

Meta reward learning; Multimodal rewards

Overview of Papers

01. Few Shot Preference Learning (Meta Learning)

Yibo Zhang

02. Watch, Try, Learn (Meta Learning)

Max Sobol Mark

03. Learning Multi-Modal Rewards from Rankings

Laya Iyer and Shreyas Kar

Introduction to Meta Learning

Let's Begin with an Example of "Classical" Learning

Task (1)

Given a *Spanish* dataset, learn a model to speak *Spanish*.



Hola!

Not Ideal!
Slow and require too
many data!

Another Example of “Classical” Learning

Task (2)

Given a *Japanese* dataset, learn a model to speak *Japanese*.



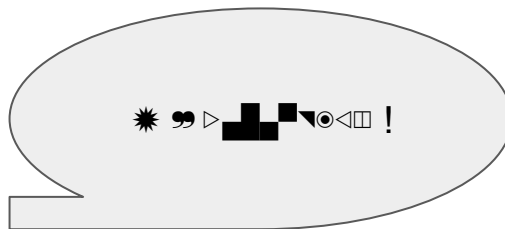
こんにちは!

Not Ideal!
Slow and require too
many data!

More...Examples of “Classical” Learning

Task (N)

Given a *Alien language* dataset, learn a model to speak *Alien language*.



Can it learn to learn?

Meta Learning: Learning to Learn

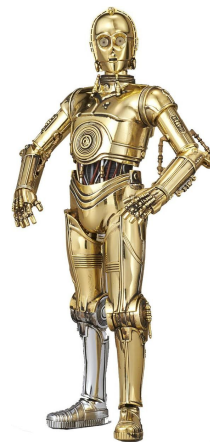
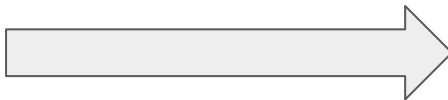
Meta Task

Given N language datasets, learn a *new* language more *quickly/proficiently*.



Hola!
こんにちは!
你好!
Hello!

Learning (from past tasks)
to learn (a new task).



* " ▶ █ █ █ !

A Formalism of “Classical” Learning

- A training dataset D ; a test dataset S .
- parameterized model θ ; training loss $L(\theta, D)$; test loss $L(\theta, S)$.
- The process of “Classical” Machine Learning:

$$\theta^* = \arg \min_{\theta} L(\theta, D)$$

- Denote the learning procedure as \mathcal{A} :

$$\theta^* = \mathcal{A}(D)$$

- The test result: $L(\mathcal{A}(D), S)$

Classically: a predefined learning procedure \mathcal{A} .
Meta learning: learning \mathcal{A} !

A Formalism of Meta Learning

- A set of training datasets $\{D_i\}_{i=1}^N$; a set of test dataset $\{S_i\}_{i=1}^N$.
- Parameterized model θ ; training loss $L(\theta, D)$; test loss $L(\theta, S)$.
- A family of learning algorithm \mathcal{A} parameterized by ω , given a dataset D_i :

$$\theta_i^* = \mathcal{A}(\omega, D_i)$$

What could \mathcal{A} be?

Meta learning:

$$\min_{\omega} \sum_i L(\mathcal{A}(\omega, D_i), S_i)$$

MAML: One of the Most Popular Meta Learning Method

- Recall our meta learning formalism: $\min_{\omega} \sum_i L(\mathcal{A}(\omega, D_i), S_i)$
- Let meta-parameter ω and model parameter θ be from the same space.
- Model-Agnostic Meta-Learning (MAML): let the learning algorithm be one step of gradient update (fine-tuning) with step size α , i.e.,

$$\mathcal{A}(\omega, D_i) = \omega - \alpha \nabla_{\omega} L(\omega, D_i) =: \theta^*$$

- Interpretation: the meta-parameter ω is the initialization of fine-tuning.



One Sentence Summary of Meta Learning

Given data from N tasks,
solve a new task more quickly/proficiently.

What if the task is...
Preference Learning?

Overview of Papers

01. Few Shot Preference Learning

Yibo Zhang

02. Watch, Try, Learn

Max Sobol Mark

03. Learning Multi-Modal Rewards from Rankings

Laya Iyer and Shreyas Kar

Few-Shot Preference Learning for Human-in-the-Loop RL

Joey Hejna

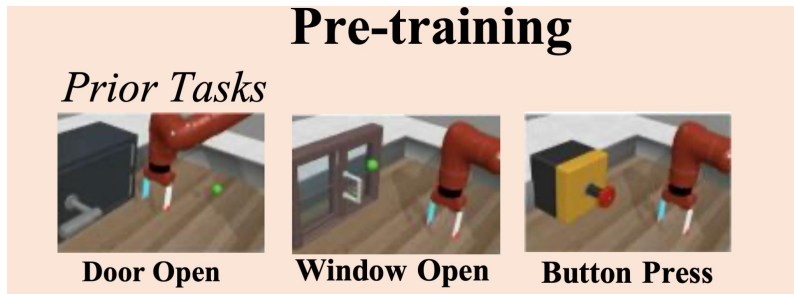
Dorsa Sadigh

Stanford University

Motivation of Few-shot Preference Learning

- Problem: learning reward function for a RL robot require expensive preference queries from human.
- Solution: meta-learning solve a new task with few number of human queries if given a dataset of tasks for pre-training.

Meta-learning with prior tasks



Few-shot adaptation



Problem Setup

- Given a state s , an action a , the reward function $r(s, a)$ is unknown.
- Each task has its own reward function and transition probabilities.
- The reward model is parameterized by ψ .
- A reward model ψ determines a RL policy ϕ .
- Consider preferences over trajectory segments:

$$\sigma = (s_t, a_t, s_{t+1}, a_{t+1}, \dots, s_{t+k-1}, a_{t+k-1})$$

- A dataset D consists of labeled queries (σ_1, σ_2, y) where y is the label.
- A loss function $L(\psi, D)$ captures how well the reward model ψ characterizes the preferences in dataset D .

Problem Setup (Define the Loss Function)

- A dataset D consists of labeled queries (σ_1, σ_2, y) where y is the label.
- A segment of trajectory:
- $$\sigma = (s_t, a_t, s_{t+1}, a_{t+1}, \dots, s_{t+k-1}, a_{t+k-1})$$
- The loss function is binary cross entropy (CE):

$$L(\psi, D) = \mathbb{E}_{(\sigma_1, \sigma_2, y) \sim D} [\text{CE}(y, P_\psi(\sigma_1 \succ \sigma_2))]$$

where

$$P_\psi(\sigma_1 \succ \sigma_2) := \frac{\exp\left(\sum_{(s,a) \in \sigma_1} r_\psi(s, a)\right)}{\exp\left(\sum_{(s,a) \in \sigma_1} r_\psi(s, a)\right) + \exp\left(\sum_{(s,a) \in \sigma_2} r_\psi(s, a)\right)}$$

Method Component 1: Pre-training with Meta Learning

- Given datasets $\{D_i\}_{i=1}^N$ of labeled preferences queries for N Pretraining tasks;
- Train a reward model ψ using MAML!

$$\min_{\psi} \sum_i L(\psi - \alpha \nabla L(\psi, D_i), D_i)$$

- Note: data $\{D_i\}_{i=1}^N$ for prior tasks can come from offline datasets, simulated policies, or actual humans.

Method Component 2: Few-shot Adaptation

- Given a pre-trained reward model ψ
- For time step $t = 1, 2, \dots$
 - Find pairs of trajectories (σ_1, σ_2) with preference uncertainty based on ψ .
 - Query human preference y and forms a new dataset D_{new}
 - Update the reward model:

$$\psi' \leftarrow \psi - \alpha \nabla_{\psi} L(\psi, D_{new})$$

- Update the policy with the new reward model ψ' .

Method: Overview

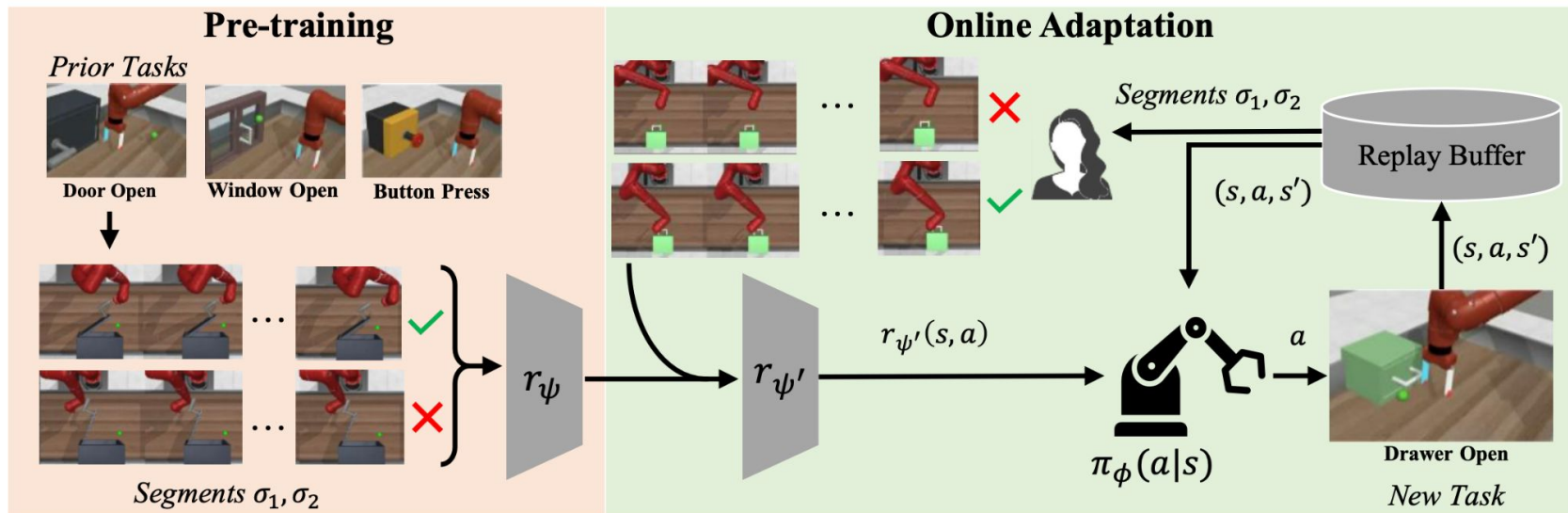


Figure 1: An overview of our method. **Pre-training (left):** In the pre-training phase we generate trajectory segment comparisons using data from a family of previously learned tasks and use them to train a reward model. **Online-Adaptation (Right):** After pre-training the reward model, we adapt it to new data from human feedback use it to train a policy for a new task in a closed loop manner.

Experimental Results

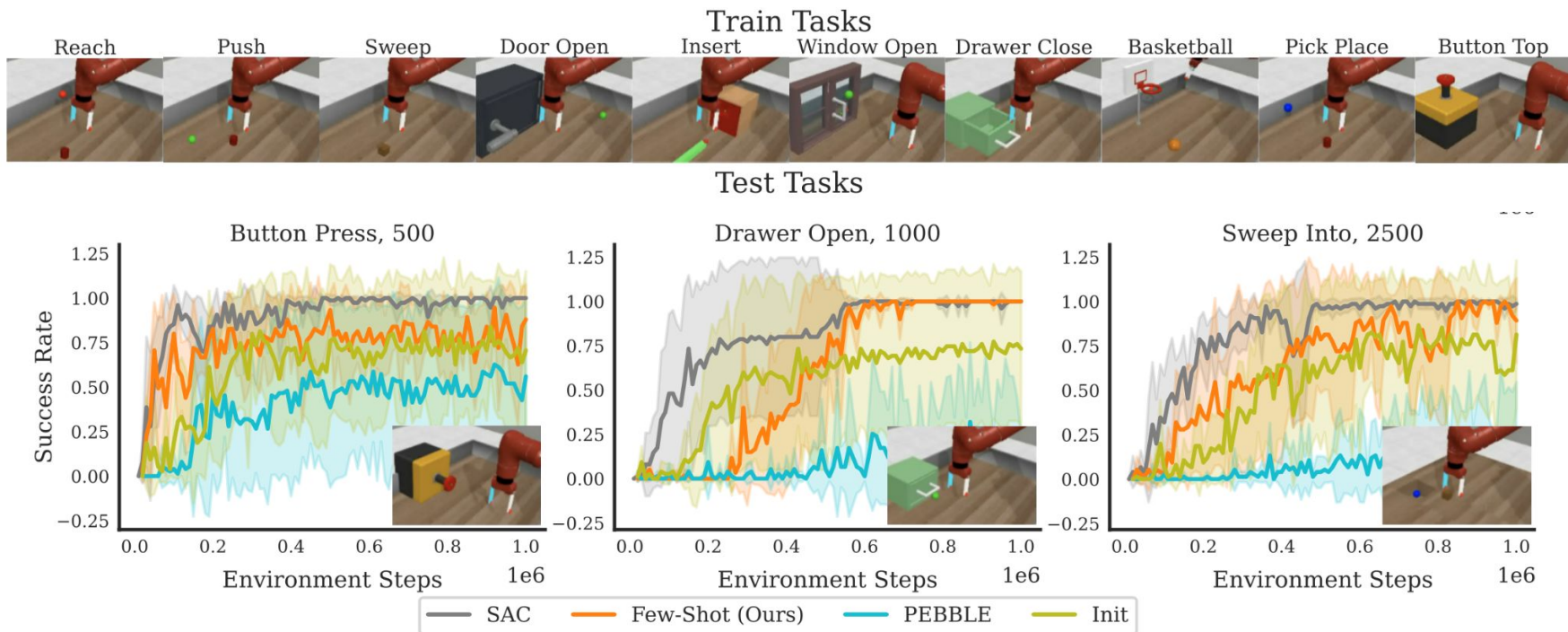


Figure 2: Results on MetaWorld tasks. The title of each subplot indicates the task and number of artificial feedback queries used in training. Results for each method are shown across five seeds.

Conclusion and Discussions

- Conclusion: meta learning reward models reduce the number of queries of human preferences.
- Discussions:
 - Many queries ask human to compare almost identical trajectories.
 - Despite the improved query complexity, it still needs an impractical amount of queries.
 - If the new task is too out-of-distribution, training from scratch is better.

Overview of Papers

01. Few Shot Preference Learning

Yibo Zhang

02. Watch, Try, Learn

Max Sobol Mark

03. Learning Multi-Modal Rewards from Rankings

Laya Iyer and Shreyas Kar

Watch, Try, Learn

Meta-Learning from Demonstrations and Rewards

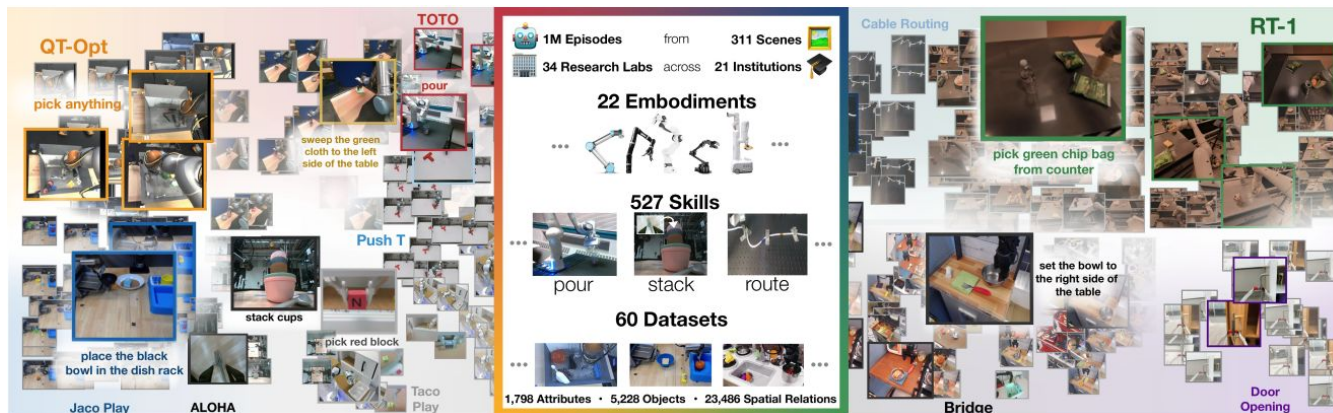
Allan Zhou, Eric Jang, Daniel Kappler, Alex Herzog, Mohi Khansari, Paul Wohlhart, Yunfei Bai,
Mrinal Kalakrishnan, Sergey Levine, Chelsea Finn

Motivation

- What is the bottleneck for robotics progress?
- Key insight of the paper to bypass the bottleneck.

Data is a key bottleneck

- LLMs have internet-scale datasets to train from, but there isn't (yet) an equivalent for robotics.
- Data collection is expensive
 - Robots are expensive, so people don't have access to them.
 - Human time is expensive, so it's hard to collect a lot of demonstrations.
- Largest dataset today: Open X-Embodiment



Insight: binary user feedback is easy to give!

- Getting sparse reward feedback is much cheaper than hiring robot operators.
- Learn new tasks by:
 - Getting only 1 demonstration.
 - Having the robot **try to solve** the task 1+ times.
 - Learning from binary success annotations for each trial.

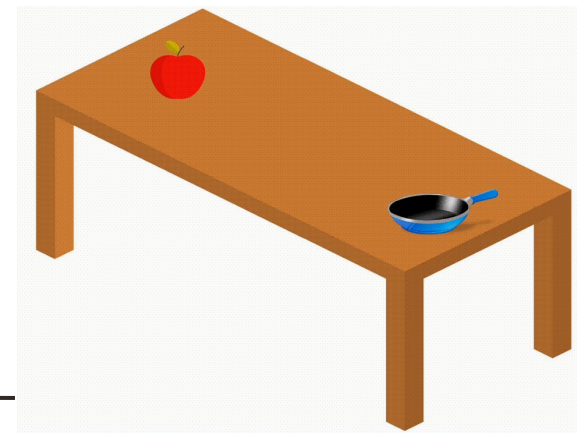
Example: what task is this:

Insight: binary user feedback is easy to give!

- Getting sparse reward feedback is much cheaper than hiring robot operators.
- Learn new tasks by:
 - **Getting only 1 demonstration.**
 - Having the robot try to solve the task 1+ times.
 - Learning from binary success annotations for each trial.

Example:

I'm going to show you a demonstration

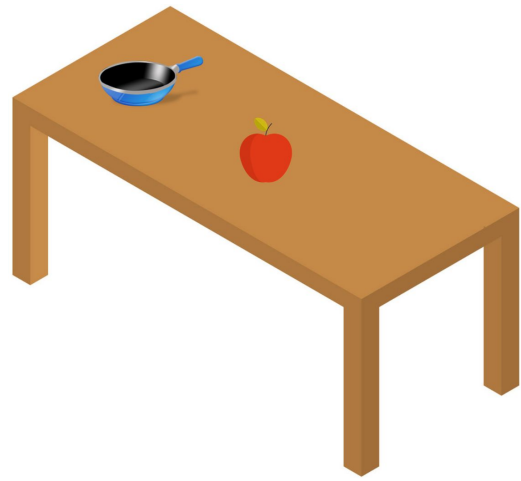


Insight: binary user feedback is easy to give!

- Getting sparse reward feedback is much cheaper than hiring robot operators.
- Learn new tasks by:
 - Getting only 1 demonstration.
 - **Having the robot try to solve the task 1+ times.**
 - Learning from binary success annotations for each trial.

Example:

We are now presented with this situation:

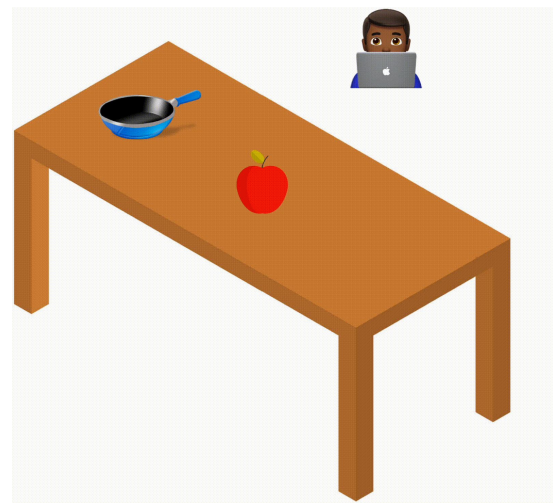


Insight: binary user feedback is easy to give!

- Getting sparse reward feedback is much cheaper than hiring robot operators.
- Learn new tasks by:
 - Getting only 1 demonstration.
 - **Having the robot try to solve the task 1+ times.**
 - Learning from binary success annotations for each trial.

Example:

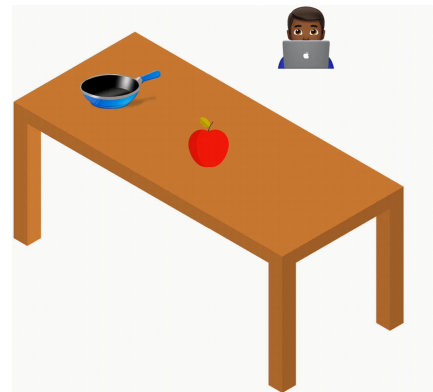
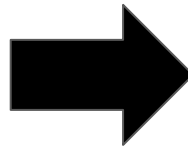
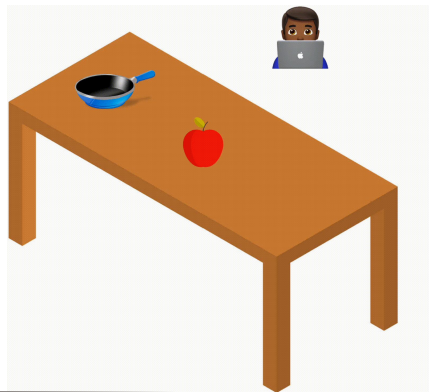
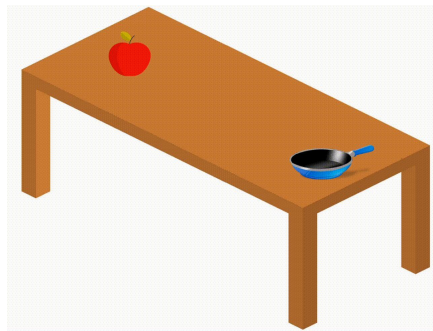
We try, and get feedback



Insight: binary user feedback is easy to give!

- Getting sparse reward feedback is much cheaper than hiring robot operators.
- Learn new tasks by:
 - Getting only 1 demonstration.
 - Having the robot try to solve the task 1+ times.
 - **Learning from binary success annotations for each trial.**

Example: learn from demo and trial



Summary of contributions

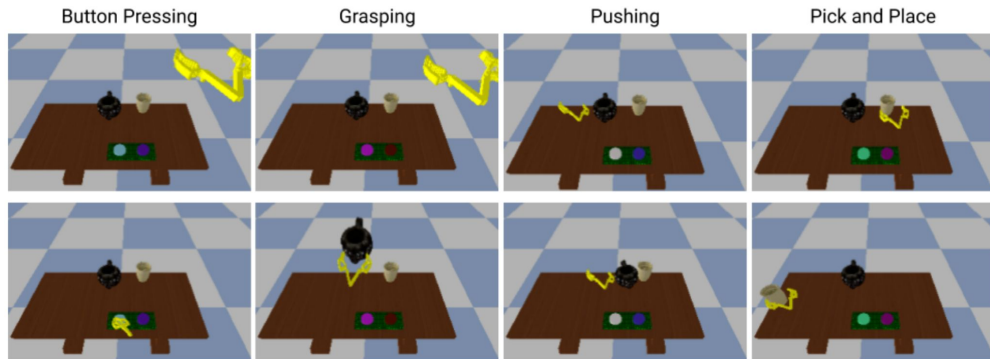
- Meta-learning algorithm for incorporating demos and binary feedback to solve new tasks.
- Especially applicable for robotics, where we can have many robots try things in parallel.

Preliminaries

- Meta-Learning Primer
- Tasks as MDPs

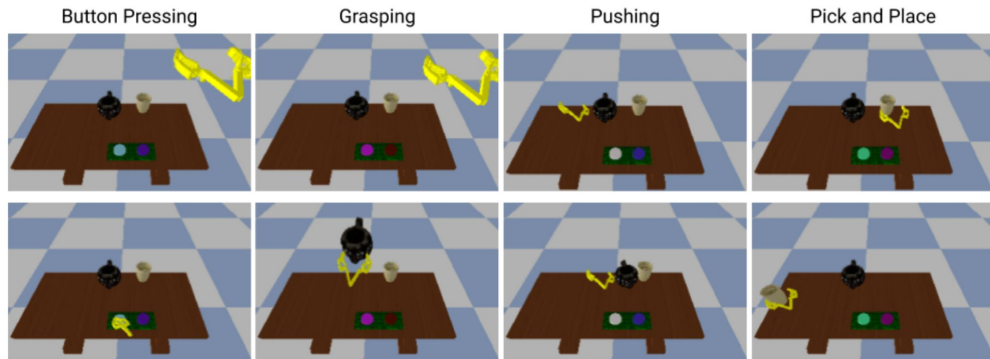
Meta-Learning: Learning to learn new tasks.

- Single-task paradigm:
 - Given a dataset $\{(x, y)_k\}$, learn function $f_{\theta}(x) \rightarrow y$.
- Meta-learning paradigm:
 - Given a task distribution $P(T)$,
 - And data for n tasks coming from distributions $\{p_i(x), p_i(y | x)\}_i^n$,
 - Learn function $f'(\text{new task}) \rightarrow (f(x) \rightarrow y)$



Task as MDPs

- Define task T_i as finite-horizon MDP $\{S, A, r_i, P_i\}$
 - S, A are shared between tasks, reward function r and dynamics P are not.
 - S is the space of RGB images.
 - A is the space of end-effector positions, rotations, openings.
 - Assume r is sparse ($r_i(s) = 1$ if task i is completed at s).
 - P_i is the (unknown) dynamics function $P_i(s_{t+1} | s_t, a_t)$
 - In the real world, there is only one dynamics function.
 - This paper considers simulations, where there's different objects per task.
- Tasks come from (unknown) distribution $p(T)$.



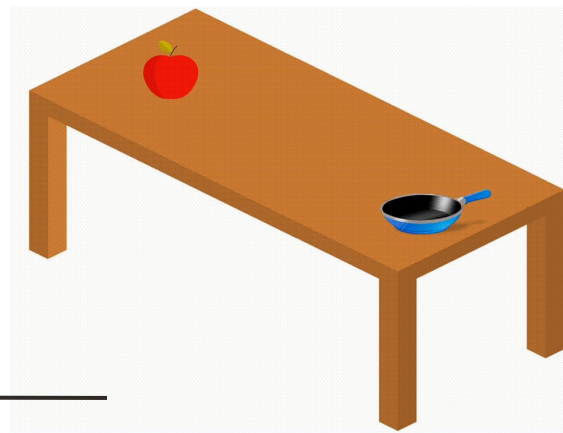
Problem statement

We *ultimately* want to obtain an agent that can

- 1) WATCH
- 2) TRY
- 3) LEARN

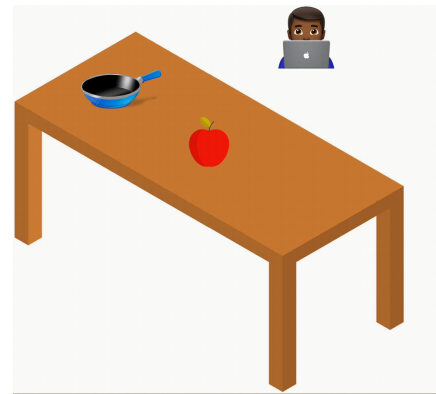
Problem Statement: **Watch**, Try, Learn

- We give the agent K demonstrations of the *target* task
 - K demonstrations $\{(s_0, a_0), \dots, (s_H, a_H)\}_i$ that succeed at target task.
- These demonstrations alone might not be sufficient for full task specification.
- Output of this phase: a policy capable of gathering information about a task given demonstrations.



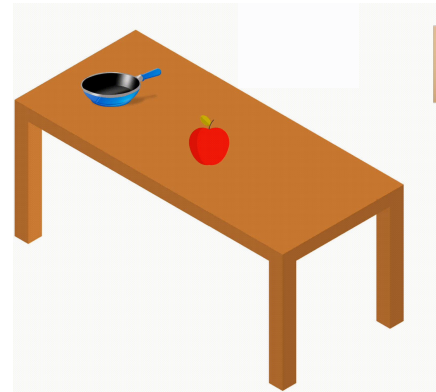
Problem Statement: Watch, **Try**, Learn

- Agent attempts the task for L trials.
- Humans provide one binary reward for each trial.
- Output of this phase: L trajectories and corresponding feedback that hopefully disambiguate the task.



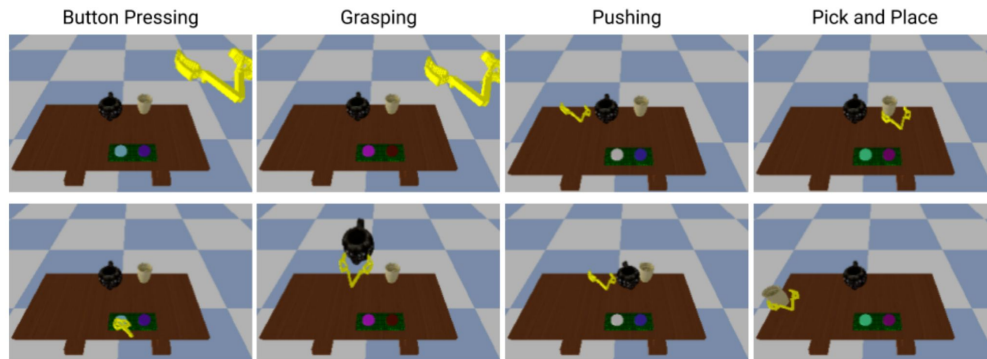
Problem Statement: Watch, Try, **Learn**

- Agent learns from both demonstrations and trials
- Output of this phase: policy capable of solving the target task.



Problem Statement: What are we given?

- To train agents that can watch, try, and learn, we are given a dataset of expert demonstrations containing multiple demos per task.



Method

- How do we train an agent from the expert demonstrations
- How do we train an agent from trials, human feedback, and demos

Method: Training to **Watch**

- Sample task T_i with expert demonstrations $\{\mathbf{d}_{i,k}\}$
- Train $\pi_{\theta}^{\text{watch}}(a|s, \{\mathbf{d}_{i,k}\})$ with *meta-imitation learning*:
 - Sample another demonstration of the same task $\mathbf{d}_i^{\text{test}}$
 - Regress $\pi_{\theta}^{\text{watch}}(a|s, \{\mathbf{d}_{i,k}\})$ to the actions taken on $\mathbf{d}_i^{\text{test}}$

Concrete example:

Demo1 Demo2 Demo3 Demo4

Method: Training to **Watch**

- Sample task T_i with expert demonstrations $\{\mathbf{d}_{i,k}\}$
- Train $\pi_{\theta}^{\text{watch}}(a|s, \{\mathbf{d}_{i,k}\})$ with *meta-imitation learning*:
 - Sample another demonstration of the same task $\mathbf{d}_i^{\text{test}}$
 - Regress $\pi_{\theta}^{\text{watch}}(a|s, \{\mathbf{d}_{i,k}\})$ to the actions taken on $\mathbf{d}_i^{\text{test}}$

Concrete example:

Demo1 Demo2 Demo3 Demo4

If the agent watches Demo1 and Demo2, it should act like Demo3 and Demo4

Method: Training to Watch

- Sample task T_i with expert demonstrations $\{\mathbf{d}_{i,k}\}$
- Train $\pi_\theta^{\text{watch}}(a|s, \{\mathbf{d}_{i,k}\})$ with *meta-imitation learning*:
 - Sample another demonstration of the same task $\mathbf{d}_i^{\text{test}}$
 - Regress $\pi_\theta^{\text{watch}}(a|s, \{\mathbf{d}_{i,k}\})$ to the actions taken on $\mathbf{d}_i^{\text{test}}$

Concrete example:

Demo1 Demo2 Demo3 Demo4

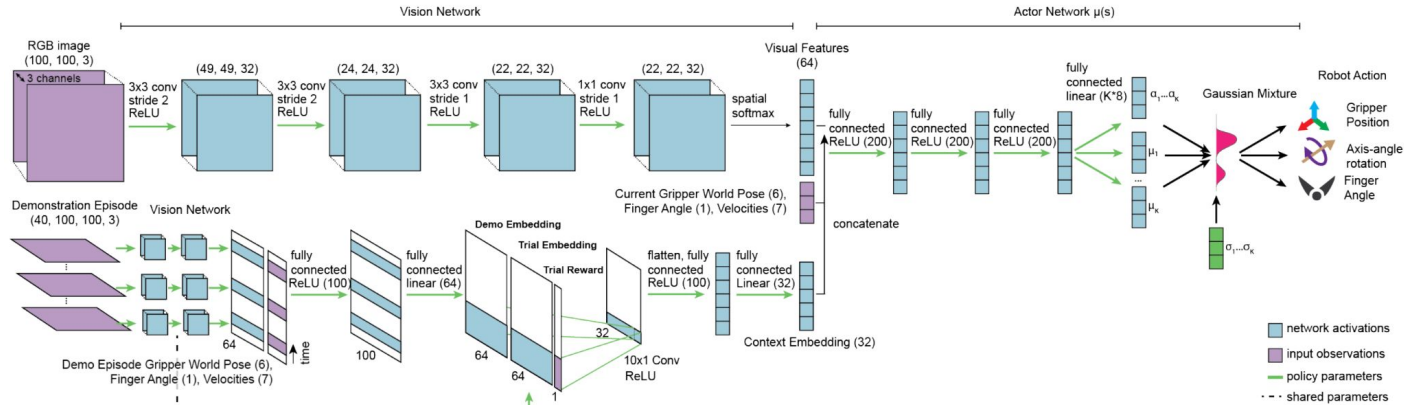
If the agent watches Demo1 and Demo2, it should act like Demo3 and Demo4

$$\mathcal{L}^I(\theta, \mathcal{D}_i^*) = \underbrace{\mathbb{E}_{\{\mathbf{d}_{i,k}\} \sim \mathcal{D}_i^*}}_{\text{Sample demos}} \underbrace{\mathbb{E}_{\mathbf{d}_i^{\text{test}} \sim \mathcal{D}_i^* \setminus \{\mathbf{d}_{i,k}\}}}_{\text{Sample optimal trajectory from the same task}} \underbrace{\mathbb{E}_{(s_t, a_t) \sim \mathbf{d}_i^{\text{test}}}}_{\text{For each timestep in the optimal trajectory}} \left[-\log \pi_\theta^I(a_t | s_t, \{\mathbf{d}_{i,k}\}) \right]$$

Maximize the likelihood of optimal actions given the demos.

Method: How do we condition the policy on demos?

- Encode 40 ordered frames from the demos
- Concatenate with current observation features



Method: What does the agent **Try**?

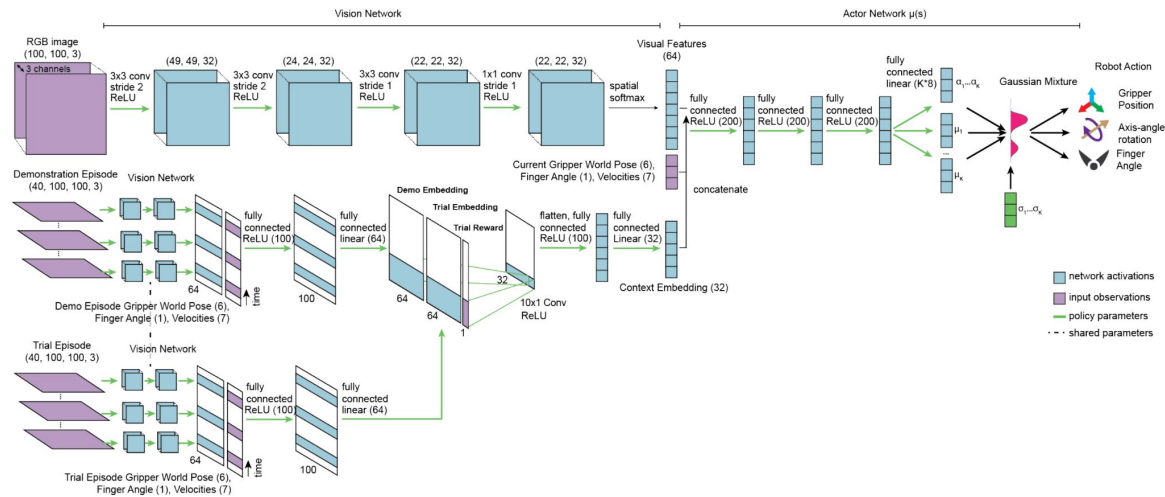
- When given demonstrations $\{\mathbf{d}_{i,k}\}$, deploy $\pi_{\theta}^{\text{watch}}(a|s, \{\mathbf{d}_{i,k}\})$ to collect L trials.
- This is equivalent to exploring with Thompson sampling.
- Human labels each trial as success/failure.

Discussion Questions:

Method: Training to **Learn** from the trials

- Want to train $\pi_{\theta}^{\text{watch}}(a|s, \{\mathbf{d}_{i,k}\}, \{\tau_{i,j}\})$ to act optimally.
- We can use the same strategy as before, but conditioning also on the trials!

$$\mathcal{L}^{\Pi}(\phi, \mathcal{D}_i, \mathcal{D}_i^*) = \mathbb{E}_{(\{\mathbf{d}_{i,k}\}, \{\tau_{i,\ell}\}) \sim \mathcal{D}_i} \mathbb{E}_{\mathbf{d}_i^{\text{test}} \sim \mathcal{D}_i^* \setminus \{\mathbf{d}_{i,k}\}} \mathbb{E}_{(s_t, a_t) \sim \mathbf{d}_i^{\text{test}}} \left[-\log \pi_{\phi}^{\Pi}(a_t | s_t, \{\mathbf{d}_{i,k}\}, \{\tau_{i,\ell}\}) \right]$$

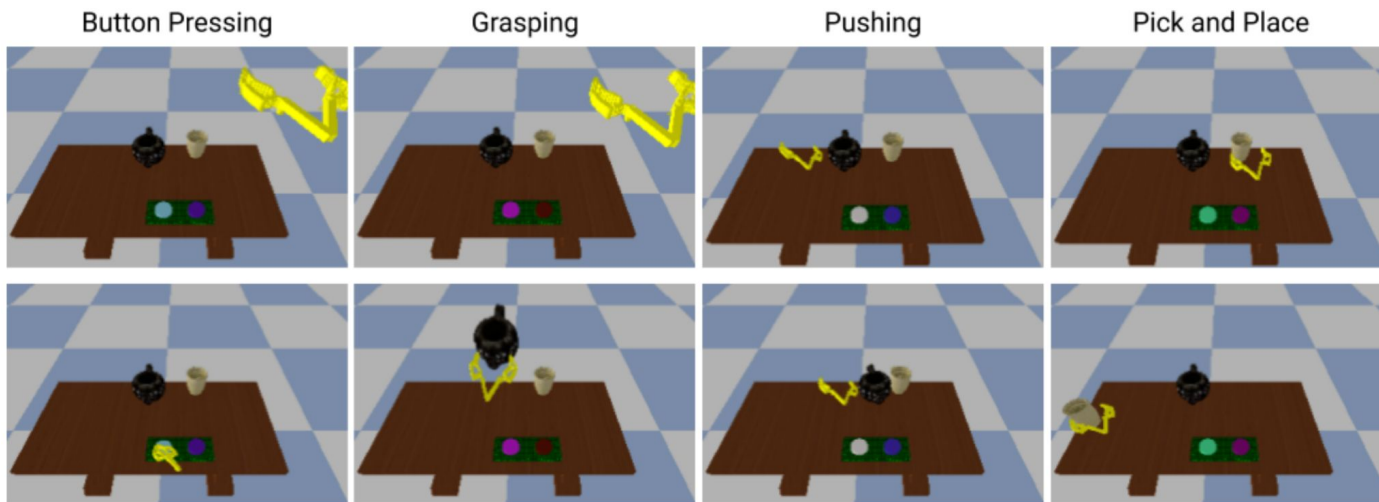


Experiments

- Gripper environment setup
- Baselines
- Results

Gripper environments

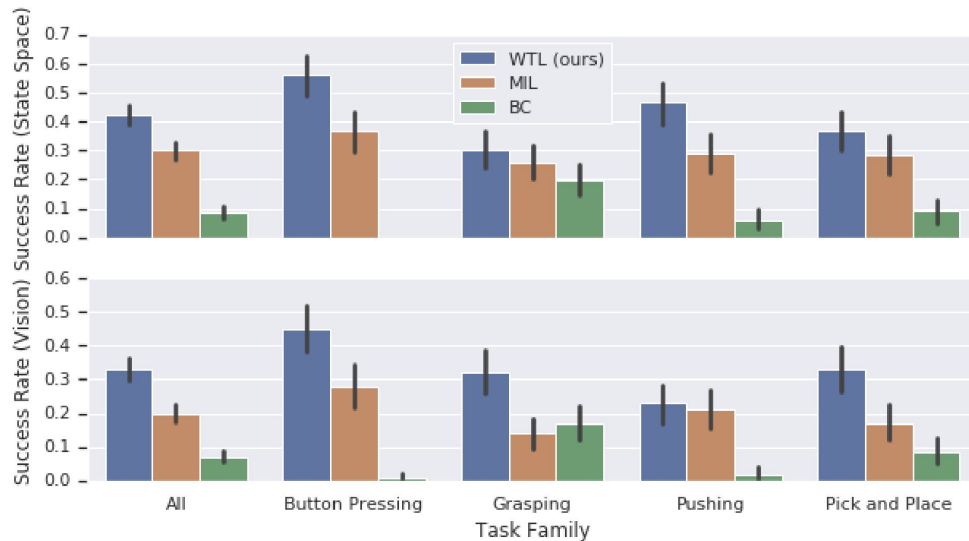
- Each task has two objects sampled from hundreds.
- Tasks are generally ambiguous given just 1 demo.



Baselines

- **Behavior Cloning**: simply train with maximum log-likelihood on all tasks.
- **Meta-imitation learning**: Run the policy after the Watch step, with no trials.
- **Behavior Cloning + SAC**: BC pre-training followed by Reinforcement Learning with the task-specific data.

Results



METHOD	SUCCESS RATE
BC	.09 ± .01
MIL	.30 ± .02
WTL, 1 TRIAL (OURS)	.42 ± .02
RL FINE-TUNING WITH SAC	
BC + SAC, 1500 TRIALS	.11 ± .07
BC + SAC, 2000 TRIALS	.29 ± .10
BC + SAC, 2500 TRIALS	.39 ± .11

Discussion question

- Task distribution considered is pretty narrow - do you think this algo could learn entirely new tasks?
 - Relevant posterior work: Decision-Pretrained Transformer.

- The L trials are independent, i.e. the policy has no idea what happened in other trials.
 - What is one key advantage of this?
 - What is one key disadvantage?

Overview of Papers

01. Few Shot Preference Learning

Yibo Zhang

02. Watch, Try, Learn

Max Sobol Mark

03. Learning Multi-Modal Rewards from Rankings

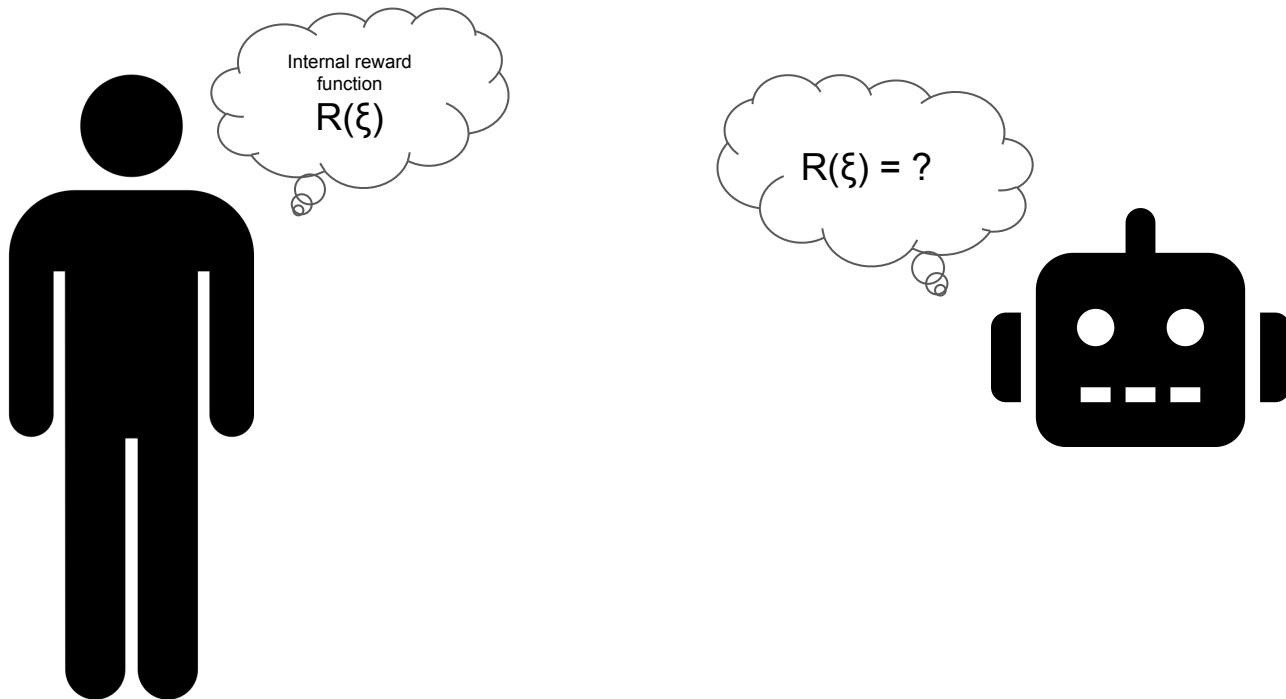
Laya Iyer and Shreyas Kar

Learning Multi-modal Rewards from Ranking

Vivek Myers, Erdem Bıyık, Nima Anari, Dorsa Sadigh

Intro to Reward Learning Problem

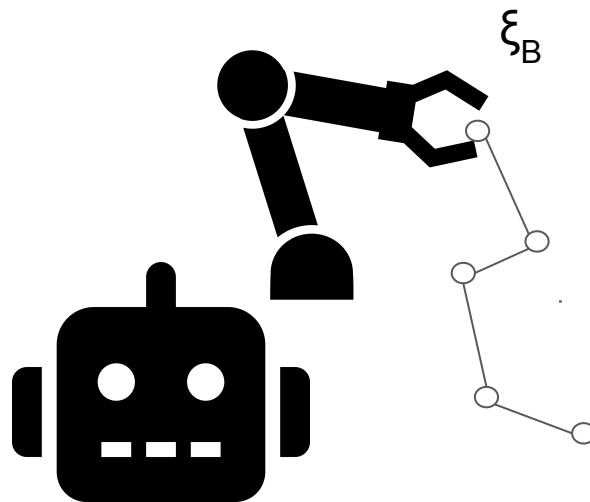
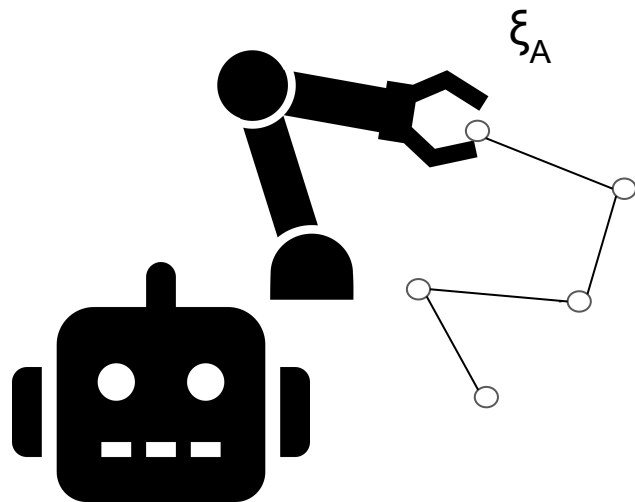
We want the a robot to perform a specific task



Learning based on Comparison

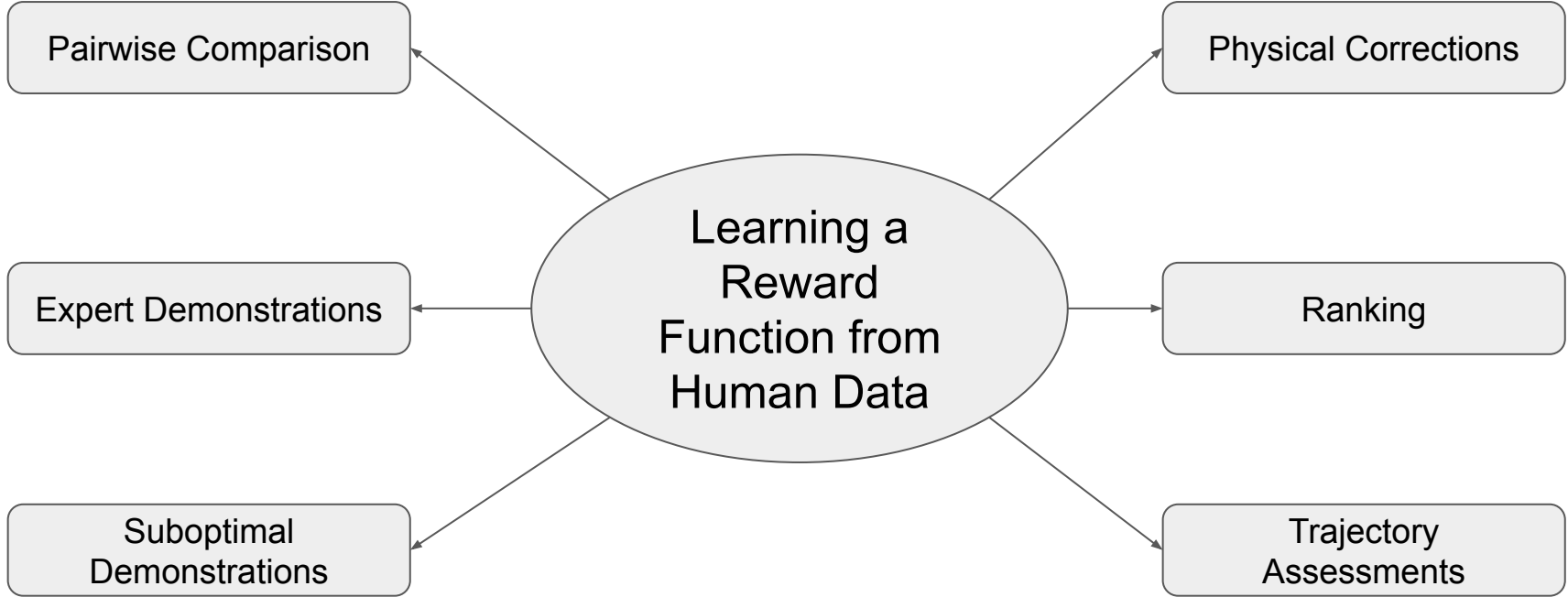
Pairwise Comparison

The robot can show different trajectories to the user



ξ_A or ξ_B

User indicates preferences \rightarrow Robot learns reward function

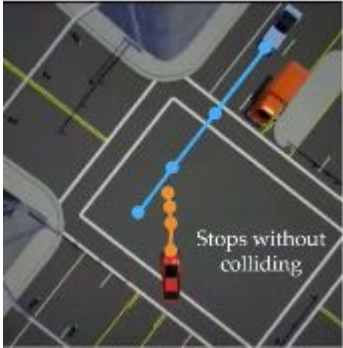


ALL UNDER THE ASSUMPTION THAT THE MODEL IS UNIMODAL ... BUT WHAT DOES THAT MEAN ???

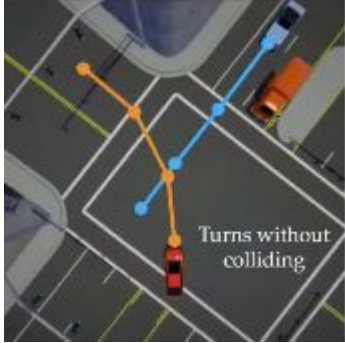
Example One: Autonomous Vehicle Simulation



Timid Driver Pattern



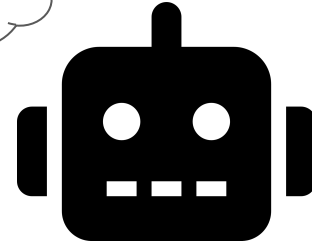
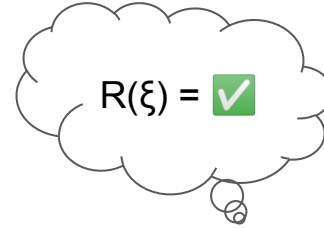
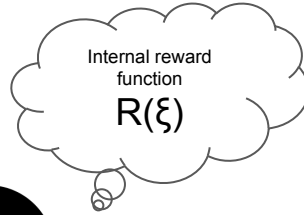
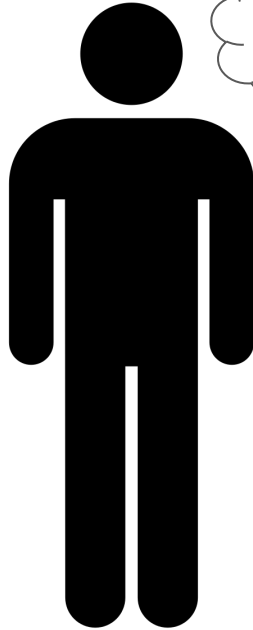
Aggressive Driver Pattern



Car Example

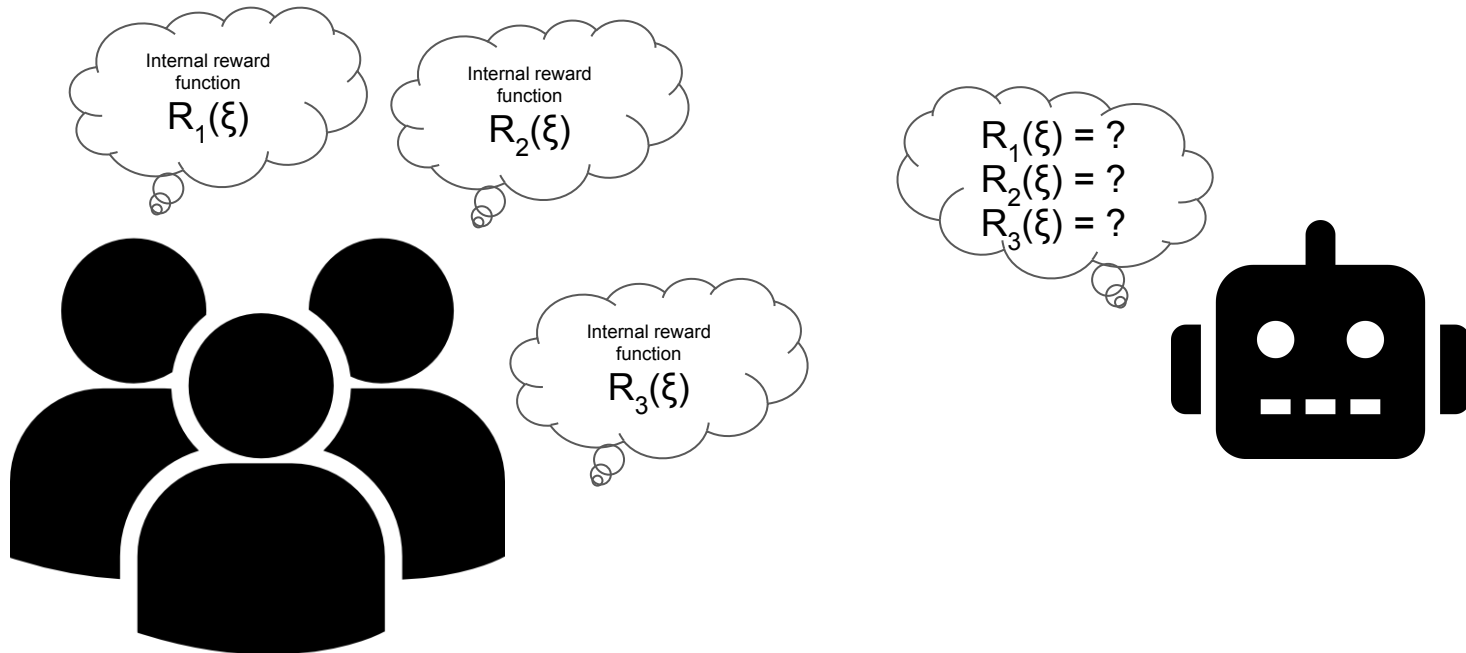
We have data from one type of driver \Rightarrow Unimodal

Timid
OR
Aggressive



Car Example

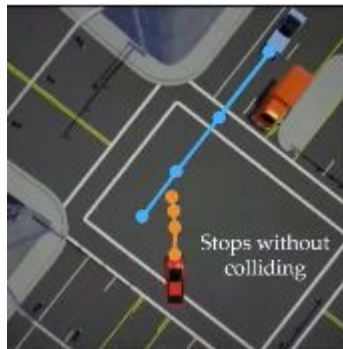
But what if we had data from multiple different kinds of drivers?



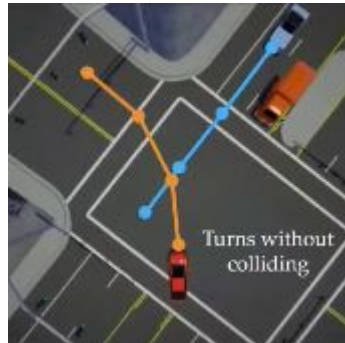
And we don't know which type of driver corresponds to what data

Standard learning from comparison techniques

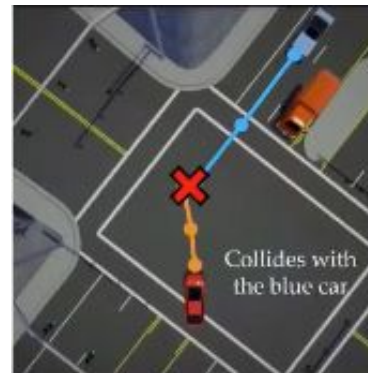
Timid Driver Pattern



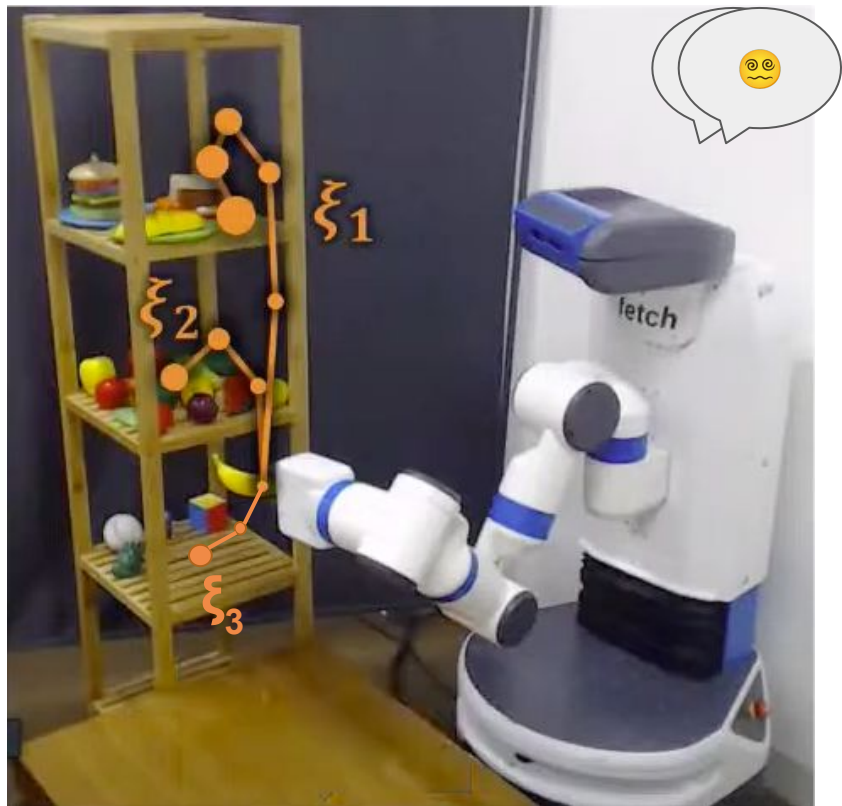
Aggressive Driver Pattern



The car would have an accident trying to find a policy close enough to both the drivers



Example Two: Shelf Task



Which trajectory would you prefer?

All your reasonings may be different based on what is in each of these shelves!

$$\xi_1 > \xi_3 > \xi_2$$



$$\xi_2 > \xi_3 > \xi_1$$



We see why multimodality is important



... but why can't we just label the groups and learn separate reward functions

Example: separate the timid driver's data from the aggressive driver's data

First of all, extremely inefficient to have to separate the data based on reward models

Secondly, it's not accurate to just separate data based on drivers because a driver who is more timid can be aggressive when they are in a hurry

Why can't we just do pairwise comparisons (condt.)





	ξ_1	ξ_2		
	Reward:	2	1	20% of the data vs 80% of the data
	Probability:	0.73	0.27	$\times 0.2$
	Reward:	0	2	
	Probability:	0.12	0.88	$\times 0.8$


$0.24 \xi_1 \succ \xi_2$
 $0.76 \xi_2 \succ \xi_1$

We model the user's comparison with the softmax model

$$P(\xi_1 \succ \xi_2) = \frac{e^{R(\xi_1)}}{e^{R(\xi_1)} + e^{R(\xi_2)}}$$

Let's come up with another set of users

		ξ_1	ξ_2		
	Reward:	2	1	$\times 0.2$	0.24 $\xi_1 > \xi_2$ 0.76 $\xi_2 > \xi_1$
	Probability:	0.73	0.27		
	Reward:	0	2	$\times 0.8$	0.24 $\xi_1 > \xi_2$ 0.76 $\xi_2 > \xi_1$
	Probability:	0.12	0.88		
	Reward:	0	3	$\times 0.13$	0.24 $\xi_1 > \xi_2$ 0.76 $\xi_2 > \xi_1$
	Probability:	0.05	0.95		
	Reward:	0	1	$\times 0.87$	0.24 $\xi_1 > \xi_2$ 0.76 $\xi_2 > \xi_1$
	Probability:	0.27	0.73		



We know that these two groups of users are different, but to the robot, they are indistinguishable and we would not be able to identify the true reward... this is the problem.

Related Work + Limitations

Reward Learning in Robotics (Ng and Russell) → Learning from demonstration

Limitation → Difficult to provide expert demos in robotics

Preference Based Learning → Bradley Terry, Multinomial logits (MNL), Plackett-Luce, Mallows

Limitation → All works published based on these models focus on unimodal cases

Mixture Models → Mix of MNLs, Plackett-Luce, Mallows

Limitation → Assume latent state dynamic that transition between modes or learn the different modes from labelled data

So this paper was the first (and currently one of the only) papers tackling multimodality in reward functions!

Problem Formulation

Setup → consider a fully observable deterministic dynamical system

$$\xi = (s_0, a_0, \dots, s_T, a_T)$$

A trajectory ξ is a series of states and actions $\xi = (s_0, a_0, \dots, s_T, a_T)$

Assume there is a set of M reward functions and we refer to each reward function as an expert

Common Linearity Assumption in reward learning

Each preference modeled as linear reward function over a known fixed feature space Φ

$$R_m(\xi) = \omega_m^\top \Phi(\xi) \Rightarrow \text{with respect to the } m\text{th expert}$$

There exists an unknown distribution over the reward parameters

represent this distribution with mixing coefficients α_m such that $\sum_M^{m=1} \alpha_m = 1$

Learn reward functions and mixing coefficients using Ranking Queries!

Problem Formulation

Ranking Model \Rightarrow using robot example



Users asked to rank all these trajectories and the robot will be given back a **set of trajectory rankings** (coming from M humans) \Rightarrow and we want to learn the reward functions

Problem Formulation

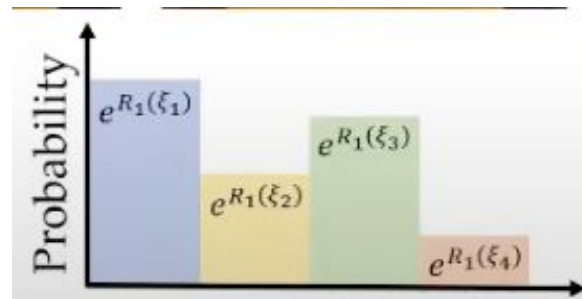
The response to ranking query $x = (\xi_{a_1}, \dots, \xi_{a_K})$ where a_1 is the index of the expert's top choice, ...



Let's say the first user responds to the query:

We have our probability distribution generated with the softmax rule

$$\Pr(x_1 = \xi_{a_1} \mid R = R_m) = \frac{e^{R_m(\xi_{a_1})}}{\sum_{j=1}^K e^{R_m(\xi_{a_j})}}.$$

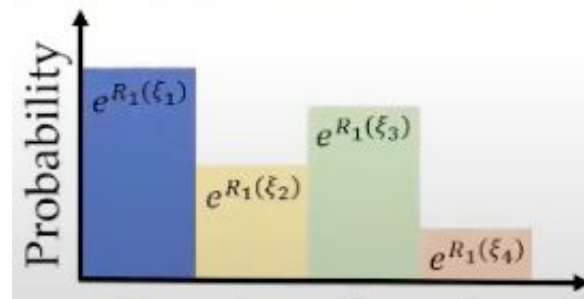
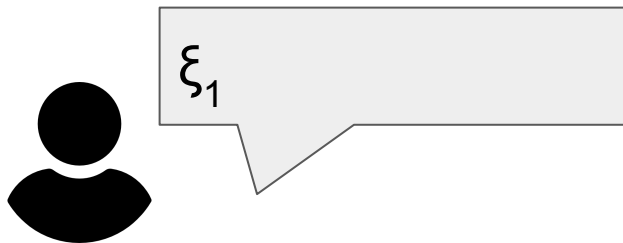


Problem Formulation



User noisily chooses best option:

Randomly sample distribution to pick top choice

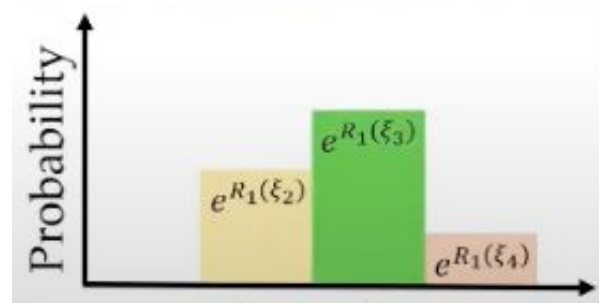
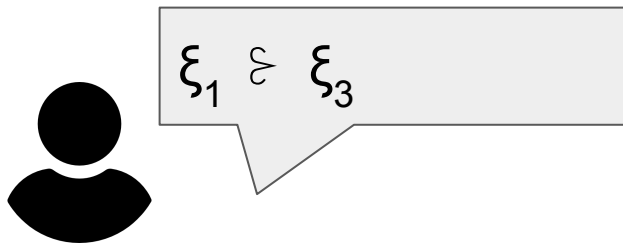


Problem Formulation



User noisily chooses second best option:

Randomly sample distribution to pick top choice from remaining

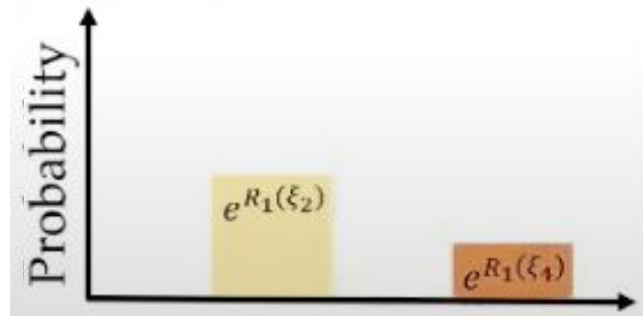
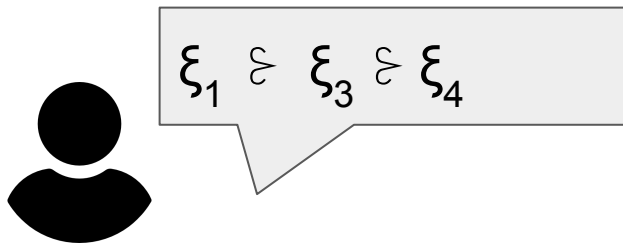


Problem Formulation



User noisily chooses third best option:

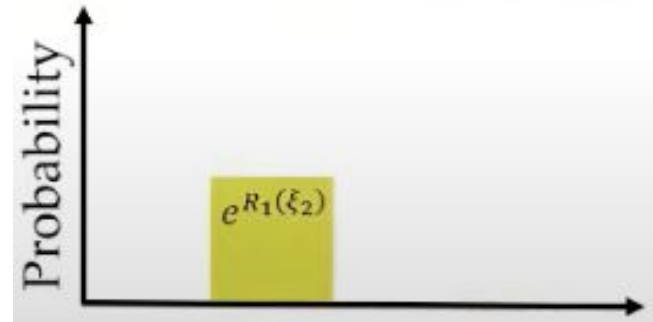
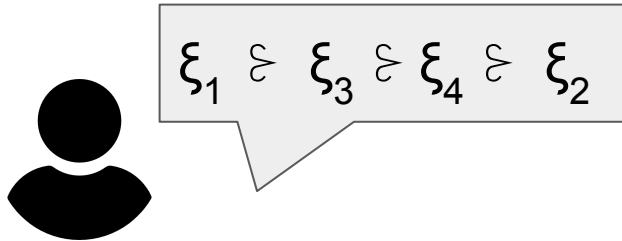
Randomly sample distribution to pick top choice from remaining



Problem Formulation



User chooses last option:



This is called the Plackett-Luce Ranking Model!

So given knowledge of the true reward function weights ω_m and mixing coefficients α_m , we have the following joint mass over observations x from a query Q :

$$\Pr(x \mid Q) = \sum_{m=1}^M \alpha_m \prod_{i=1}^K \frac{e^{\omega_m^\top \Phi(\xi_{a_i})}}{\sum_{j=i}^K e^{\omega_m^\top \Phi(\xi_{a_j})}} \cdot$$

Objective

Goal is to present user with best set of queries that learn reward weights, ω , and mixing coefficient, α , based upon user rankings of preferred query responses

Learning Parameters with Bayesian Learning

Define: $\Theta = \alpha, \omega$

$$\begin{aligned}\Pr(\Theta|Q^{(1)}, x^{(1)}, Q^{(2)}, x^{(2)}, \dots) &\propto \Pr(\Theta) \Pr(Q^{(1)}, x^{(1)}, Q^{(2)}, x^{(2)}, \dots | \Theta) \\ &= \Pr(\Theta) \prod \Pr(x^{(t)}, Q^{(t)} | \Theta, Q^{(1)}, x^{(1)}, \dots, Q^{(t-1)}, x^{(t-1)}) \\ &\propto \Pr(\Theta) \prod_t \Pr(x^{(t)} | \Theta, Q^{(t)})\end{aligned}$$

Assumptions:

1. Ranked queries conditionally independent given parameters
2. Queries at timestamp t conditionally independent on the parameters given previous queries & rankings.

Computing Posterior Final Details

1. Prior Distribution: weights have standard gaussian distribution and mixing coefficients have uniform prior
2. Use maximum likelihood estimation to compute the parameters with the posterior from before.

Types of Queries Presented Matter

- The posterior implies nature of the queries influences the kind of responses you get.
 - The more informative the queries, the faster you'll learn or reduce uncertainty about Θ .
- Robotic queries are generally very costly

Need to find a way to minimize the number of queries!

Solution: Active Querying to Maximize Information Gain

- Greedily select the the most informative query at each timestep.
- That is, given previous ranked queries, at timestep t the query selected is

$$Q^* = \arg \max_Q I(X; \Theta | Q, D)$$

$D =$ Past Observations
 $I =$ Mutual Information Function

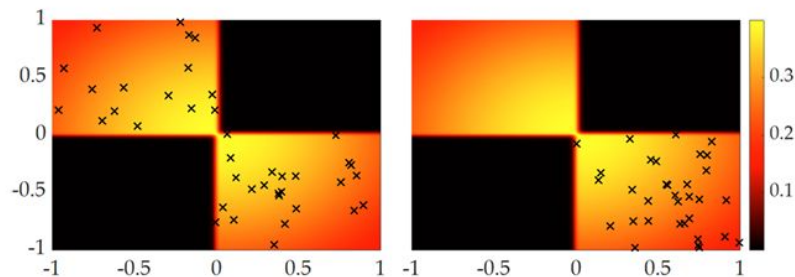
This is equivalent to,

$$Q^* = \arg \min_Q \mathbb{E}_{P(X, \Theta | Q, D)} \log \frac{\mathbb{E}_{\theta' \sim \Theta | D} \Pr[X = x | Q, \theta']}{\Pr[X = x | Q, \theta]}$$

Practical Details

- **How to sample from multimodal distribution?**

Use Metropolis-Hastings with multiple chains.



- **How to minimize?**

Since the trajectory space is continuous, we need to use simulated annealing.

Benchmarks & Environments

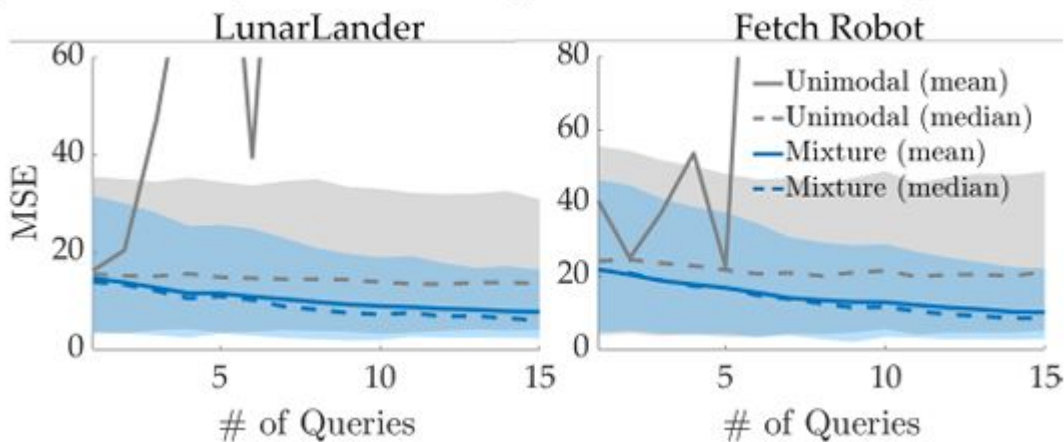
Benchmarks

- Random Query: At each step, the robot selects K random trajectories to present to the user
 - Common baseline in this field.
- Volume Removal: Seeks to maximize difference between prior and unnormalized posterior distribution.

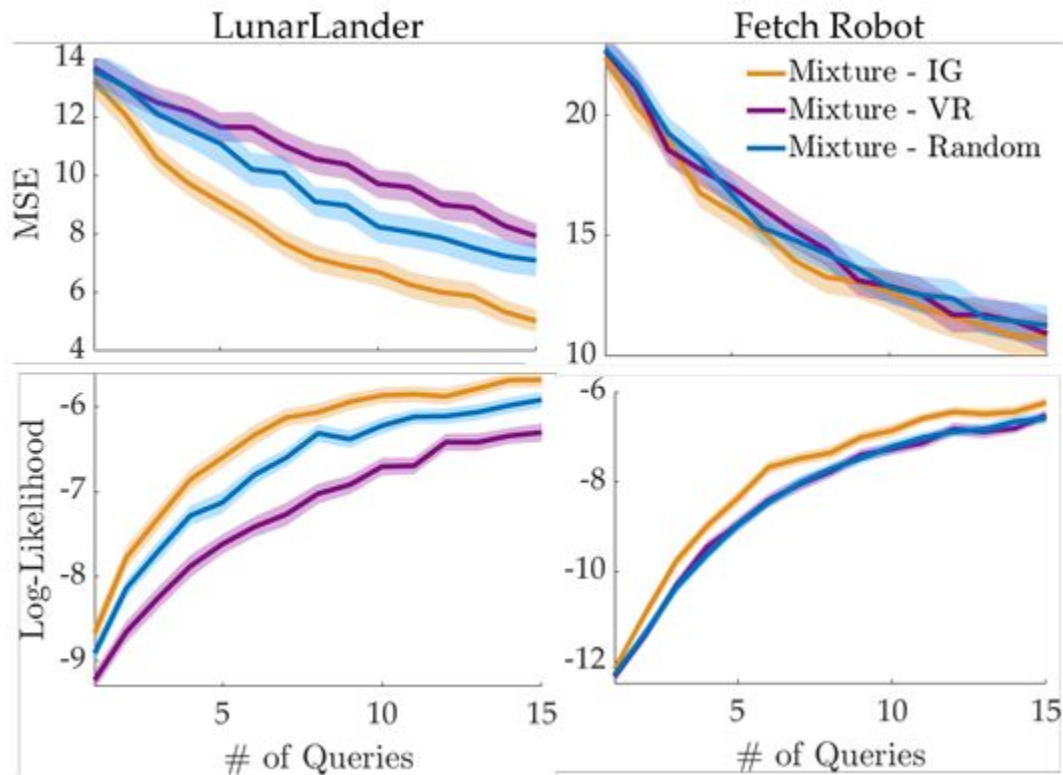
Environments

- LunarLander: Set of 1000 trajectories in OpenAI's LunarLander environment
- Banana Placing: 351 trajectories of robot putting banana into shelf

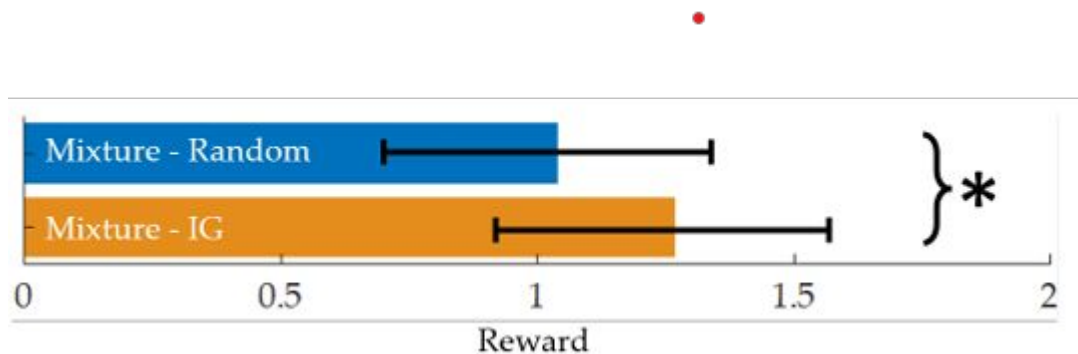
Comparison to Unimodal Rewards



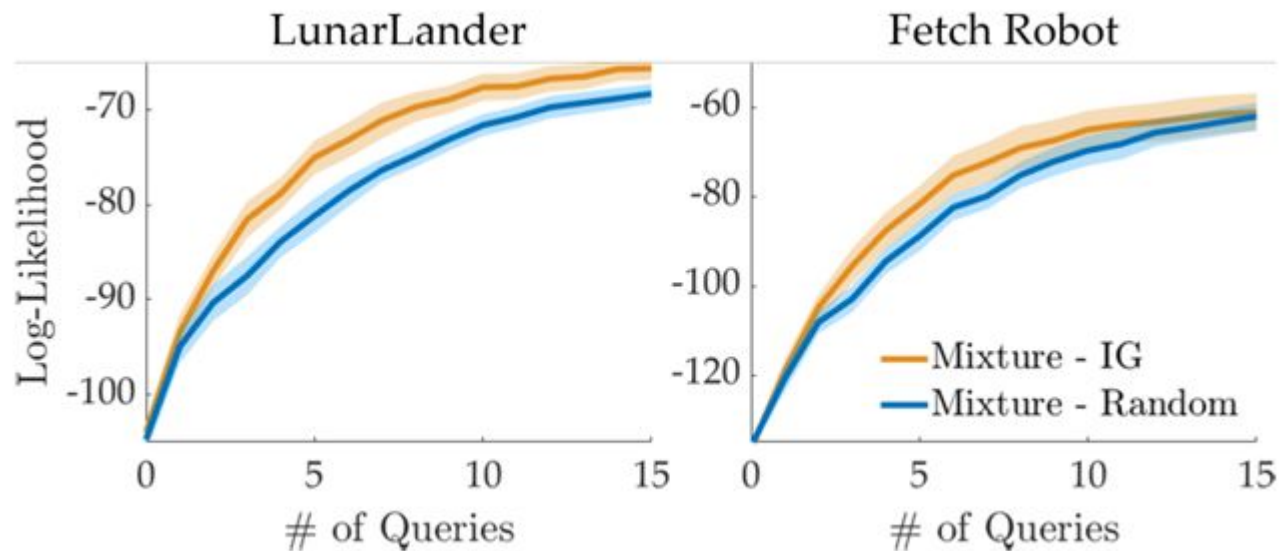
Data Efficiency of Information Gain Approach



Learning Performance of Information Gain Approach



Results on User Studies



Works Cited

1. Hejna, Sadigh. [Few-Shot Preference Learning for Human-in-the-Loop RL](#). CoRL, 2023.
2. Zhou, Jang, Kappler, Herzog, Khansari, Wohlhart, Bai, Kalakrishnan, Levine, Finn. [Watch, Try, Learn: Meta-Learning from Demonstrations and Reward](#). Arxiv, 2019.
3. Myers, Bıyık, Anari, Sadigh. [Learning Multimodal Rewards from Rankings](#). Arxiv, 2021.