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 9, 2026 CSC 59866

Robustness, Consistency and Risk Mitigation in AI Agents & Offline Reinforcement Learning and Inverse Reinforcement Learning

Last Time on CSC59866...

Recall Lecture 11: We discussed PPO, DPO, and GRPO—algorithms that assume we can either interact with a simulation safely or sample an LLM millions of times.

The Reality Check: What happens when failure is catastrophic (e.g., autonomous driving)? We cannot let a car learn by randomly crashing into trees.

Today's Agenda:

1. **Robustness & Risk:** Formally defining safe boundaries for AI agents.
2. **Offline RL:** Learning strictly from pre-recorded datasets without environmental interaction.
3. **Inverse RL:** Figuring out the "reward function" by watching human experts.

Today's Agenda

Robustness, Consistency, and Risk Mitigation

Offline Reinforcement Learning

- Conservative Q-Learning

Inverse Imitation Learning

- Maximum Entropy

Robustness, Consistency, and Risk Mitigation

The Sim-to-Real Gap

The Problem: Agents trained in perfect simulations often fail in the real world due to **Distributional Shift**. The real-world data (sensor noise, unpredictable pedestrians) comes from a different distribution than the training data.

Robustness: A robust agent maintains a baseline level of performance even when the environment's dynamics are perturbed.

Adversarial Perturbations: Deep neural networks in RL are highly susceptible to tiny input noises that can completely flip a policy's output. Robustness requires training against these worst-case scenarios.

Consistency in Multi-Agent Environments

The Problem: We've talked about robustness against a static environment. But what happens when our agent is deployed into the real world alongside *other* learning agents and humans?

Consistency: A consistent agent must reliably maintain its performance and cooperative abilities even when interacting with a heterogeneous, unknown population of partners.

The Zero-Shot Coordination Challenge: Can we train an agent in simulation that can seamlessly cooperate with a novel human or AI partner on the very first try, without prior coordination?

Consistency in Multi-Agent Environments

The Problem: We've talked about robustness against a static environment. But what happens when our agent is deployed into the real world alongside *other* learning agents and humans?

Consistency: A consistent agent must reliably maintain its performance and cooperative abilities even when interacting with a heterogeneous, unknown population of partners.

The Zero-Shot Coordination Challenge: Can we train an agent in simulation that can seamlessly cooperate with a novel human or AI partner on the very first try, without prior coordination?

Modeling "Socially Rational" Populations

The Setup: Let's look at a 2-player general-sum game where each agent's utility (goal) is private.

The Assumptions: To mathematically guarantee cooperation, we might logically assume the target population we are deploying into is "Socially Rational":

1. **Individual Rationality:** All agents in the target population are rational learners trying to maximize their own rewards.
2. **Pareto-Efficiency:** When paired with each other, they naturally reach an equilibrium where no one can do better without hurting the other.

The Question: If we know the population acts this logically, is that enough information for our new agent to successfully learn to cooperate with them?

The Complexity of Learning to Cooperate

The Reality (My Recent Research): Those two logical assumptions are fundamentally **insufficient** to guarantee cooperation.

The "Adaptability" Trap: The problem is that the other agents are *just as adaptable as our agent*. If both agents are constantly updating their policies based on the other's actions, they become moving targets, leading to miscoordination.

The Takeaway: You cannot assume that just because a human or another AI is "rational," your agent will automatically sync up with them. Formal guarantees of cooperation require strict constraints, communication, or pre-established handshake protocols (like we discussed in Lecture 11!)

Risk Mitigation & Constrained MDPs

- **The Variance Problem:** Standard the Bellman equation and RL algorithms maximize *Expected Return*. They do not care if an action works perfectly 99% of the time but destroys the agent 1% of the time (high variance).
- **Constrained Markov Decision Processes (CMDPs):** To mitigate risk, we redefine the environment.
- **The Math:** We introduce a cost function $C(s, a)$ alongside the reward function $R(s, a)$. The agent must maximize expected reward subject to a strict safety threshold c:

$$\max_{\pi} \mathbb{E}_{\pi} \left[\sum \gamma^t R(s_t, a_t) \right] \quad \text{subject to} \quad \mathbb{E}_{\pi} \left[\sum \gamma^t C(s_t, a_t) \right] \leq c$$

Offline Reinforcement Learning

Introduction to Offline RL

The Concept: Offline RL is the setting where the agent learns from a static, previously collected dataset without any further environmental interaction.

The Motivation: In domains like healthcare or autonomous driving, running an untrained exploratory policy online is dangerous, unethical, or prohibitively expensive.

The Data: We rely entirely on a fixed buffer of logged experiences $\mathcal{D} = \{(s_i, a_i, r_i, s_{i+1})\}$ collected by some unknown behavior policy.

Offline RL vs. Imitation Learning

A Critical Distinction: Unlike Imitation Learning (like Behavioral Cloning or DAgger), the dataset in Offline RL does *not* need to come from an expert.

Suboptimal Data: The dataset can be a mix of optimal, suboptimal, heavily exploratory, and even completely random behaviors.

The Goal: The algorithm must stitch together the best parts of these varied, suboptimal trajectories to form a final policy that actually *outperforms* the data it was trained on.

Distributional Shift in Offline RL

The Core Failure: The biggest challenge in Offline RL is *distributional shift* causing severe extrapolation error.

The Q-Value Delusion: When a neural network evaluates an Out-of-Distribution (OOD) action (an action not present in the dataset) it often erroneously overestimates its value.

The Feedback Loop: Because the agent cannot interact with the environment to try the action and realize it was a bad idea, the policy exploits this overestimation and collapses.

Conservative Q-Learning

The Fix: We must constrain the learned policy to stay relatively close to the data distribution.

Conservative Q-Learning (CQL): An algorithmic approach that actively penalizes the Q-values of out-of-distribution actions during training.

The Result: This ensures the network maintains a conservative lower-bound estimate of the true Q-values, preventing the agent from executing dangerous, unproven actions in the real world.

Q-Value Update

Problem: Offline data doesn't contain all actions and states.

Impact: Learned model OVERestimates outcomes for states it has rarely seen

Solution: Regularize the Q-Values on some distribution that is NOT the dataset

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \; \alpha \cdot \left(\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s},\mathbf{a})\right] - \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\hat{\pi}_\beta(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s},\mathbf{a})\right]\right) + \frac{1}{2}\,\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(Q(\mathbf{s},\mathbf{a}) - \hat{\mathcal{B}}^{\pi}\hat{Q}^{k}(\mathbf{s},\mathbf{a})\right)^2\right].$$

Blue: Expected Q value, over the actions from the policy

Red: Expected Q value, over the actions from the behaviour policy (dataset) i.e. blue - red enforces the solution

Inverse Reinforcement Learning

The Reward Engineering Bottleneck

The Problem: In standard RL, you must manually program the reward function $R(s, a)$. For simple games, winning = +1. But how do you mathematically define the reward for "driving a car smoothly and politely"?

Misspecification: If you tell a car to "minimize travel time," it will run red lights. Hand-tuning rewards for complex behaviors is notoriously difficult and leads to reward hacking.

What is Inverse RL?

The Paradigm Shift: Instead of being given a reward and finding a policy, IRL is given an expert policy (or demonstrations) and attempts to extract the underlying reward function the expert is optimizing.

The Core Assumption: We assume the human expert is acting optimally (or near-optimally) with respect to some unknown, internal reward function.

The Ambiguity of IRL

The Mathematical Hurdle: The IRL problem is fundamentally ill-posed.
Multiple Solutions: Many different reward functions can explain the exact same expert behavior.
Example: If an expert stops at a stop sign, is the reward for "obeying the law", "avoiding a crash", or is the reward simply 0 everywhere, making *any* action technically optimal?.

Principle of Maximum Entropy

Breaking the Ambiguity: To solve this, modern IRL uses the Principle of Maximum Entropy.

The Solution: Among all the possible reward functions that match the expert's behavior, we choose the one that results in the policy with the highest entropy (the most randomness).

Why it Works: This ensures we make absolutely no additional assumptions about the expert's motivations other than the constraints we actually observed in the data.

Maximum Entropy IRL: The Math

To formalize this, we define the probability of the expert taking a specific trajectory τ as being exponentially proportional to the total reward of that trajectory:

$$P(\tau|\theta) = \frac{1}{Z(\theta)} \exp\left(\sum_t \theta^T f(s_t, a_t)\right)$$

- **The Features (f):** $f(s_t, a_t)$ represents the features of the state (e.g., speed, distance to the lane center).
- **The Weights (θ):** θ represents the hidden reward weights we are trying to learn.
- **The Partition Function (Z):** $Z(\theta)$ normalizes the probabilities.
- **The Optimization:** We find θ by maximizing the log-likelihood of the human's demonstrations. This mathematically forces our learned agent to perfectly match the human's *expected feature counts* while remaining maximally uncertain about everything else.

Questions?