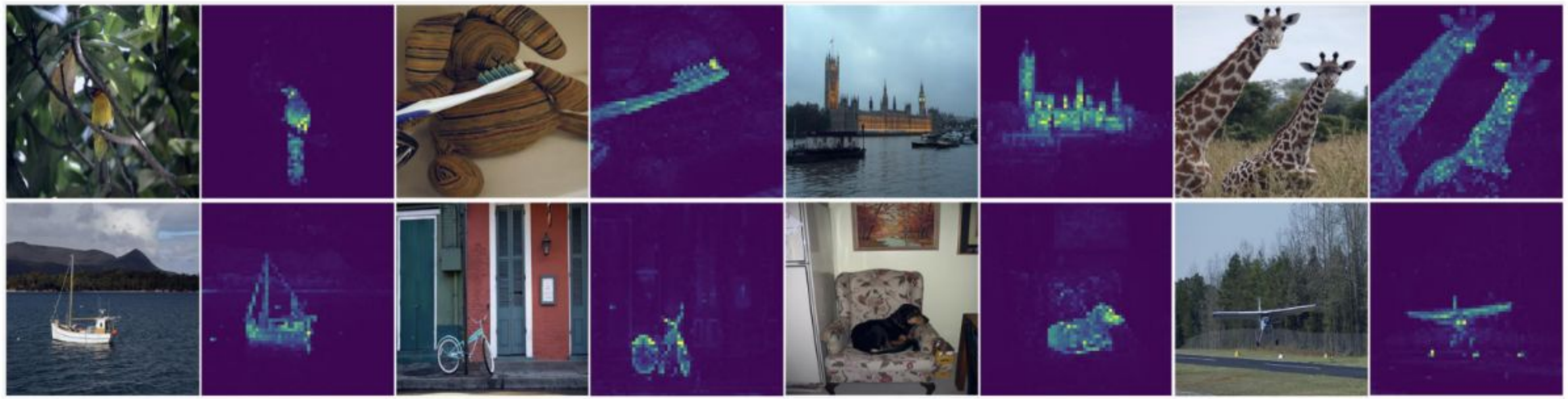


Figure 1: **Self-attention from a Vision Transformer with** 8×8 **patches trained with no supervision.** We look at the self-attention of the [CLS] token on the heads of the last layer. This token is not attached to any label nor supervision. These maps show that the model automatically learns class-specific features leading to unsupervised object segmentations.

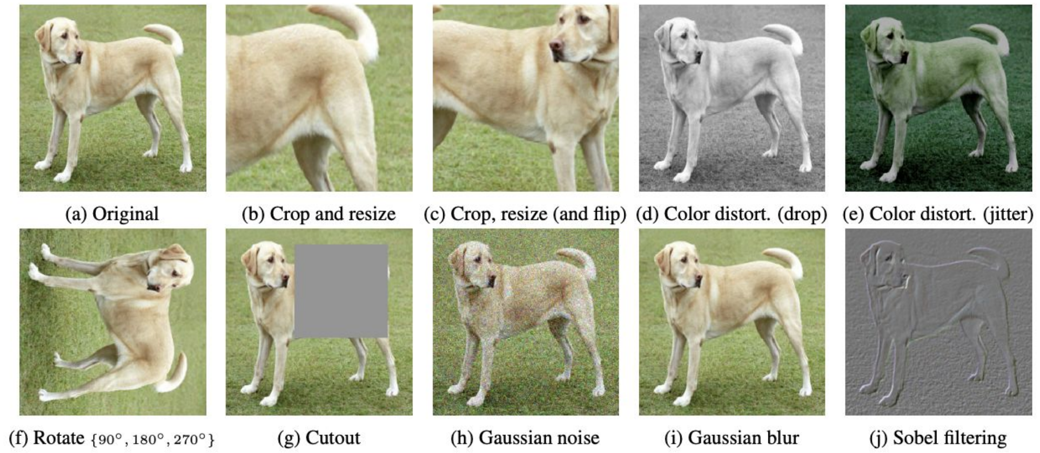
Contrastive Learning

- Find pairs of examples that are positive or negative.
- Learn embeddings such that positive pairs are closer than negative ones.
- Can be done supervised, eg., for face verification.
- Can be done through augmentations
- Positive class can be two crops of same image, change in contrast, etc....
- Danger of shortcuts.
- We want the network to learn important features, semantics, not trivial low-level cues.

Example: If we cut an image in half, just comparing the edges would identify positive examples. If we crop the image, color histogram comparison might work.

Example: Data Augmentation

- Contrastive learning allows us to classify images even after changing (*augmenting*) them!
- If we want to recognize image A, augmented versions of image A would be positive examples, and other images would be negative examples.



(a) Original
(b) Crop and resize
(c) Crop, resize (and flip)
(d) Color distort. (drop)
(e) Color distort. (jitter)
(f) Rotate {90°, 180°, 270°}
(g) Cutout
(h) Gaussian noise
(i) Gaussian blur
(j) Sobel filtering

Figure 4. Illustrations of the studied data augmentation operators. Each augmentation can transform data stochastically with some internal parameters (e.g. rotation degree, noise level). Note that we *only* test these operators in ablation, the *augmentation policy used to train our models* only includes *random crop (with flip and resize)*, *color distortion*, and *Gaussian blur*. (Original image cc-by: Von.grzanka)

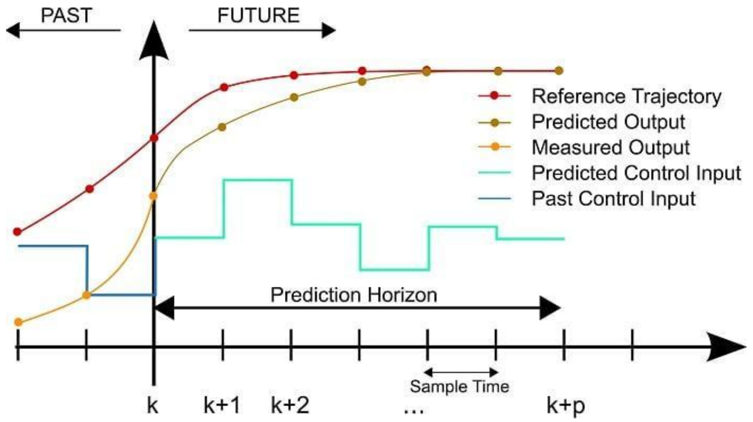
Model Predictive Control

Model Predictive Control

- **MPC** is a well-established framework for many safety-critical areas
  - Chemical industrial processes
  - Power grid management
  - Motion planning
- **Objective:** minimize cost while adhering to constraints and adapting to new environmental information in real-time
- Distributed Model Predictive Control (DMPC)
  - Divides a complex learning system into smaller subsystems,
  - Each subsystem is managed by a local predictive controller (agent)
  - The local agent optimizes its actions using a dynamic model and predicted future outcomes.
  - Each agent computes its control decisions locally and in parallel with other agents
  - Sharing some state or intent information helps to coordinate action space across the whole system
    - Information sharing based on past Distributed Deep Learning lectures

Model Predictive Control

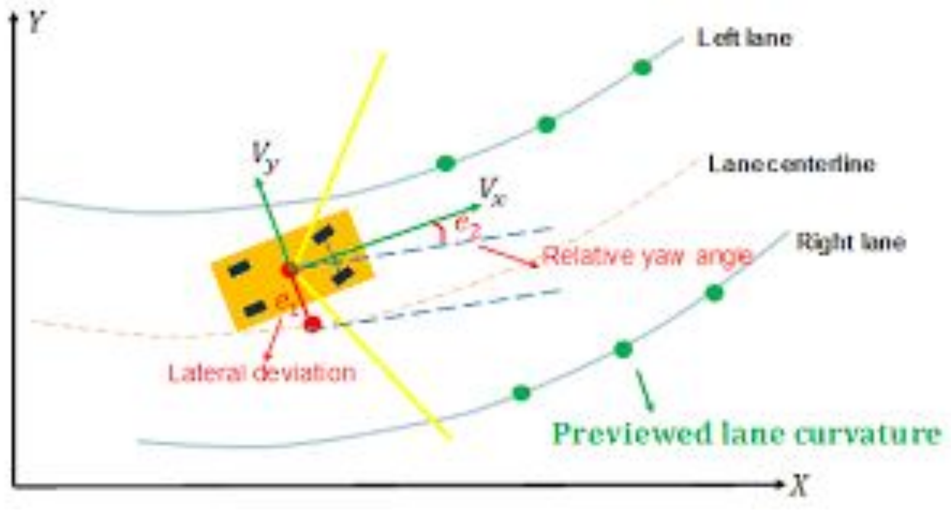
- At the current time k, the controller observes the "Measured Output" (the system's actual state).
- It computes an optimal sequence of "Predicted Control Inputs" for a future "Prediction Horizon" (from k to k+p).
- This sequence is calculated to make the "Predicted Output" follow the "Reference Trajectory" as closely as possible.
- Only the first control input (from k to k+1) is actually applied to the system.
- At the next time step (k+1), the entire process repeats: the controller measures the new state and calculates a new optimal plan.



Example: Self-Driving Car Steering

MPC is often used for problems involving known physical dynamics, such as the steering component of a self-driving car

Our *model* tells us how the car will move, and based on that we *predict* the steering angle that will safely *control* the vehicle!



Evolutionary Learning

Evolutionary Learning

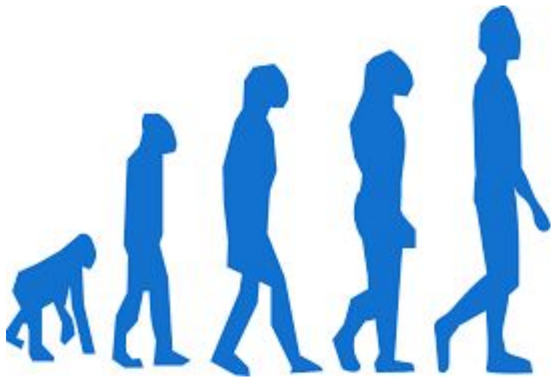
What if we could have algorithms that adapt to the environment similar to how species evolve over time?

Inspiration: Mimic biological evolution (natural selection) to solve problems.

Method: Start with a population of random solutions and iteratively improve them over "generations."

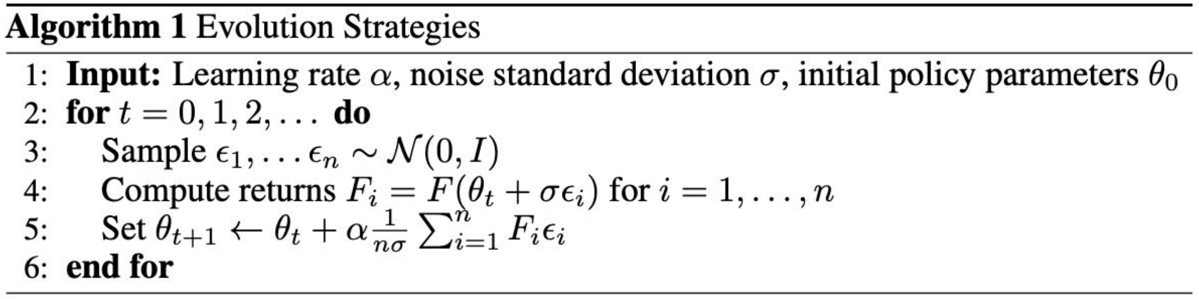
Process: In each generation, the best solutions are selected to "reproduce" (using recombination and mutation) to create the next generation, gradually "evolving" a better solution.

Can we make evolution faster? Answer: Distributed Learning



Evolution Strategy (OpenAI-ES)

- Explore by adding noise to the parameters θ, not by sampling actions
- Create a *“population”* by adding different Gaussian noise vectors (ε) to θ
- Evaluate each new policy in the environment to get a *fitness score* F
- Update with weighted average of noise vectors
    - Better policies have more influence



```
Algorithm 1 Evolution Strategies
1: Input: Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\theta_0$
2: for $t = 0, 1, 2, \ldots$ do
3:     Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
4:     Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
5:     Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
6: end for
```

Computational Game Theory

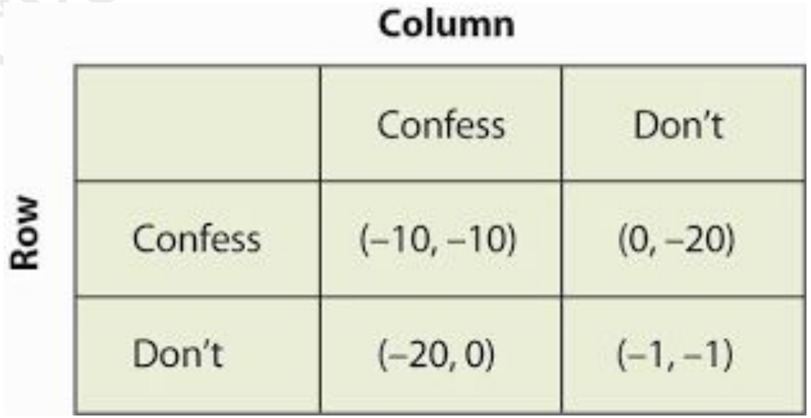
Game Theory for Distributed Systems

- In a multi-agent system, the environment is other agents. Everyone optimizes simultaneously.
- While we optimize system efficiency with distributed computing, we cannot forget about system performance which game theory helps to solve for.
- Sometimes, each agent has their own interests but cooperating (at least to an extent) is still necessary (e.g. cars merging into a lane)
- An agent's optimal action depends on what other agents do, creating a complex circular dependency.
- When we don’t have a centralized system, as is often the case with distributed systems, the problem is often to reach a “stable” state (e.g. a *Nash Equilibrium*)

Nash Equilibria

- A set of strategies (one per player) where no one can improve their outcome by unilaterally changing their strategy.
- It's a "stable" state.
- Everyone is playing their best response to everyone else's actions.

Example: Prisoner's Dilemma



Column

Row	Confess	Don't
Confess	(–10, –10)	(0, –20)
Don't	(–20, 0)	(–1, –1)

Hardness of Finding Nash Equilibrium

Nash's 1951 theorem proved an equilibrium (using mixed strategies) always exists for finite games.

However, multiple equilibria can exist, and finding the “best” equilibria (Pareto-optimal) is NP-hard!

The goal of many Multi-Agent Reinforcement Learning Problems is to converge to a good Nash Equilibrium.

Questions?