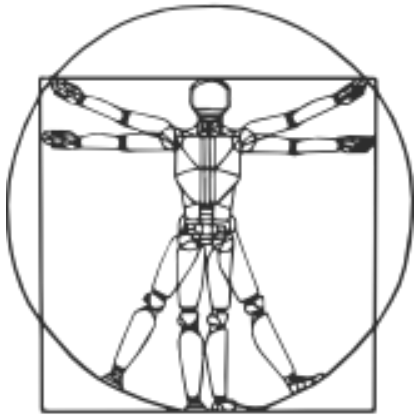


CSC2626
Imitation Learning for Robotics

Florian Shkurti

Week 6: Inverse Reinforcement Learning #1

Today's agenda

- Learning linear rewards from trajectory demonstrations
- Learning nonlinear rewards from trajectory demonstrations
- Updating distributions over reward parameters using preference elicitation
- Combining policy search with reward learning in inverse optimal control

Acknowledgments
Today's slides are based on student presentations from 2019 by: Sergio Casas, Sean Liu, Jacky Liao, Siva Manivasagam

Maximum Entropy Inverse Reinforcement Learning

...

Ziebart, Maas, Bagnell and Dey

Presented by Sergio Casas

Imitation Learning approaches

- In Imitation Learning, we want to learn to **predict the behavior an expert agent would choose.**
- So far, we have seen two main paradigms to tackle this problem

Behavior Cloning

Direct Policy Learning (interactive)

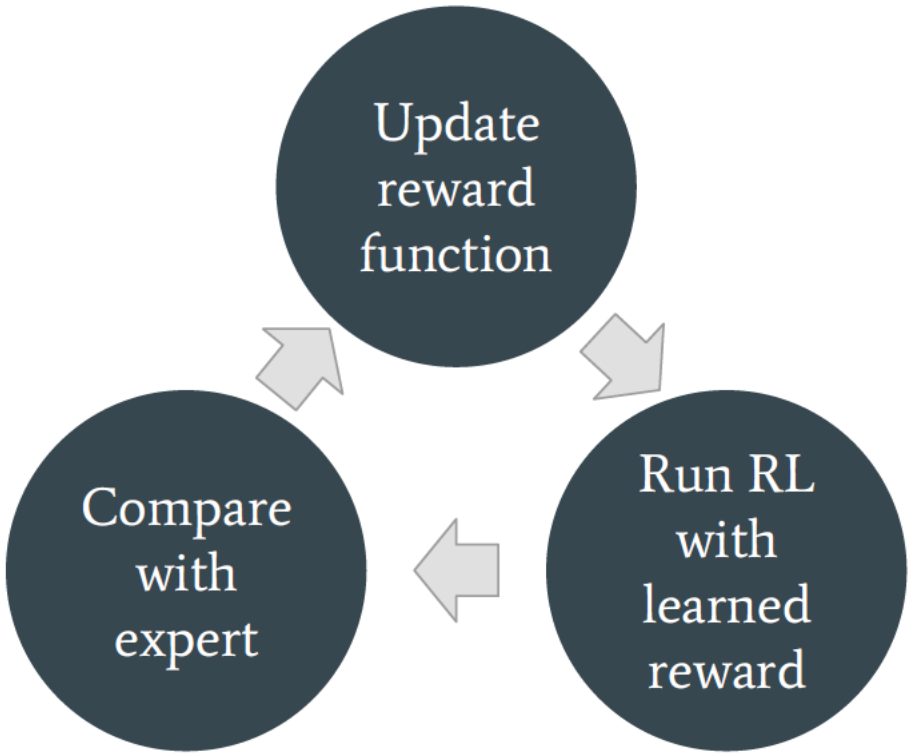
Imitation Learning approaches

- In Imitation Learning, we want to learn to **predict the behavior an expert agent would choose.**
- Today, we introduce a third paradigm: **Inverse Reinforcement Learning (IRL)**

Behavior Cloning	Direct Policy Learning (interactive)	Inverse Reinforcement Learning

Basic principle

- IRL reduces the imitation problem to:
  - **Recovering a reward function** given a set of demonstrations.
  - Solving the MDP using RL to recover the policy, conditioned on our learned reward.
- IRL assumes that the reward function provides the most concise and transferable definition of the task



Background [Ng & Russell 2000, Abbeel & Ng 2004]

- More formally, we want to find a reward function R^* that explains the expert behavior such that:

$$\mathrm{E}\left[\sum_t R^*\left(s_t\right)|\pi^*\right] \geq \mathrm{E}\left[\sum_t R^*\left(s_t\right)|\pi\right] \qquad \forall \pi$$

- IRL Challenges:
  - Assumes we know the expert policy π^*, but we only observe sample trajectories
  - Assumes optimality of the expert
  - Assumes we can enumerate all policies
  - Reward function ambiguity (e.g. R=0 is a solution)

Background [Ng & Russell 2000, Abbeel & Ng 2004]

- We define feature expectations (or feature counts) as:

$$\mathbf{f}_\pi = \mathrm{E}\left[\sum_t \mathbf{f}_{s_t} | \pi\right]$$

- Let the reward be a linear function of the state features:

$$R(s) = \theta^\top \mathbf{f}_s$$

- Therefore, we can calculate the expected reward of a policy as:

$$\mathrm{E}\left[\sum_t R\left(s_t\right) | \pi\right] = \mathrm{E}\left[\sum_t \theta^\top \mathbf{f}_{s_t} | \pi\right] = \theta^\top \mathrm{E}\left[\sum_t \mathbf{f}_{s_t} | \pi\right] = \theta^\top \mathbf{f}_\pi$$

Background [Ng & Russell 2000, Abbeel & Ng 2004]

- We can also define the feature counts of a trajectory τ:

$$\mathbf{f}_\tau = \sum_{s_t \in \tau} \mathbf{f}_{s_t}$$

- And the expected empirical feature count from m sample trajectories of a policy:

$$\tilde{\mathbf{f}}_\pi = \frac{1}{m} \sum_i \mathbf{f}_{\tau_i}$$

- Finally, we can obtain an unbiased estimate of the expected reward of a policy as:

$$\mathrm{E}\left[\sum_t R\left(s_t\right) | \pi\right] \approx \theta^\top \tilde{\mathbf{f}}_\pi$$

Background [Ng & Russell 2000, Abbeel & Ng 2004]

- Therefore, we can rewrite our inequality as:

$$\theta^{*\top} \mathbf{f}_{\pi^*} \geq \theta^{*\top} \mathbf{f}_{\pi}$$

which can in turn be **approximated** when having a dataset D of expert demonstrated trajectories D as:

$$\theta^{*\top} \mathbf{f}_D \geq \theta^{*\top} \mathbf{f}_{\pi} \quad \text{where} \quad \mathbf{f}_D = \tilde{\mathbf{f}}_{\pi^*}$$

- By sampling expert trajectories to compute the feature count estimate we **tackle the challenge of the partial observability** of the expert policy.

Background [Ng & Russell 2000, Abbeel & Ng 2004]

- Observation in [Abbeel & Ng 2004]: for a policy π to be guaranteed to perform as well as the expert policy π*, it suffices that the feature expectations match:

$$\left\| \mathbf{f}_{\pi} - \mathbf{f}_{\pi^*} \right\|_1 \leq \epsilon$$

  which implies that for all $\boldsymbol{\theta}$ with $\|\theta\|_\infty \leq 1$:

$$\left| \theta^{*\top} \mathbf{f}_{\pi} - \theta^{*\top} \mathbf{f}_{\pi^*} \right| \leq \epsilon$$

- This method **removes the need of enumerating all policies** and **tackles the challenge of suboptimal expert policies**. It also comes with nice guarantees about execution time and number of demonstrations.

Background summary

- [Ng & Russell 2000] Algorithms for Inverse Reinforcement Learning
  - First MDP formulation of IRL
  - **Reward ambiguity** pointed out: each policy can be optimal for many reward functions (e.g. zero-policy is always optimal)
- [Abbeel & Ng 2004] Apprenticeship Learning via IRL
  - Strategy of matching feature expectations to the empirical feature mean of the expert policy
  - **Matching of feature counts are ambiguous too:** many different policies satisfy feature matching
- [Ratliff, Bagnell & Zinkevich 2006] Structured Maximum Margin Prediction
  - Non-ambiguous formulation by directly measuring disagreement between the expert and learned policy to learn the reward.
  - Fails when no reward function makes demonstrated behavior optimal and significantly better

Maximum Entropy IRL [Ziebart et al. 2008]

- Let's recap the IRL Challenges:
  - Assumes we know the expert policy π^*
  - Assumes optimality of the expert
  - Assumes we can enumerate all policies
  - **Reward function ambiguity** (e.g. R=0 is a solution)

- Principle of Maximum Entropy [Jaynes 1957]

$$\begin{aligned} \text{argmax}_P \quad & -\sum_\tau P(\tau|\theta) \log P(\tau|\theta) \\ \text{s.t.} \quad & \sum_\tau P(\tau|\theta)\, \mathbf{f}_\tau = \mathbf{f}_{\pi^*} \end{aligned}$$

$$P(\tau|\theta) = \frac{1}{Z(\theta)} \exp\left(\theta^\top \mathbf{f}_\tau\right)$$

Trajectory reward

Maximum Entropy IRL [Ziebart et al. 2008]

- Applying the principle of maximum entropy **breaks the ambiguity of the reward function.**
- Leads us to a distribution over behaviors constrained to match feature expectations of the demonstrations while **having no preference to any particular path that fits this constraint.**

Deterministic MDPs [Ziebart et al. 2008]

- The resulting distribution over paths is parametrized by the reward weights θ

$$P(\tau|\theta) = \frac{1}{Z(\theta)} \exp\left(\theta^\top \mathbf{f}_\tau\right) = \frac{1}{Z(\theta)} \exp\left(\sum_{s_t \in \tau} \theta^\top \mathbf{f}_{s_t}\right)$$

- **Plans with equivalent rewards have equal probabilities**, and plans with higher rewards are exponentially more preferred.

- The partition function always converges for finite horizon problems and infinite horizon problems with discounted reward weights.

    - Can fail in infinite horizon problems with zero-reward absorbing states. However, the partition function would still converge given demonstrated trajectories that are absorbed in a finite number of steps.

Learning from demonstrations [Ziebart et al. 2008]

- As we have seen, maximizing the entropy subject to the feature counts constraint is equivalent to maximize the likelihood of the demonstrated trajectories D with an exponential family as our path distribution:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, L(\theta) = \underset{\theta}{\operatorname{argmax}} \sum_{\tilde{\tau} \in D} \log P(\tilde{\tau}|\theta, T)$$

- For deterministic MDPs, this function is convex and can be optimized using gradient descent:

$$\nabla L(\theta) = \underbrace{\tilde{\mathbf{f}}}_{} - \sum_{\tau} P(\tau|\theta, T)\mathbf{f}_\tau = \tilde{\mathbf{f}} - \sum_{s_i} \underbrace{\mu_{s_i}}_{} \mathbf{f}_{s_i}$$

In practice we use empirical, sample-based expectations of the expert agent

State visitation distribution

State visitation distribution

- The exponential growth of paths with the MDPs time horizon makes enumeration-based approaches infeasible.
- The authors proposed a **DP algorithm similar to value iteration** to compute the state visitation distribution efficiently.

Algorithm 1 Expected Edge Frequency Calculation

Backward pass

1. Set $Z_{s_i,0} = 1$
2. Recursively compute for N iterations

$$Z_{a_{i,j}} = \sum_k P(s_k|s_i, a_{i,j}) e^{\text{reward}(s_i|\theta)} Z_{s_k}$$

$$Z_{s_i} = \sum_{a_{i,j}} Z_{a_{i,j}}$$

Local action probability computation

3. $P(a_{i,j}|s_i) = \dfrac{Z_{a_{i,j}}}{Z_{s_i}}$

Forward pass

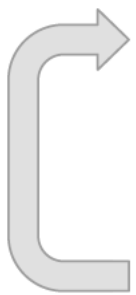
4. Set $\mu_{s_i,t} = P(s_i = s_{\text{initial}})$
5. Recursively compute for $t = 1$ to N

$$\mu_{s_i,t+1} = \sum_{a_{i,j}} \sum_k \mu_{s_k,t} P(a_{i,j}|s_i) P(s_k|a_{i,j}, s_i)$$

Summing frequencies

6. $\mu_{s_i} = \sum_t \mu_{s_{i,t}}$

MaxEnt high-level algorithm



0. Initialize θ, gather demonstrations $\mathcal{D}$
1. Solve for optimal policy $\pi(a|s)$ w.r.t. R_θ with value iteration
2. Solve for state visitation frequencies $\mu(s|\theta, T)$
3. Compute gradient $\nabla_\theta \mathcal{L} = \frac{1}{M} \sum_{\tau_d \in \mathcal{D}} \mathbf{f}_{\tau_d} + \sum_s \mu(s|\theta, T)\mathbf{f}_s$
4. Update θ with one gradient step using $\nabla_\theta \mathcal{L}$

Application: Driver Route Modelling

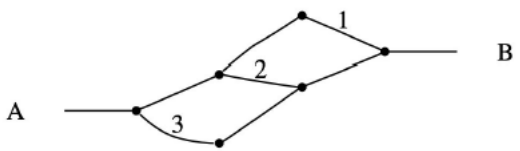
- Interested in **predicting driver behavior** and **route recommendation**
- Pittsburgh's **road network as an MDP**
  - >300,000 states or road segments
  - >900,000 actions or transitions at intersections
- Destination is represented as an absorbing state with zero-cost. Thus, trips with different destinations will have slightly different MDPs
- **Assumption: the reward weights are independent of the goal state**. A single reward weight is then learned from many MDPs that only differ in the goal

Application: Driver Route Modelling

- Dataset
  - GPS data from >100,000 miles and 3,000 hours of taxi trips
  - Fit the sparse GPS data to the road network using a particle filter
  - Segmented the traces into 13,000 trips using a time-based threshold to determine stops
- Path features (low dimensional, 22 counts)
  - Road type: from interstate to local road
  - Speed: from high speed to low speed
  - Lanes: from single-lane to many-lanes
  - Transitions: straight, left, right, hard left, hard right

Application: Driver Route Modelling

- **Maximize the probability of demonstrated paths** using MaxEnt IRL*
- Baselines:
  - **Time-based**: Based on expected time travels. Weights the cost of a unit distance of road to be inversely proportional to the speed of the road.
  - **Max-margin** [Ratliff et al. 2006]: Model capable of predicting new paths, but incapable of density estimation. Directly measures disagreement between the expert and learned policy
  - **Action-based** [Ramachandran et al. 2007, Neu et al. 2007]: The choice of an action is distributed according to the future expected reward of the best policy after taking that action. Suffers from label bias (local distribution of probability mass):



MaxEnt: paths 1, 2, 3 will have 33% probability

Action-based: 50% path 3 , 25% paths 1 and 2

* applied to a “fixed class of reasonably good paths” instead of the full training set

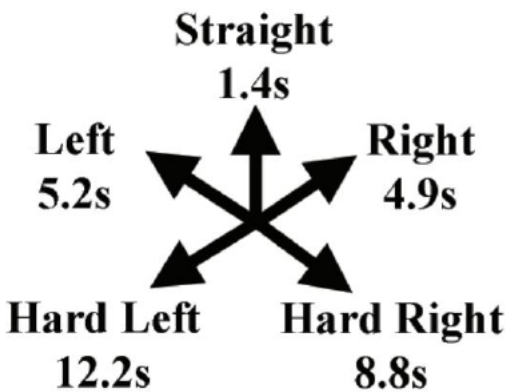
Application: Driver Route Modelling

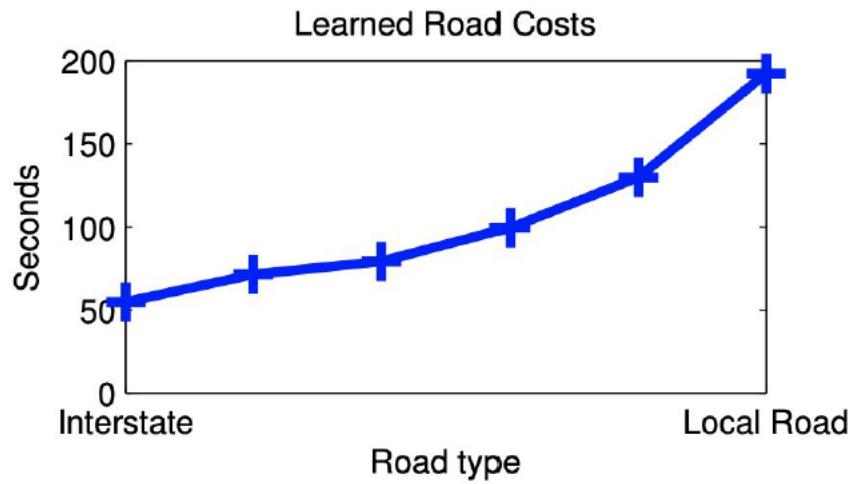
	Matching	90% Match	Log Prob
Time-based	72.38%	43.12%	N/A
Max Margin	75.29%	46.56%	N/A
Action	77.30%	50.37%	-7.91
Action (costs)	77.74%	50.75%	N/A
MaxEnt paths	**78.79%**	**52.98%**	**-6.85**

- **Matching**: Average percentage of distance matching
- **90% Match**: Percentage of examples with at least 90% matching distance
- **Log Prob**: Average log probability

Application: Driver Route Modelling

- Learned costs:
  - Additionally, learned a fixed per edge cost of 1.4 seconds to penalize roads composed of many short paths



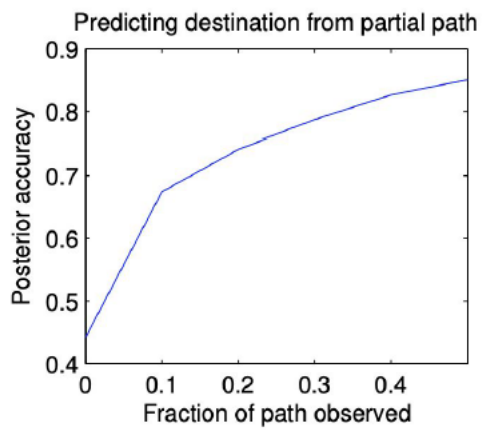


Application: Driver Route Modelling

- **Predicting destination**: so far we have only described situations where the driver intended destination is known. We can use Bayes' rule to predict destination* given our current model.

$$P(\text{dest}|\tilde{\tau}_{A\to B}) \propto P\left(\tilde{\tau}_{A\to B}|dest\right) P(\text{dest})$$

$$\propto \frac{\sum_{\tau_{B\to dest}} e^{\theta^\top \mathbf{f}_\tau}}{\sum_{\tau_{A\to dest}} e^{\theta^\top \mathbf{f}_\tau}} P(dest)$$



* posed as a multiclass classification problem over 5 possible destinations

Reflections

- Solves the reward ambiguity problem by applying the Maximum Entropy theorem, i.e. using path distributions in the exponential family
- SOTA performance and guarantees at the time for linear reward functions
- Derivations assume linear reward function
- MaxEnt IRL requires to know the environment dynamics T (model-given)
- Need to solve full RL problem at each iteration. Only reasonable for small MDPs, i.e. low-dimensional state-action spaces

Follow-up work [Wulfmeier et al. 2016]

- Task: learn cost-maps to navigate in a 2d grid (top-down view of the world)
- Scale to the non-linear reward function to deep neural networks
- Efficient enough to deploy it on a real robot!
- Needs to run full RL at every reward weights update
- Assumes known dynamics (not too bad for navigation) and discrete states and action (simplified, discretized set of motions)
    - Check [Finn et al. 2016] for a proposed workaround

Sean Liu

Large scale cost function learning for path planning using deep inverse reinforcement learning

Wulfmeier et. al (IJRR 2017)

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Motivation for Maximum Entropy Deep Inverse Reinforcement Learning (MEDIRL)

- Non-parametric state-of-the-art IRL methods do not scale well
  - Gaussian Processes based IRL (GPIRL) have runtimes that scale with the size of the dataset
- Maximum Entropy IRL assumes linearity, which is usually not sufficient and limiting
- Goal: design a model tractable for large-scale, real-world autonomous vehicle traversal

Principle of Maximum Entropy for Trajectories

Trajectory:

$$\varsigma = \{(s, a)\}$$

State Feature Vector:

$$f_s$$

Feature Count:

$$f_\varsigma = \sum_{s \in \varsigma} f_s$$

Principle of Maximum Entropy for Trajectories

Result (Principle of Maximum Entropy):

$$\boldsymbol{P}(\zeta|\boldsymbol{R})^* = \frac{\boldsymbol{e}^{\boldsymbol{R}(\zeta)}}{\boldsymbol{Z}}$$

(Probability for user preference along any given trajectory is proportional to the exponential to the reward along its path)

Linear MaxEnt IRL vs. MEDIRL

Gradient of loss function:

$$\frac{\partial L_D}{\partial \theta} = \frac{\partial L_D}{\partial r}\frac{\partial r}{\partial \theta}$$

Linear MaxEnt IRL

Assumption:

$$R(\zeta) = r\left(f_\zeta, \theta\right) = \theta^T f_\zeta$$

$$\frac{\partial L_D}{\partial \theta} = (\boldsymbol{\mu}_{\mathcal{D}} - \mathbb{E}[\boldsymbol{\mu}])\, \boldsymbol{F}_s$$

MaxEnt Deep IRL (MEDIRL)

Assumption:

$$R(\zeta) = FCN$$

(fully convolutional NN)

$$\frac{\partial L_D}{\partial \theta} = \underbrace{(\boldsymbol{\mu}_{\mathcal{D}} - \mathbb{E}[\boldsymbol{\mu}])}_{\text{State Visitation Matching}} \quad \underbrace{\partial r(\boldsymbol{\theta})/\partial \boldsymbol{\theta}}_{\text{Backpropagation}}$$

MEDIRL Main Algorithm

Algorithm 1 MEDIRL

Input: $\boldsymbol{\mu}_D^a, \boldsymbol{f}, \mathcal{S}, \mathcal{A}, T, \gamma, \alpha$

Output: network parameters $\boldsymbol{\theta}^*$

for n = 1 : N **do**

$r^n = r(\boldsymbol{f}, \boldsymbol{\theta}^n)$

planning step

$\boldsymbol{\pi}^n = \text{approx\_value\_iteration}(r^n, \mathcal{S}, \mathcal{A}, T, \gamma)$

$\mathbb{E}[\boldsymbol{\mu}^n] = \text{propagate\_policy}(\boldsymbol{\pi}^n, \mathcal{S}, \mathcal{A}, T)$

determine objective

$\mathcal{L}_D^n = \log(\boldsymbol{\pi}^n) \times \boldsymbol{\mu}_D^a$

$\frac{\partial \mathcal{L}_D^n}{\partial r^n} = \boldsymbol{\mu}_D - \mathbb{E}[\boldsymbol{\mu}^n]$

parameter update

$\frac{\partial \mathcal{L}_D^n}{\partial \boldsymbol{\theta}^n} = \frac{\partial \mathcal{L}_D^n}{\partial r^n} \frac{\partial r^n}{\partial \boldsymbol{\theta}^n}$

$\boldsymbol{\theta}^{n+1} = \boldsymbol{\theta}^n + \alpha \frac{\partial \mathcal{L}_D^n}{\partial \boldsymbol{\theta}^n}$

end for

MEDIRL Main Algorithm

Algorithm 1 MEDIRL

Input: $\mu_D^a, f, \mathcal{S}, \mathcal{A}, T, \gamma, \alpha$
Output: network parameters θ^*

1: **for** n = 1 : N **do**
2: $r^n = r(f, \theta^n)$

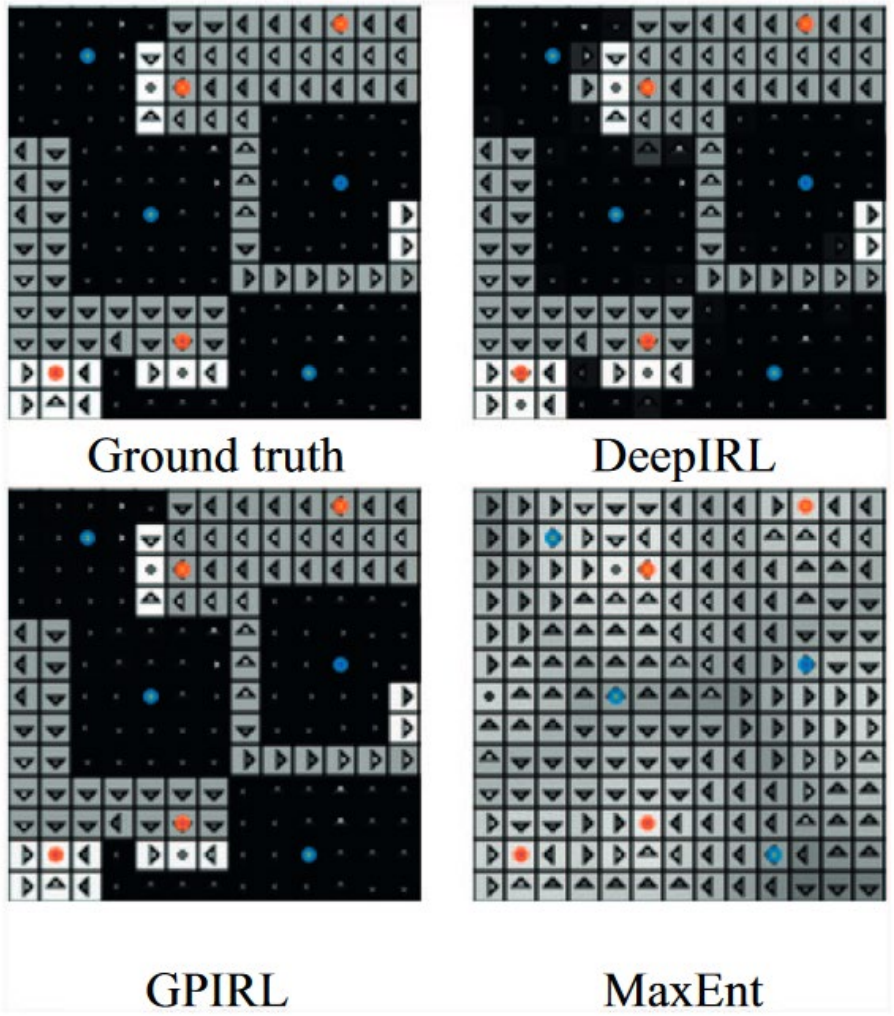
planning step
3: $\pi^n = \text{approx\_value\_iteration}(r^n, \mathcal{S}, \mathcal{A}, T, \gamma)$
4: $\mathbb{E}[\mu^n] = \text{propagate\_policy}(\pi^n, \mathcal{S}, \mathcal{A}, T)$

determine objective
5: $\mathcal{L}_D^n = \log(\pi^n) \times \mu_D^a$
6: $\frac{\partial \mathcal{L}_D^n}{\partial r^n} = \mu_D - \mathbb{E}[\mu^n]$

parameter update
7: $\frac{\partial \mathcal{L}_D^n}{\partial \theta^n} = \frac{\partial \mathcal{L}_D^n}{\partial r^n} \frac{\partial r^n}{\partial \theta^n}$
8: $\theta^{n+1} = \theta^n + \alpha \frac{\partial \mathcal{L}_D^n}{\partial \theta^n}$
9: **end for**

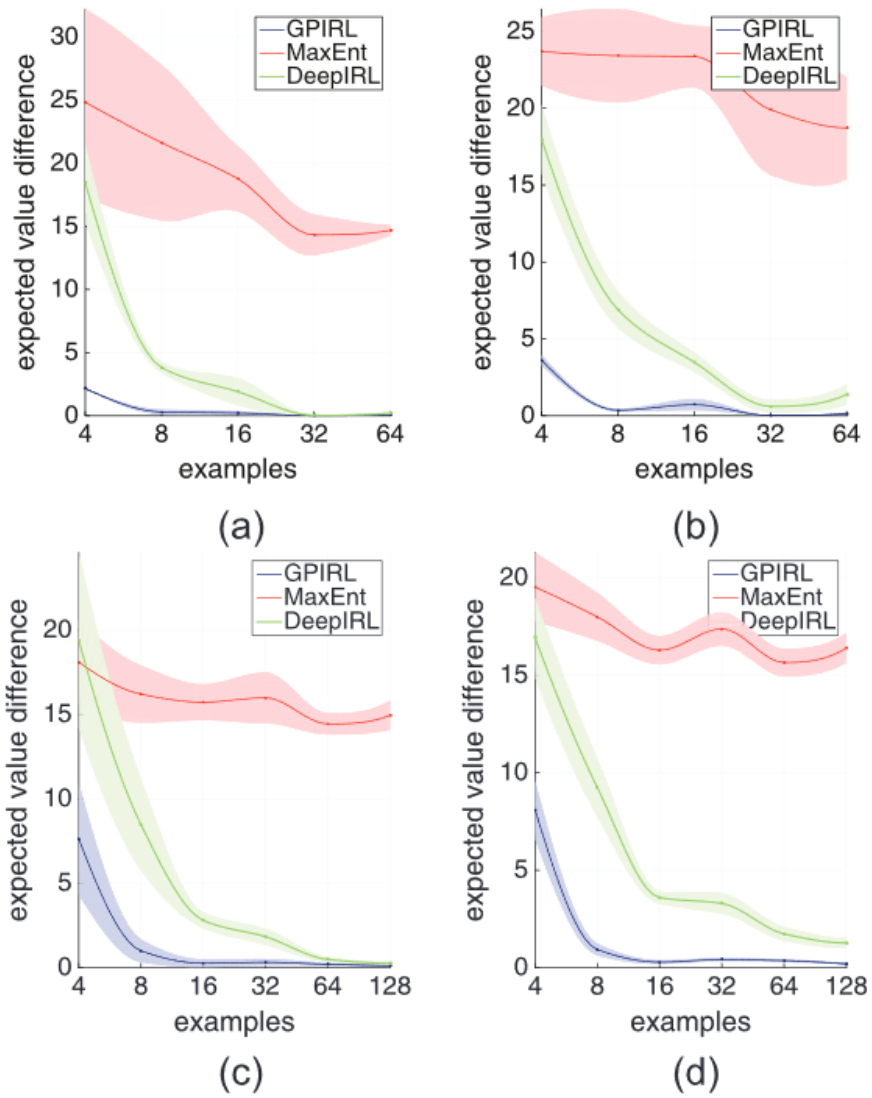
Benchmarking

- Reward function FCN:
  - Two hidden layers
  - ReLU activation
  - 1x1 filter weights
- Evaluation metric: *expected value difference*
- Compared against Linear MaxEnt, GPIRL, NPB-FIRL

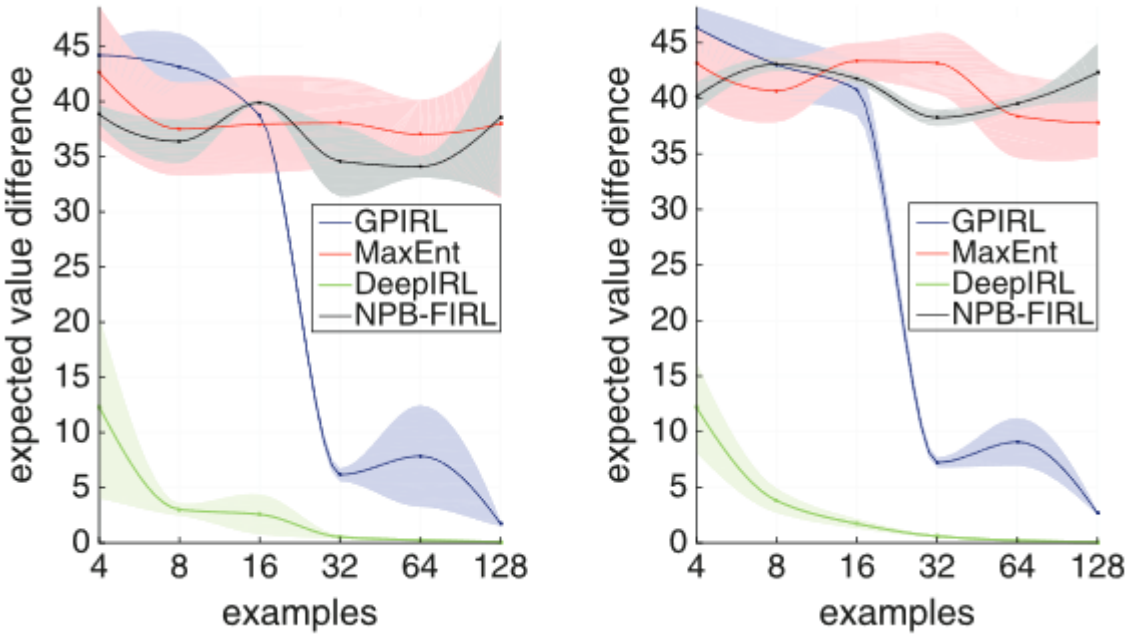


ObjectWorld

Benchmarking

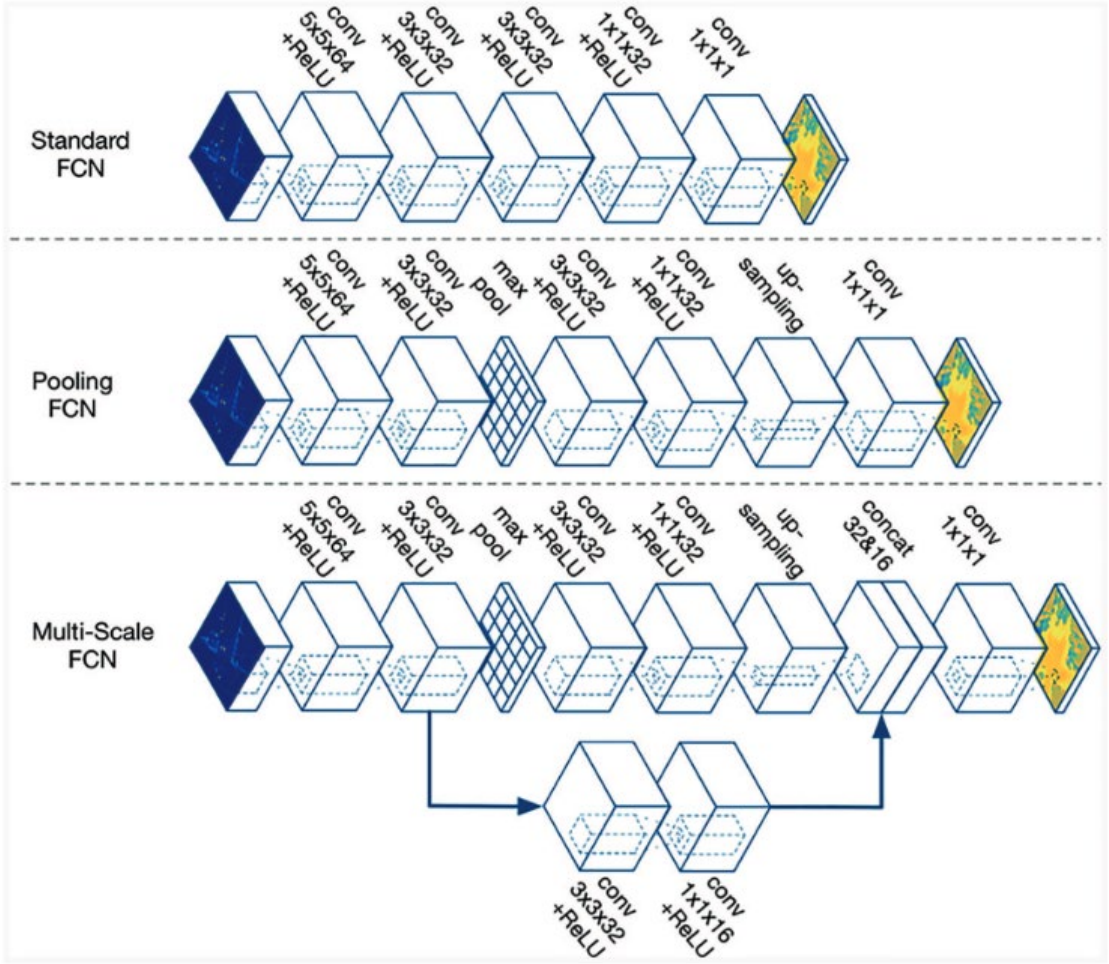


ObjectWorld



BinaryWorld

Proposed Network Architectures



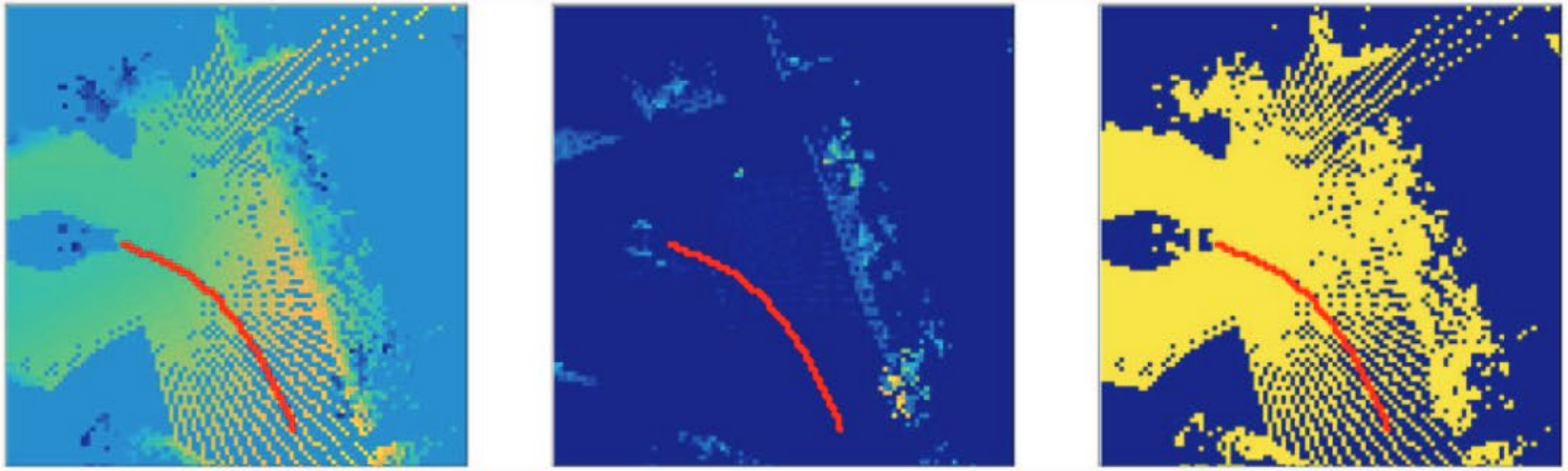
Large-scale Demonstration

- 13 drivers
- >25,000 trajectories 12m-15m long

- Goal: reward map given features
  - Steepness
  - Corner cases (underpasses, stairs)

Mobile research platform: a modified GEM golf cart.

Network Input Data



Visualisation of demonstration trajectories (in red) on all channels of a static map. From left to right: mean height, height variance, cell visibility.

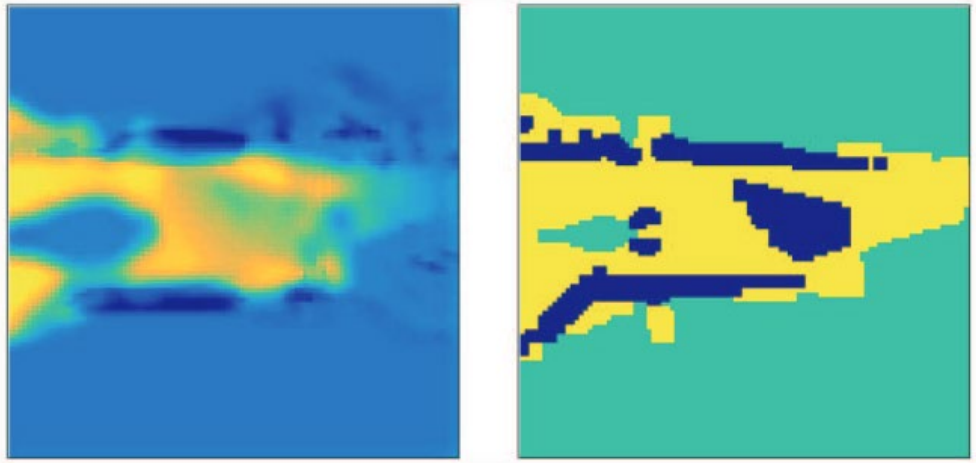
Evaluation

- No absolute ground truth
- Compared against manual cost functions
- Metrics:
  - NLL – negative log-likelihood
  - MHD – Hausdorff distance
  - FNR – False negative rate
  - FPR – False positive rate

Table 1. Model performance on calibrated data.

Metric	NLL	MHD	FNR	FPR
Manual CF	78.13	0.284	0.441	0.000
Standard FCN	69.35	0.221	0.471	0.000
Pooling FCN	69.73	0.230	1.000	0.000
MS FCN	**65.39**	**0.200**	**0.206**	0.000

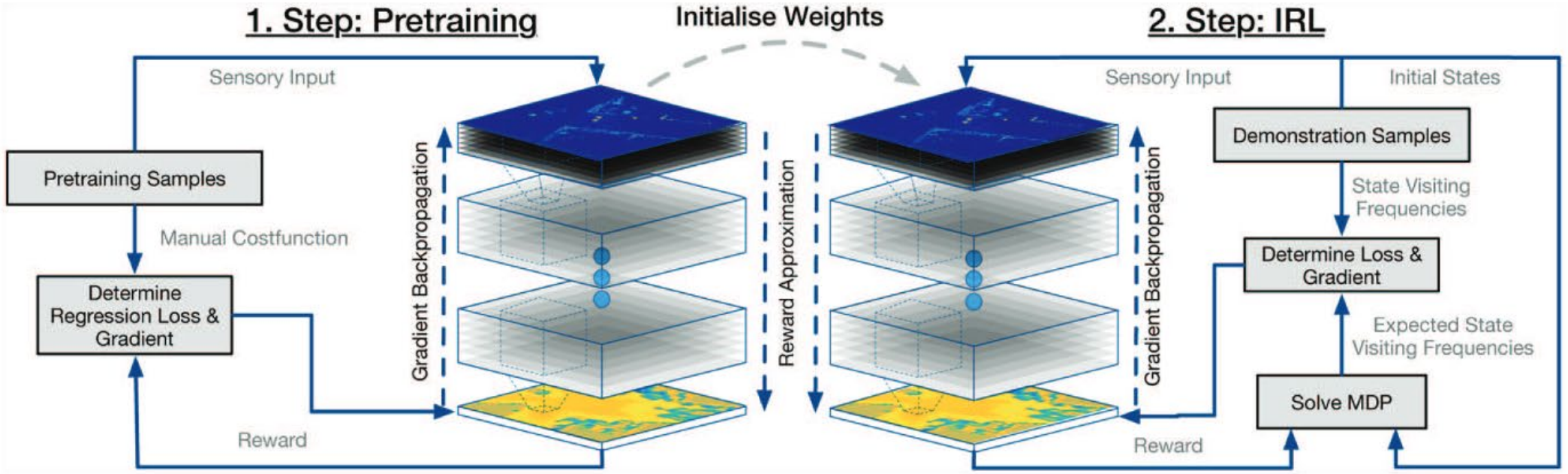
Robustness to Systematic Noise



Example cost maps based on miscalibrated data with MS FCN (left), and the handcrafted cost function (right).

Metric	NLL	MHD	FNR	FPR
Manual CF	89.40	0.432	0.971	0.000
MS FCN	**69.35**	**0.267**	**0.525**	0.000

Pretraining



Schema for additional network pretraining, where the model learns to regress to a manual prior cost map. Subsequently the network is fine-tuned to predict the reward under the MEDIRL framework.

Limitations

- Does not address velocity profiles
- Does not consider temporal consistency between consecutive cost maps
  - Possibly introduce RNNs or temporal convolutions

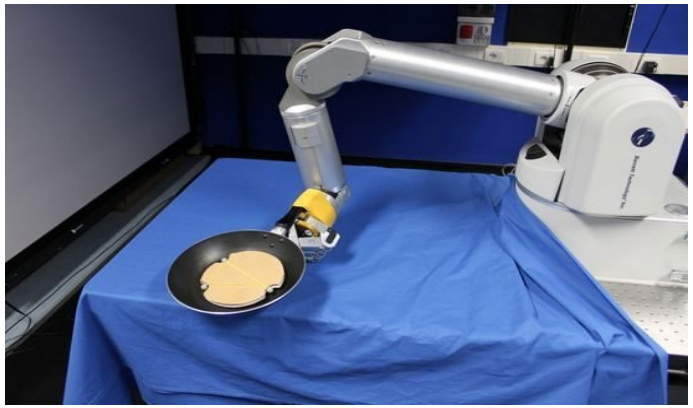
Active Preference-Based Learning of Reward Functions

By: Dorsa Sadigh, Anca D. Dragan, Shankar Sastry, and Sanjit A. Seshia

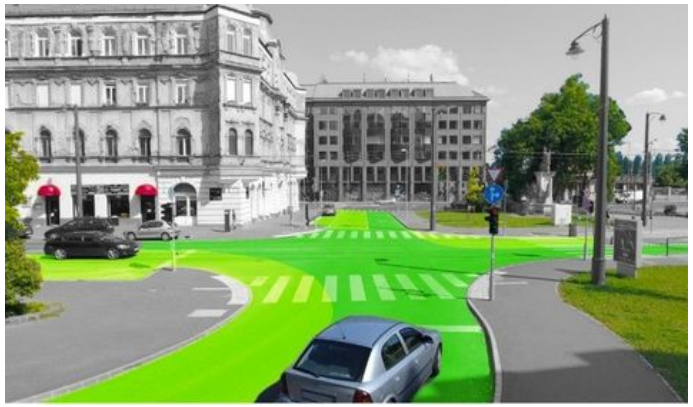
Presented By: Jacky Liao

Solving For Rewards

- How to obtain rewards?
  - Sometimes given (e.g. games)
  - Non trivial for complex tasks

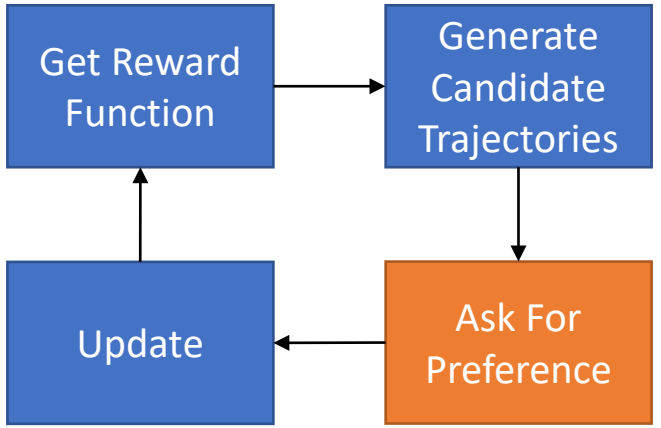


- Inverse RL solves rewards using expert demonstrations
- Expert data not always available
- Problem is too complex or too big



- A new approach: preference based learning

Preference Based Learning

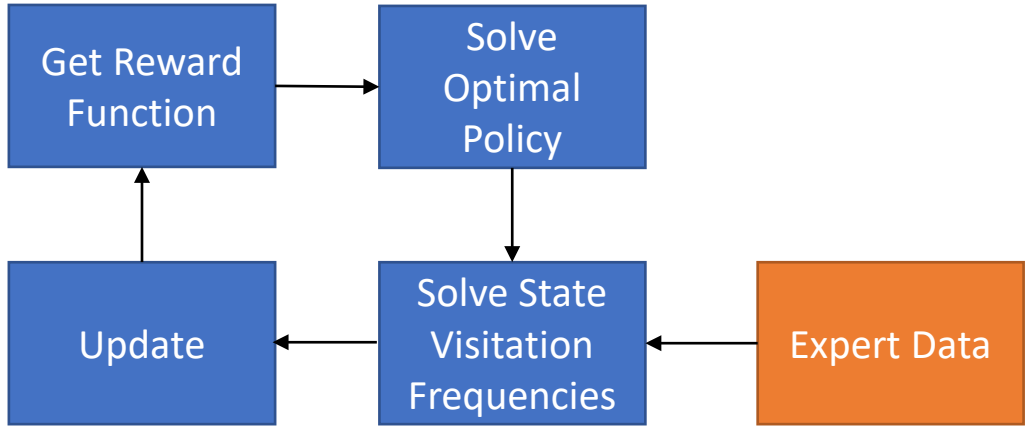


Preference Based Learning Diagram

Learn rewards from expert preference

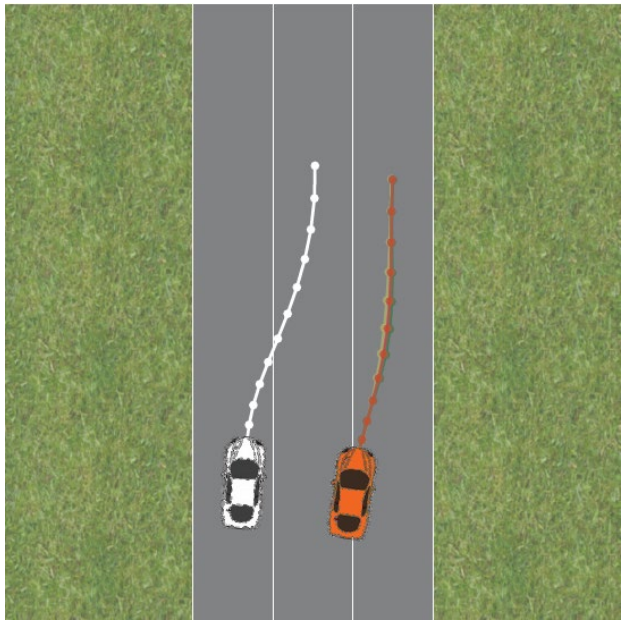
1) Have an estimate of reward function
2) Pick two candidate trajectories
3) Ask the human which trajectory is preferred
4) Use preference as feedback to update reward function

- Preference based learning is active
- Rewards updated directly
  - No inner RL loop
  - No probability estimation required



Inverse RL Diagram

Problem Statement: Autonomous Driving



- 2 Vehicles on the road:
  - Our orange vehicle denoted H
  - Other white vehicle/robot denoted R
- States: (x_H, x_R)
- Inputs: (u_H, u_R)
- Dynamics: $x^{t+1} = f_{HR}(x^t, u_H, u_R)$
- Finite Trajectories: $\xi = \{(x^0, u_H^0, u_R^0), \ldots, (x^N, u_H^N, u_R^N)\}$
- Feasible Trajectories: $\xi \in \Xi$

Reward Function

- Linear Reward Function
  - Features $\phi(x, u_H, u_R) \in \mathbb{R}^d$
    - E.g. matching speed, distance to road boundary, staying within lane
  - Weights $\boldsymbol{w} \in \mathbb{R}^d$
- Reward per time instance:

$$r_H(x^t, u_H^t, u_R^t) = \boldsymbol{w}^T \phi\left(x^t, u_H^t, u_R^t\right)$$

- Sum of rewards over a finite trajectory:

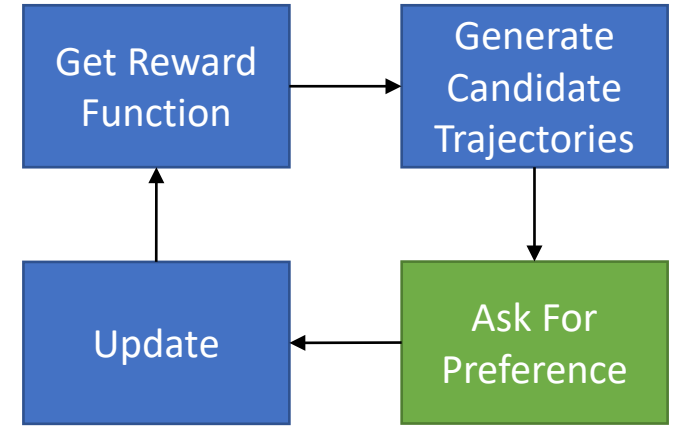
$$R_H(\xi) = R_H(x_0, \boldsymbol{u_H}, \boldsymbol{u_R}) = \sum_{t=0}^{N} r_H\left(x^t,\ u_H^t, u_R^t\right)$$

$$= \boldsymbol{w^T \Phi}(\xi)$$

Preference

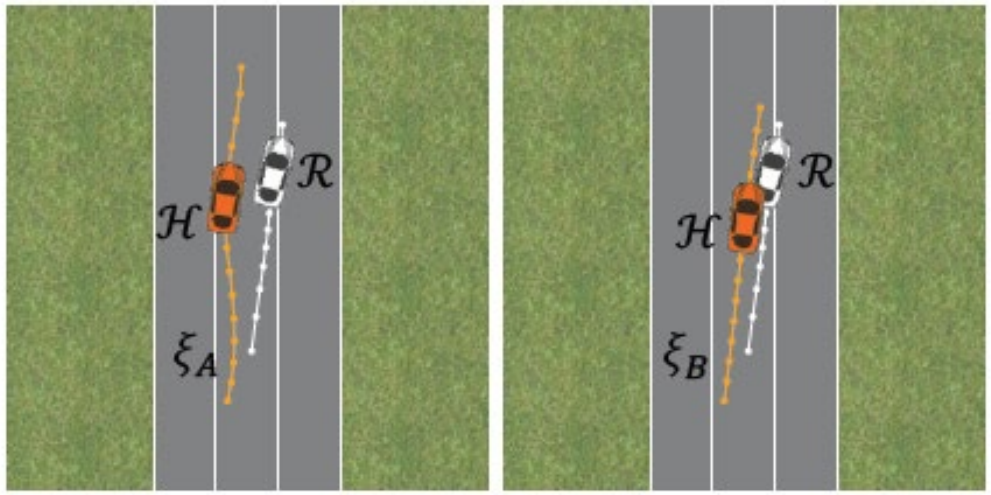
- Given 2 trajectories ξ_A and ξ_B
- Preference variable I

$$I = \begin{cases} +1, & \text{if } \xi_A \text{ is preferred} \\ -1, & \text{if } \xi_B \text{ is preferred} \end{cases}$$

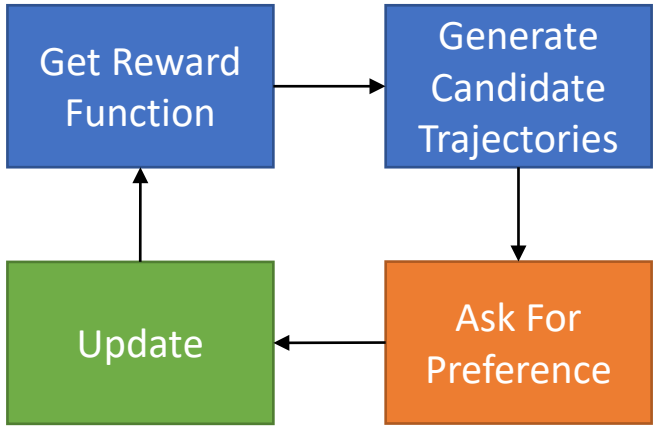


Preference Based Learning Diagram

ξ_A or $\xi_B \rightarrow I$



Weight Update



Preference Based Learning Diagram

- Assume probabilistic model: weights come from a distribution
- Preference is noisy:

$$P(I|\boldsymbol{w}) = \begin{cases} \dfrac{\exp(R_H(\xi_A))}{\exp(R_H(\xi_A)) + \exp(R_H(\xi_B))}, & \text{if } I = +1 \\ \dfrac{\exp(R_H(\xi_B))}{\exp(R_H(\xi_A)) + \exp(R_H(\xi_B))}, & \text{if } I = -1 \end{cases}$$

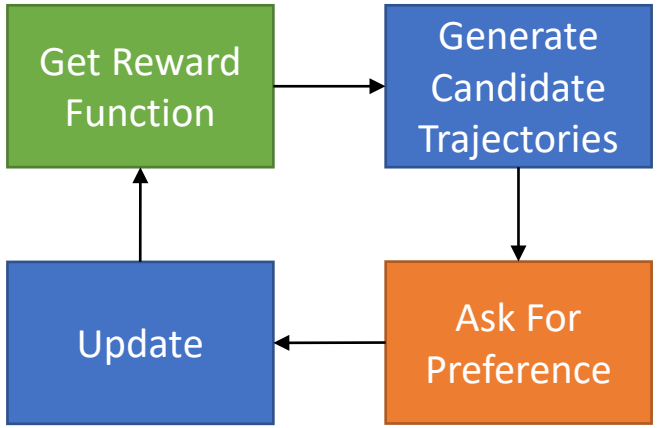
- Some simplification:

$$\varphi = R_H(\xi_A) - R_H(\xi_B), \qquad P(I|\boldsymbol{w}) = \frac{1}{1 + \exp(I\boldsymbol{w}^T\varphi)}$$

- Bayesian update:

$$P(\boldsymbol{w}|I) \propto P(\boldsymbol{w})P(I|\boldsymbol{w})$$

Obtaining Weights



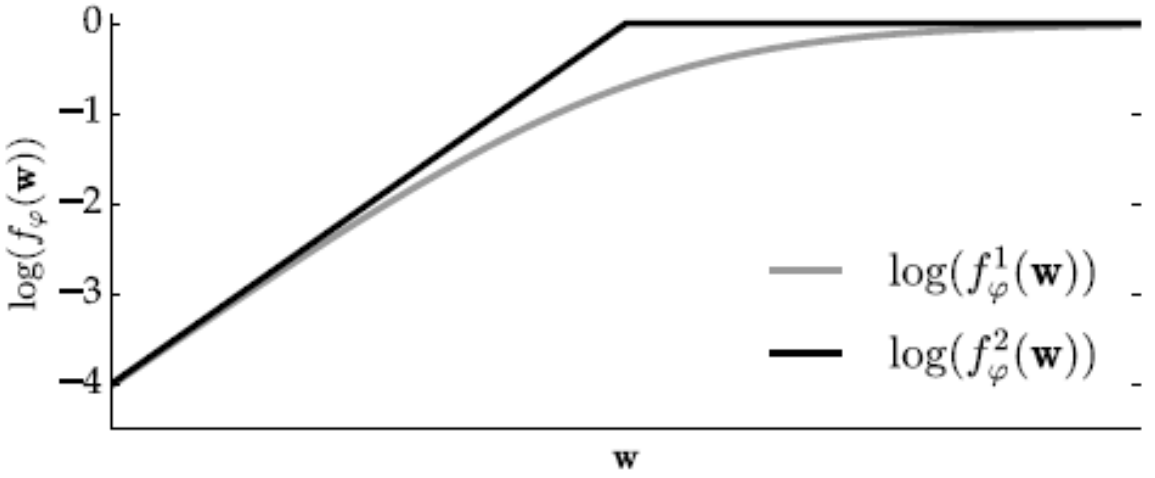
Preference Based Learning Diagram

- $P(\boldsymbol{w}|I) \propto P(\boldsymbol{w})P(I|\boldsymbol{w})$ is often intractable
- Sample using Adaptive Metropolis Algorithm
- Some more simplifications:

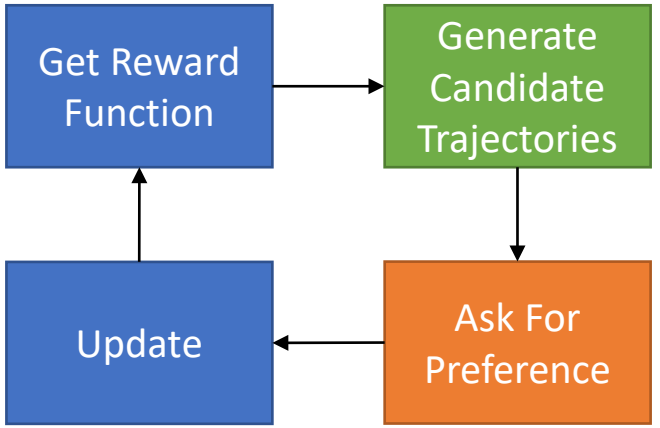
$$P(I|\boldsymbol{w}) = \frac{1}{1+\exp(I\boldsymbol{w}^T\varphi)} \approx \min\left(1, \exp\left(I\boldsymbol{w}^T\varphi\right)\right)$$

- Allows log sampling starting at 0

$$P(w) \approx \frac{1}{M}\sum_{i=0}^{M}\delta(w_i)$$



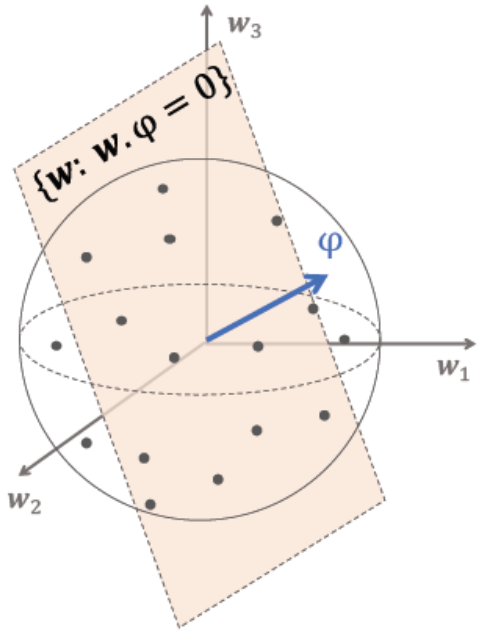
Generate Trajectories



Preference Based Learning Diagram

- Split the trajectory into 2 parts:
  - Scenario $\left(x^0, \boldsymbol{u_R}\right)$
  - Human inputs $\boldsymbol{u_H}$
- Two feasible trajectories: $\boldsymbol{u_H^A}, \quad \boldsymbol{u_H^B}$
- Want each update to give most information
  - Maximize volume removed:

$$\max_{\boldsymbol{u_H^A}, \boldsymbol{u_H^B}} \min(\mathbb{E}_{\boldsymbol{w}}[1 - f_{\varphi}(\boldsymbol{w})], \mathbb{E}_{\boldsymbol{w}}[1 - f_{-\varphi}(\boldsymbol{w})])$$



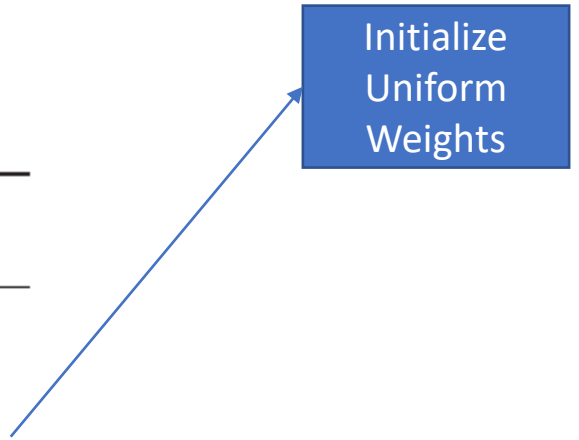
  - Trajectories lie on hyperplane $\boldsymbol{w^T}\varphi = 0$
  - Similar to binary search, find side of hyperplane to update

- Solve the optimization problem using Newton-based methods

Algorithm Summary

Algorithm 1 Preference-Based Learning of Reward Functions

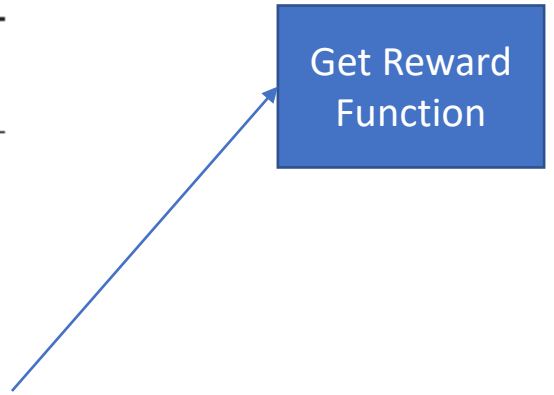
Input: Features ϕ, horizon N, dynamics f, *iter*
Output: Distribution of $\mathbf{w}$: $p(\mathbf{w})$
Initialize $p(\mathbf{w}) \sim \text{Uniform}(B)$, for a unit ball B
While $t < iter$:
  $W \leftarrow M$ samples from AdaptiveMetropolis($p(\mathbf{w})$)
  $(x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B) \leftarrow \text{SynthExps}(W, f)$
  $I_t \leftarrow \text{QueryHuman}(x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B)$
  $\varphi = \Phi(x^0, \mathbf{u}_R, \mathbf{u}_H^A) - \Phi(x^0, \mathbf{u}_R, \mathbf{u}_H^B)$
  $f_\varphi(\mathbf{w}) = \min(1, \exp(I_t \mathbf{w}^\top \varphi))$
  $p(\mathbf{w}) \leftarrow p(\mathbf{w}) \cdot f_\varphi(\mathbf{w})$
  $t \leftarrow t + 1$
End for



Algorithm Summary

Algorithm 1 Preference-Based Learning of Reward Functions

Input: Features ϕ, horizon N, dynamics f, *iter*
Output: Distribution of $\mathbf{w}$: $p(\mathbf{w})$
Initialize $p(\mathbf{w}) \sim \text{Uniform}(B)$, for a unit ball B
While $t < iter$:
    $W \leftarrow M$ samples from AdaptiveMetropolis$(p(\mathbf{w}))$
    $(x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B) \leftarrow \text{SynthExps}(W, f)$
    $I_t \leftarrow \text{QueryHuman}(x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B)$
    $\varphi = \Phi(x^0, \mathbf{u}_R, \mathbf{u}_H^A) - \Phi(x^0, \mathbf{u}_R, \mathbf{u}_H^B)$
    $f_\varphi(\mathbf{w}) = \min(1, \exp(I_t \mathbf{w}^\top \varphi))$
    $p(\mathbf{w}) \leftarrow p(\mathbf{w}) \cdot f_\varphi(\mathbf{w})$
    $t \leftarrow t + 1$
End for



Algorithm Summary

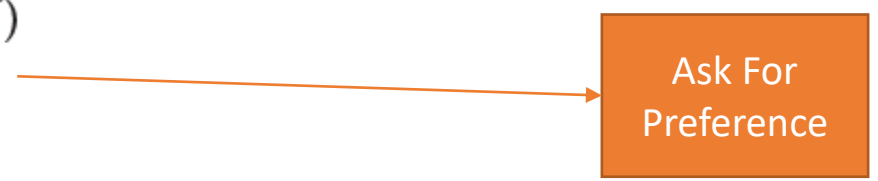
Algorithm 1 Preference-Based Learning of Reward Functions

Input: Features ϕ, horizon N, dynamics f, *iter*
Output: Distribution of $\mathbf{w}$: $p(\mathbf{w})$
Initialize $p(\mathbf{w}) \sim \text{Uniform}(B)$, for a unit ball B
While $t < iter$:
    $W \leftarrow M$ samples from AdaptiveMetropolis$(p(\mathbf{w}))$
    $(x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B) \leftarrow \text{SynthExps}(W, f)$
    $I_t \leftarrow \text{QueryHuman}(x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B)$
    $\varphi = \Phi(x^0, \mathbf{u}_R, \mathbf{u}_H^A) - \Phi(x^0, \mathbf{u}_R, \mathbf{u}_H^B)$
    $f_\varphi(\mathbf{w}) = \min(1, \exp(I_t \mathbf{w}^\top \varphi))$
    $p(\mathbf{w}) \leftarrow p(\mathbf{w}) \cdot f_\varphi(\mathbf{w})$
    $t \leftarrow t + 1$
End for

Algorithm Summary

Algorithm 1 Preference-Based Learning of Reward Functions

Input: Features ϕ, horizon N, dynamics f, *iter*
Output: Distribution of $\mathbf{w}$: $p(\mathbf{w})$
Initialize $p(\mathbf{w}) \sim \text{Uniform}(B)$, for a unit ball B
While $t < iter$:
  $W \leftarrow M$ samples from AdaptiveMetropolis$(p(\mathbf{w}))$
  $(x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B) \leftarrow \text{SynthExps}(W, f)$
  $I_t \leftarrow \text{QueryHuman}(x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B)$
  $\varphi = \Phi(x^0, \mathbf{u}_R, \mathbf{u}_H^A) - \Phi(x^0, \mathbf{u}_R, \mathbf{u}_H^B)$
  $f_\varphi(\mathbf{w}) = \min(1, \exp(I_t \mathbf{w}^\top \varphi))$
  $p(\mathbf{w}) \leftarrow p(\mathbf{w}) \cdot f_\varphi(\mathbf{w})$
  $t \leftarrow t + 1$
End for



Algorithm Summary

Algorithm 1 Preference-Based Learning of Reward Functions

Input: Features ϕ, horizon N, dynamics f, *iter*
Output: Distribution of $\mathbf{w}$: $p(\mathbf{w})$
Initialize $p(\mathbf{w}) \sim \text{Uniform}(B)$, for a unit ball B
While $t < iter$:
    $W \leftarrow M$ samples from AdaptiveMetropolis($p(\mathbf{w})$)
    $(x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B) \leftarrow \text{SynthExps}(W, f)$
    $I_t \leftarrow \text{QueryHuman}(x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B)$
    $\varphi = \Phi(x^0, \mathbf{u}_R, \mathbf{u}_H^A) - \Phi(x^0, \mathbf{u}_R, \mathbf{u}_H^B)$
    $f_\varphi(\mathbf{w}) = \min(1, \exp(I_t \mathbf{w}^\top \varphi))$
    $p(\mathbf{w}) \leftarrow p(\mathbf{w}) \cdot f_\varphi(\mathbf{w})$
    $t \leftarrow t + 1$
End for

Update

Results

- Weights begin with uniform probability
- Convergence after 200 iterations

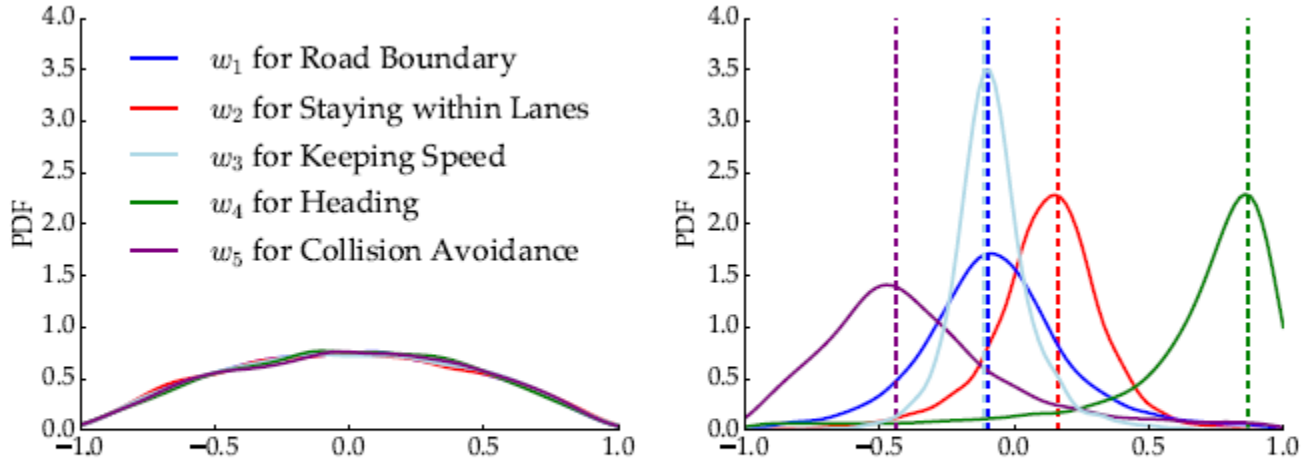
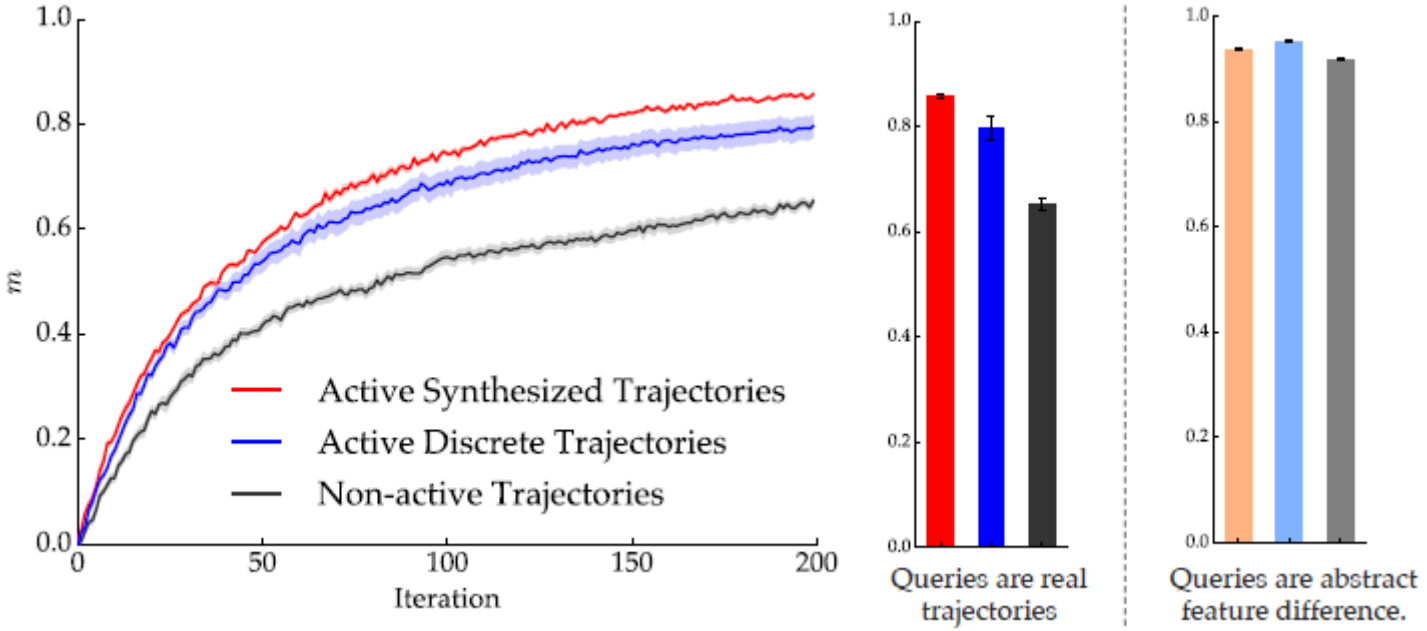


Fig. 3: Distribution over all weights before/after convergence. The dotted lines show the ground truth of the weights.

Results

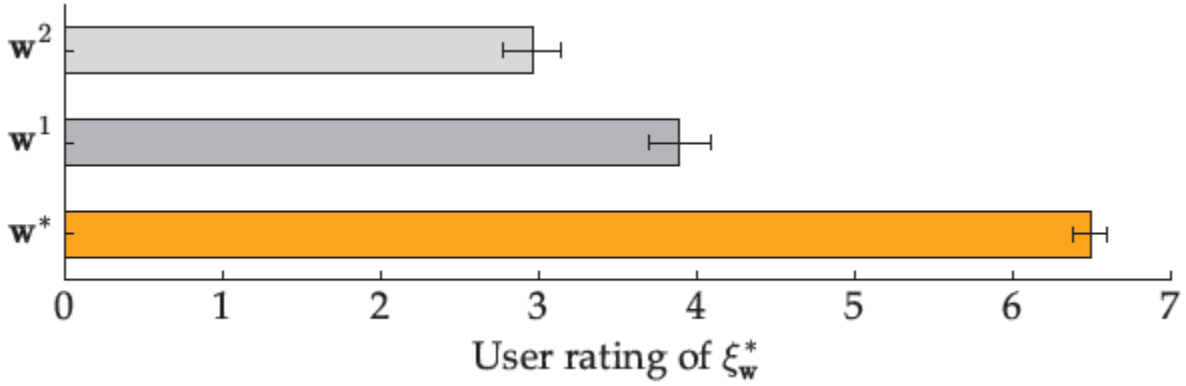
- Rate of convergence, active synthesis is faster!
- Blue curve: generated feasible trajectories not optimized for weight updates
- Black curve: non active trajectories, equivalent to expert dataset
- Lighter colours: training on non feasible trajectories

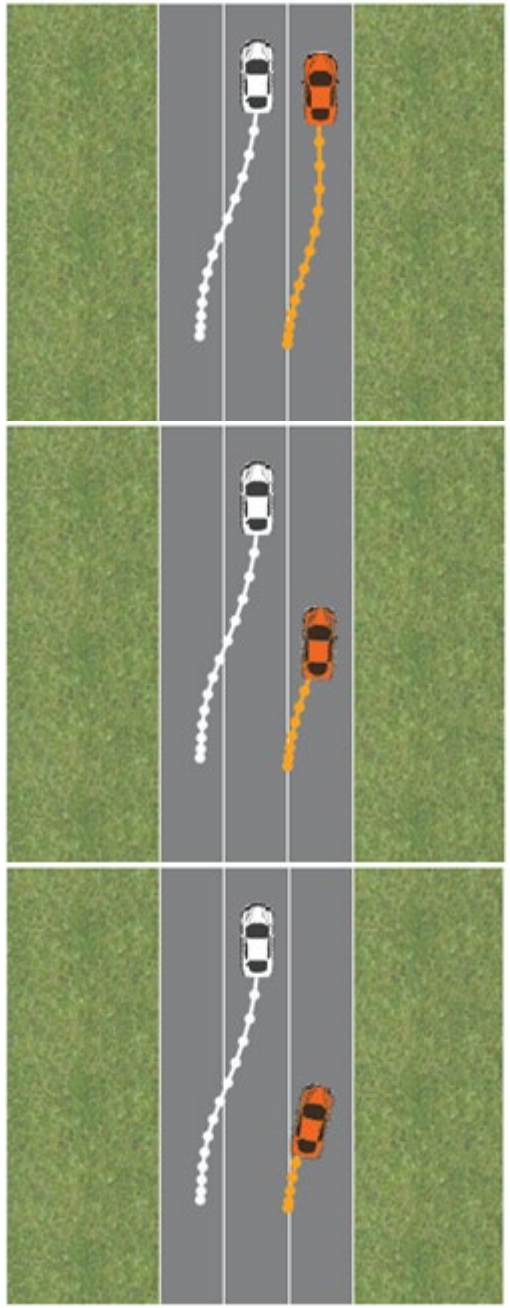
$$m = \mathbb{E}\left[\frac{\mathbf{w} \cdot \mathbf{w}_{\text{true}}}{|\mathbf{w}||\mathbf{w}_{\text{true}}|}\right].$$



Results

- Perturbation of weights
  - Learned weights: $\boldsymbol{w}^*$
  - Slightly perturbed weights: $\boldsymbol{w}^1$
  - Largely perturbed weights: $\boldsymbol{w}^2$
- Users prefer $\boldsymbol{w}^*$





Limitations

- Only 1 other agent
- Linear model: assumes features known
- Humans may act differently, no guarantee of rationality
- No guarantee that behavior can be customized
  - i.e. algorithm converges to a reward function, but not the one the user wants
  - e.g. driving slowly but safely vs faster and aggressive, but more optimal

Direct Loss Minimization Inverse Optimal Control

Andreas Doerr*, Nathan Ratliff*†, Jeannette Bohg†, Marc Toussaint* and Stefan Schaal†‡
*Machine Learning & Robotics Lab, University Stuttgart, Germany
†Max Planck Institute, Autonomous Motion Departement, Tuebingen, Germany
‡University of Southern California, Computational Learning and Motor Control Lab, Los Angeles, CA, USA
andreasdoerr@gmx.net, {ratliffn, jbohg, sschaal}@tue.mpg.de, marc.toussaint@ipvs.uni-stuttgart.de, sschaal@usc.edu

02/15/2019

What is the most compact representation to specify a task?

What is the most compact representation to specify a task?

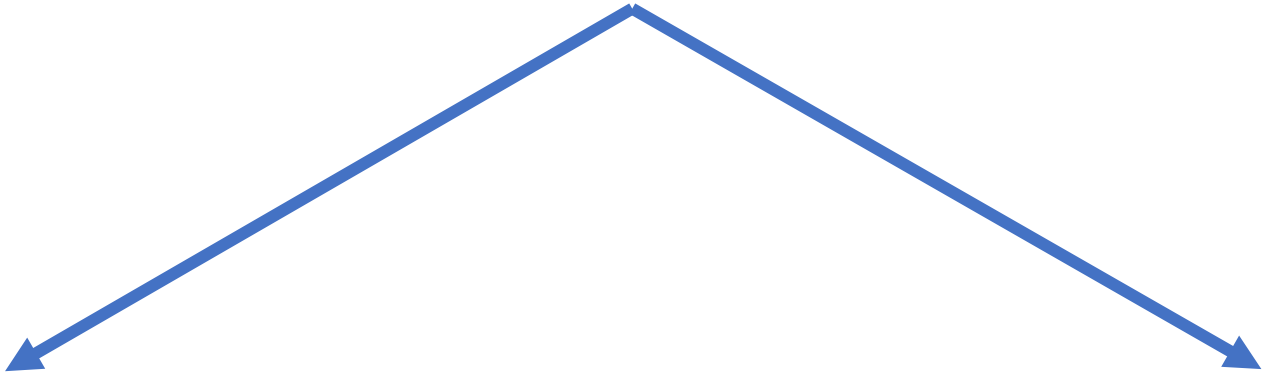
Cost Function

What is the usual cost function in reinforcement learning?

What is the usual cost function in reinforcement learning?

Human-Designed (Reach goal, get most points, avoid obstacles, etc.)

Two views in imitation learning



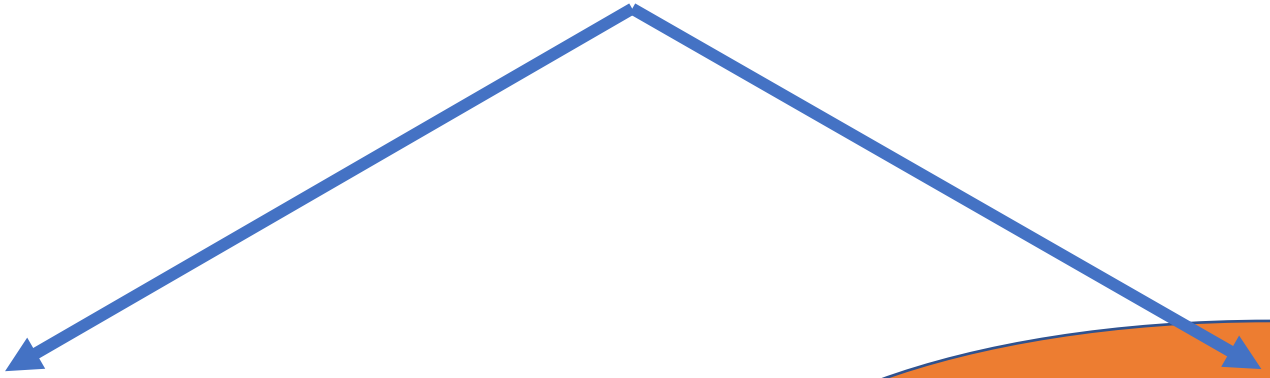
Supervised learning on actions

(e.g. Dagger, usually deterministic policy)

Match the expert distribution

(stochastic policy)

Two views in imitation learning



Supervised learning on actions

(e.g. Dagger, usually deterministic policy)

Match the expert distribution

(stochastic policy)

Direct Loss Minimization Inverse Optimal Control?

Use both!

Key insight: When viewing Imitation Learning as an inverse optimal control problem with the goal of matching an expert distribution, we can leverage reinforcement learning and solve the problem as a policy search problem

Optimal Control

$$\min_{\boldsymbol{u}_{1:T}, \boldsymbol{s}_{1:T+1}} \sum_{t=1}^{T} c_t(\boldsymbol{s}_t, \boldsymbol{u}_t) + \psi(\boldsymbol{s}_{T+1})$$

Cost(state, input)

Cost(terminal state)

$$\text{s.t.} \quad \boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{u}_t).$$

Deterministic Physics Model

Optimal Control

$$\min_{\boldsymbol{u}_{1:T}, \boldsymbol{s}_{1:T+1}} \sum_{t=1}^{T} c_t(\boldsymbol{s}_t, \boldsymbol{u}_t) + \psi(\boldsymbol{s}_{T+1})$$

Cost(state, input)

Cost(terminal state)

How to parameterize cost function?

$$\text{s.t.} \quad \boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{u}_t).$$

Deterministic Physics Model

Optimal Control

Linear combination of non-linear costs

$$\min_{\boldsymbol{u}_{1:T}, \boldsymbol{s}_{1:T+1}} \sum_{t=1}^{T} \left(\sum_{i=1}^{k} \alpha_i c_t^{(i)}(\boldsymbol{s}_t, \boldsymbol{u}_t) \right)$$

Costs(state, input)

$$\text{s.t.} \quad \boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{u}_t).$$

Deterministic Physics Model

Optimal Control

Linear combination of non-linear costs

Expressive, but large number of parameters:
how to determine coefficients?

Costs(state, input)

$$\min_{\boldsymbol{u}_{1:T}, \boldsymbol{s}_{1:T+1}} \sum_{t=1}^{T} \left(\sum_{i=1}^{k} \alpha_i c_t^{(i)} (\boldsymbol{s}_t, \boldsymbol{u}_t) \right)$$

$$\text{s.t.} \quad \boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{u}_t).$$

Deterministic Physics Model

Inverse Optimal Control

Expressive, but large number of parameters:
how to determine coefficients?

Linear combination of non-linear costs

Inverse Optimal Control

Costs(state, input)

$$\min_{\boldsymbol{u}_{1:T}, \boldsymbol{s}_{1:T+1}} \sum_{t=1}^{T} \left(\sum_{i=1}^{k} \alpha_i c_t^{(i)}(\boldsymbol{s}_t, \boldsymbol{u}_t) \right)$$

$$\text{s.t.} \quad \boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{u}_t).$$

Deterministic Physics Model

Inverse Optimal Control

Expressive, but large number of parameters: how to determine coefficients?

Linear combination of non-linear costs

Inverse Optimal Control

Structured Prediction Problem

Costs(state, input)

$$\min_{\boldsymbol{u}_{1:T}, \boldsymbol{s}_{1:T+1}} \sum_{t=1}^{T} \left(\sum_{i=1}^{k} \alpha_i c_t^{(i)}(\boldsymbol{s}_t, \boldsymbol{u}_t) \right) = \boldsymbol{w}^T \boldsymbol{f}(\xi)$$

coefficients costs Trajectory (sequence of state action pairs)

$$\text{s.t.} \quad \boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{u}_t).$$

Deterministic Physics Model

Direct Loss Update Rule

$$\xi^*_{i\,\text{direct}} = \text{argmin}_{\xi \in \Xi} \left(\boldsymbol{w}^T \boldsymbol{f}_i(\xi) + \epsilon \mathcal{L}_i(\xi) \right)$$

Find the trajectory that has a bad score and bad loss

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \sum_i \left(\boldsymbol{f}_i(\xi_i) - \boldsymbol{f}_i(\xi^*_{i\,\text{direct}}) \right)$$

Penalize features in the worst trajectory defined as high score and high loss that are different from the example

Direct Loss Minimization Inverse Optimal Control?

Use both!

Key insight: When viewing Imitation Learning as an inverse optimal control problem with the goal of matching an expert distribution, we can leverage reinforcement learning and solve the problem as a policy search problem

Direct Loss Minimization Inverse Optimal Control?

Use both!

- IL to regularize RL
- Noisy and partial expert examples

Example

$$p(\xi; \boldsymbol{w}) \propto e^{-C(\xi, \boldsymbol{w})}$$

$$\mathcal{F}_{ml}(\boldsymbol{w}) = \sum_{\xi \in D} \log(p(\xi; \boldsymbol{w}))$$

$$\mathcal{F}_{\mathrm{ps}}(\boldsymbol{w}) = \int \mathcal{R}(\xi) p(\xi; \boldsymbol{w}) d\xi$$

Maximum Likelihood Gradient

$$\boldsymbol{g}_{\mathrm{ml}} = -\mathrm{E}_{\mathcal{D}}[\nabla_{\boldsymbol{w}} C(\xi, \boldsymbol{w})] + \mathrm{E}_{p_{\boldsymbol{w}}}[\nabla_{\boldsymbol{w}} C(\xi, \boldsymbol{w})]$$

$$\mathcal{D} = \{\xi_i\}_{i=1}^{N}$$

Policy Search Gradient

$$\boldsymbol{g}_{\mathrm{ps}} = -\mathrm{E}_{p_{\boldsymbol{w}}^{\mathcal{R}}}[\nabla_{\boldsymbol{w}} C(\xi, \boldsymbol{w})] + \mathrm{E}_{p_{\boldsymbol{w}}}[\nabla_{\boldsymbol{w}} C(\xi, \boldsymbol{w})]$$

$$p_{\boldsymbol{w}}^{\mathcal{R}}(\xi) \propto \mathcal{R}(\xi) e^{-C(\xi; \boldsymbol{w})}$$

Example

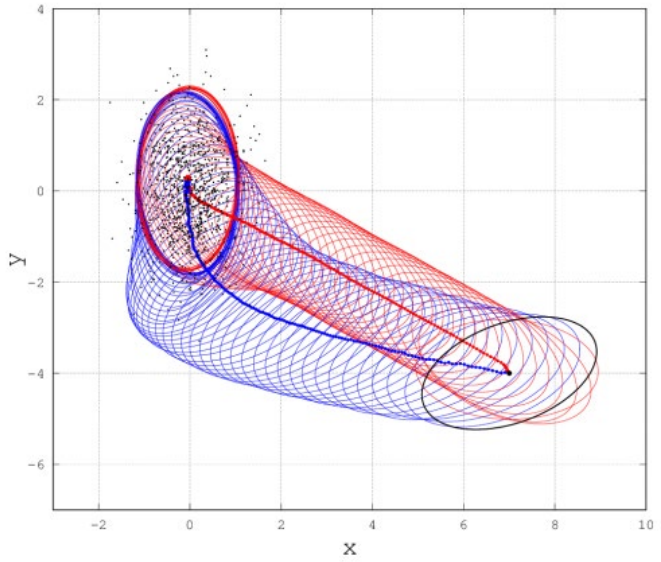
$$\mathcal{F}_{\mathrm{ps}}(\boldsymbol{w}) = \int \mathcal{R}(\xi) p(\xi; \boldsymbol{w}) d\xi$$

Policy Search Gradient

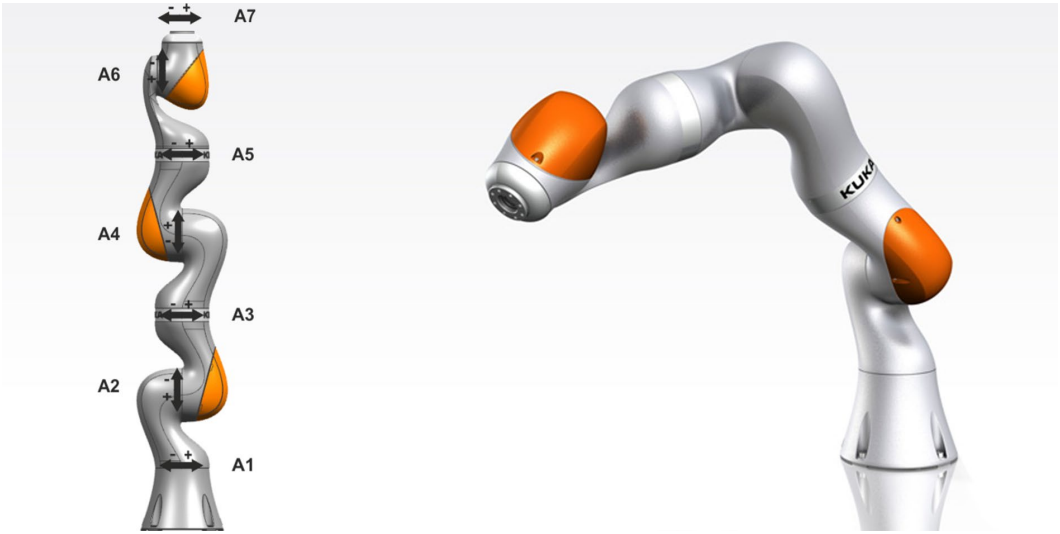
$$\boldsymbol{g}_{\mathrm{ps}} = -\mathrm{E}_{p_{\boldsymbol{w}}^{\mathcal{R}}}[\nabla_{\boldsymbol{w}} C(\xi, \boldsymbol{w})] + \mathrm{E}_{p_{\boldsymbol{w}}}[\nabla_{\boldsymbol{w}} C(\xi, \boldsymbol{w})]$$

$$p_{\boldsymbol{w}}^{\mathcal{R}}(\xi) \propto \mathcal{R}(\xi) e^{-C(\xi; \boldsymbol{w})}$$

$$\mathcal{L}(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_e)^T \boldsymbol{\Sigma}_e^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_e)$$



Experiments

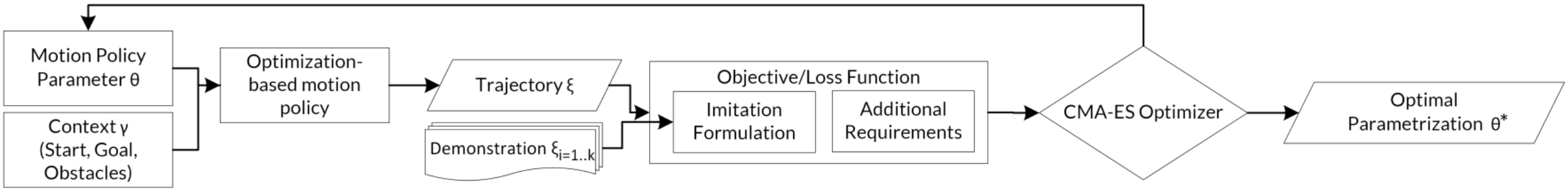


State: 11 DOF (joints), 11 velocities, 11 accelerations
Cost Function: 25 penalties, several constraints

Imitation Loss:

$$\mathcal{L}(\xi_i, \xi) = \sum_{t=1}^{T} \sum_{k=1}^{K} \left\| \phi_k(\boldsymbol{q}_i^{(t)}) - \phi_k(\boldsymbol{q}^{(t)}) \right\|^2$$

Pipeline



RIEMMO Optimization

$$\min_{\xi} \sum_{t=1}^{T} c_t(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \ddot{\boldsymbol{q}}_t)$$

$$\text{s.t.} \quad \boldsymbol{g}_t(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \ddot{\boldsymbol{q}}_t) \leq \boldsymbol{0} \quad \text{for all } t = 1, \ldots, T$$

$$\boldsymbol{h}_t(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \ddot{\boldsymbol{q}}_t) = \boldsymbol{0} \quad \text{for all } t = 1, \ldots, T,$$

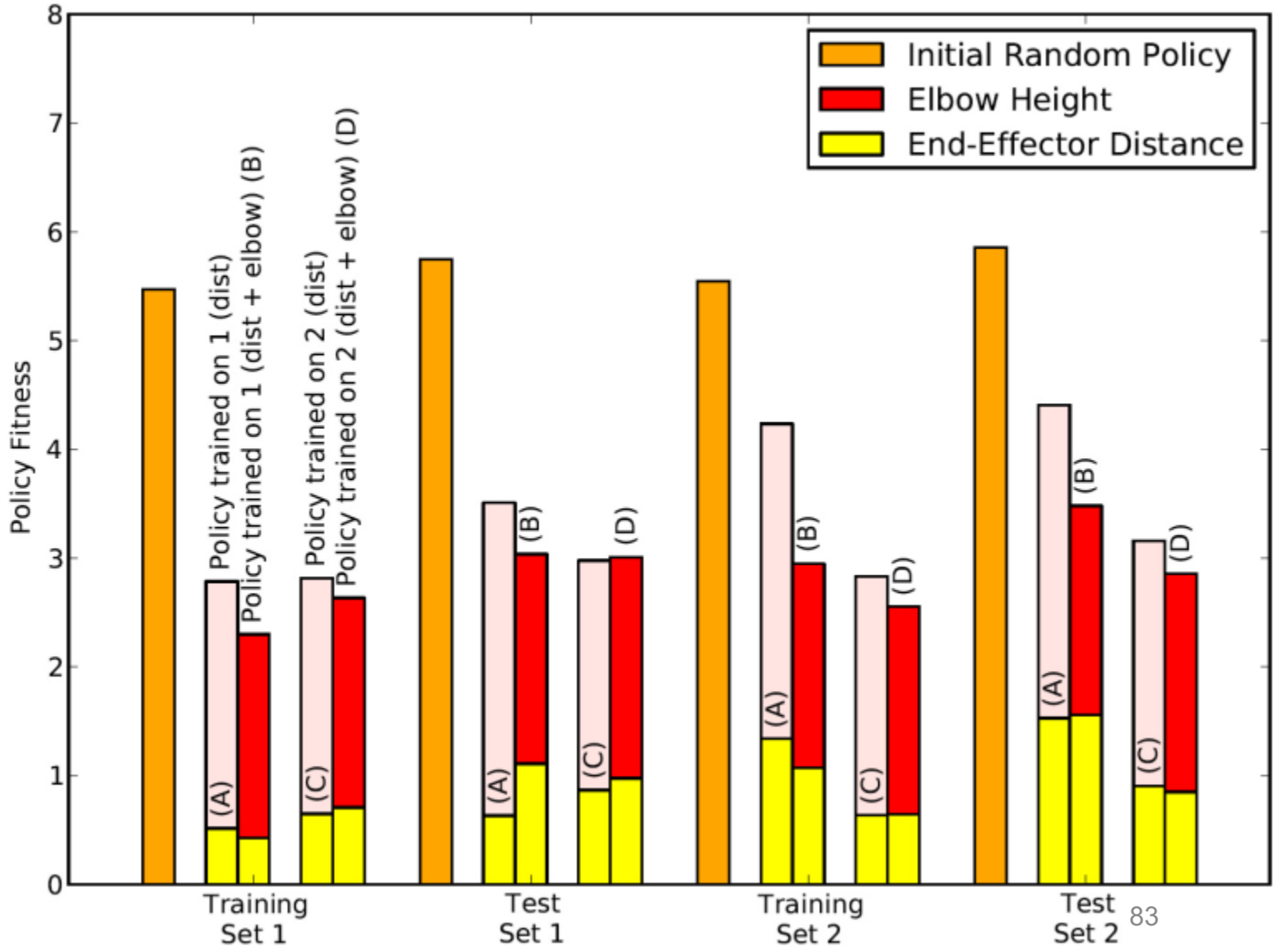
CMA-ES
Genetic Algorithm for Continuous Optimization

- Population sampled from multivariate Gaussian distribution

Results

A: Motion Type 1, Imitation Loss
B: Motion Type 1, Combined Loss
C: Motion Type 2, Imitation Loss
D: Motion Type 2, Combined Loss

Training Set 1: High obstacle avoidance
Training Set 2: Low obstacle avoidance



Paper Flaws

- No results for non-IL loss objective
- Reproducibility: Experimental Design vague and unclear

Future Work

- Determining additional objectives
- Time-dependent objectives
- Imitation Loss Metric
- Dynamic loss weighting over training time (decay IL loss)