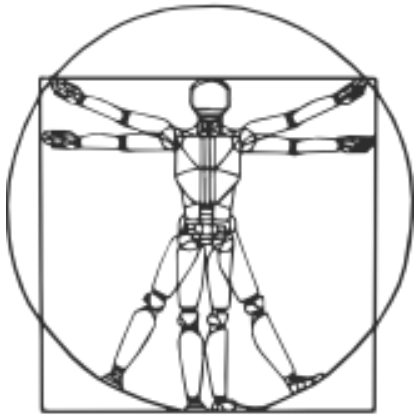


CSC2626
Imitation Learning for Robotics

Florian Shkurti

Week 6: Inverse Reinforcement Learning #1

Today's agenda

- Learning linear rewards from trajectory demonstrations in 2D
- Learning nonlinear rewards from trajectory demonstrations in 2D
- Guided cost learning in any D
- Updating distributions over reward parameters using preference elicitation

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

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)

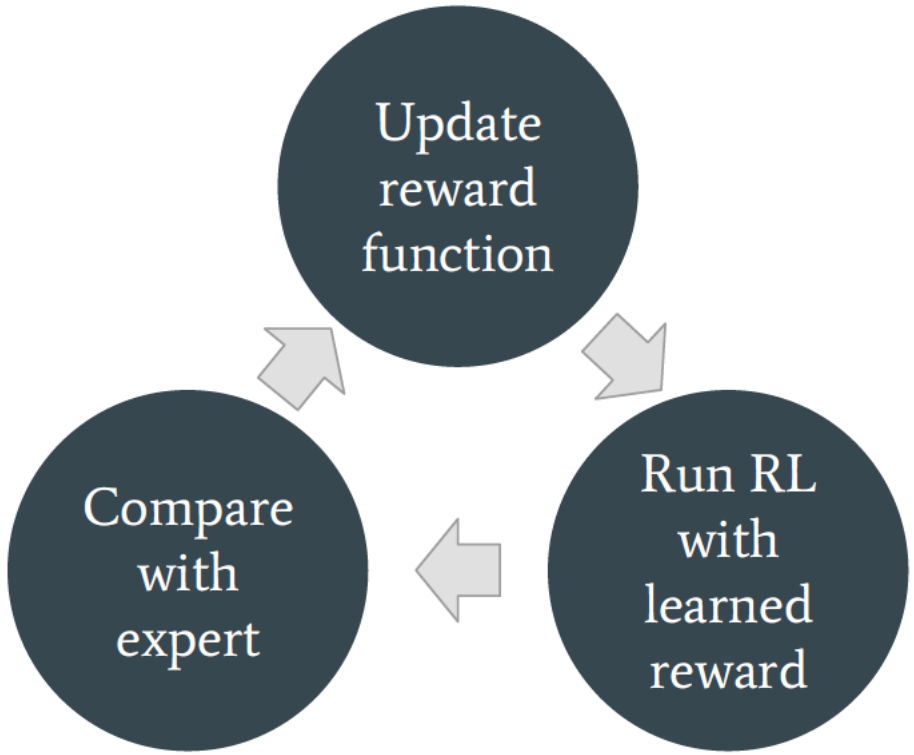
Imitation Learning approaches

- 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.

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)

Maximum Entropy Principle

$$\mathcal{H}(p) = -\int_x p(x)\log p(x)dx$$

$$p(x) = ?$$

$$\begin{cases} \underset{p(x)}{\operatorname{argmax}}\ \mathcal{H}(p) \\ \text{subject to} \\ \int_a^b p(x)dx = 1 \end{cases}$$

Maximum Entropy Principle

$$p(x) = ?$$

$$\begin{cases} \underset{p(x)}{\operatorname{argmax}}\ \mathcal{H}(p) \\ \text{subject to} \quad \int_x p(x)dx = 1 \\ \qquad\qquad\quad \mathbb{E}_{x\sim p(x)}\left[x\right] = \dfrac{1}{|D|}\sum_{x_i \in D} x_i = \hat{\mu} \\ \qquad\qquad\quad \mathbb{V}_{x\sim p(x)}\left[x\right] = \dfrac{1}{|D|}\sum_{x_i \in D} (x_i - \hat{\mu})^2 = \hat{\sigma}^2 \end{cases}$$

Maximum Entropy IRL [Ziebart et al. 2008]

$$p(\tau|\theta) = ?$$

$$\begin{cases} \underset{p(\tau|\theta)}{\operatorname{argmax}}\ \mathcal{H}(p) \\ \text{subject to} \quad \sum_{\tau} p(\tau|\theta) = 1 \\ \mathbb{E}_{\tau \sim p(\tau|\theta)}\left[\mathbf{f}_\tau\right] = \frac{1}{|D|}\sum_{\tau \in D} \mathbf{f}_\tau \end{cases}$$

Assumption: Trajectories (states and action sequences) here are discrete

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.**

Maximum Entropy IRL [Ziebart et al. 2008]

$$p(\tau|\theta) = \frac{\exp(\theta^\top \mathbf{f}_\tau)}{Z(\theta)}$$

$$\begin{cases} \underset{p(\tau|\theta)}{\operatorname{argmax}} \mathcal{H}(p) \\ \text{subject to} \quad \sum_\tau p(\tau|\theta) = 1 \\ \qquad\qquad \mathbb{E}_{\tau \sim p(\tau|\theta)}[\mathbf{f}_\tau] = \frac{1}{|D|} \sum_{\tau \in D} \mathbf{f}_\tau \end{cases}$$

Linear Reward Function

$$R_\theta(\tau) = \theta^\top \mathbf{f}_\tau$$

Maximum Entropy IRL [Ziebart et al. 2008]

$$p(\tau|\theta) = \frac{\exp(\theta^\top \mathbf{f}_\tau)}{Z(\theta)}$$

Linear Reward Function

$$R_\theta(\tau) = \theta^\top \mathbf{f}_\tau$$

$$p(\tau|\theta) = p(x_0) \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t)\, \pi_\theta(u_t|x_t) = \frac{\exp(R_\theta(\tau))}{Z(\theta)}$$

Maximum Entropy IRL [Ziebart et al. 2008]

$$p(\tau|\theta) = \frac{\exp(\theta^\top \mathbf{f}_\tau)}{Z(\theta)}$$

Linear Reward Function

$$R_\theta(\tau) = \theta^\top \mathbf{f}_\tau$$

$$p(\tau|\theta) = p(x_0) \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t)\, \pi_\theta(u_t|x_t) = \frac{\exp(R_\theta(\tau))}{Z(\theta)}$$

Assumption: known and deterministic dynamics

Maximum Entropy IRL [Ziebart et al. 2008]

$$p(\tau|\theta) = \frac{\exp(\theta^\top \mathbf{f}_\tau)}{Z(\theta)}$$

Linear Reward Function

$$R_\theta(\tau) = \theta^\top \mathbf{f}_\tau$$

$$p(\tau|\theta) = p(x_0) \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t)\, \pi_\theta(u_t|x_t) = \frac{\exp(R_\theta(\tau))}{Z(\theta)}$$

Assumption: known and deterministic dynamics

Log-likelihood of observed dataset D of trajectories

$$L(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \log p(\tau|\theta) = \frac{1}{|D|} \sum_{\tau \in D} \theta^\top \mathbf{f}_\tau - \log Z(\theta)$$

Maximum Entropy IRL [Ziebart et al. 2008]

$$p(\tau|\theta) = \frac{\exp(\theta^\top \mathbf{f}_\tau)}{Z(\theta)}$$

Linear Reward Function

$$R_\theta(\tau) = \theta^\top \mathbf{f}_\tau$$

$$p(\tau|\theta) = p(x_0) \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t)\, \pi_\theta(u_t|x_t) = \frac{\exp(R_\theta(\tau))}{Z(\theta)}$$

Assumption: known and deterministic dynamics

Log-likelihood of observed dataset D of trajectories

$$L(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \log p(\tau|\theta) = \frac{1}{|D|} \sum_{\tau \in D} \theta^\top \mathbf{f}_\tau - \log Z(\theta)$$

$$\nabla_\theta\, L(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \mathbf{f}_\tau - \sum_{\tau} p(\tau|\theta)\, \mathbf{f}_\tau$$

Maximum Entropy IRL [Ziebart et al. 2008]

$$p(\tau|\theta) = \frac{\exp(\theta^\top \mathbf{f}_\tau)}{Z(\theta)}$$

$$R_\theta(\tau) = \theta^\top \mathbf{f}_\tau$$

$$p(\tau|\theta) = p(x_0) \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t)\, \pi_\theta(u_t|x_t) = \frac{\exp(R_\theta(\tau))}{Z(\theta)}$$

Assumption: known and deterministic dynamics

Log-likelihood of observed dataset D of trajectories

$$L(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \log p(\tau|\theta) = \frac{1}{|D|} \sum_{\tau \in D} \theta^\top \mathbf{f}_\tau - \log Z(\theta)$$

$$\nabla_\theta L(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \mathbf{f}_\tau - \sum_\tau p(\tau|\theta)\, \mathbf{f}_\tau$$

Serious problem: Need to compute Z(theta) every time we compute the gradient

Maximum Entropy IRL [Ziebart et al. 2008]

$$p(\tau|\theta) = \frac{\exp(\theta^\top \mathbf{f}_\tau)}{Z(\theta)}$$

$$R_\theta(\tau) = \theta^\top \mathbf{f}_\tau$$

$$p(\tau|\theta) = p(x_0) \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t)\, \pi_\theta(u_t|x_t) = \frac{\exp(R_\theta(\tau))}{Z(\theta)}$$

Assumption: known and deterministic dynamics

Log-likelihood of observed dataset D of trajectories

$$L(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \log p(\tau|\theta) = \frac{1}{|D|} \sum_{\tau \in D} \theta^\top \mathbf{f}_\tau - \log Z(\theta)$$

$$\nabla_\theta\, L(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \mathbf{f}_\tau - \sum_x p(x|\theta)\, \mathbf{f}_x$$

Serious problem remains

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

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:

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

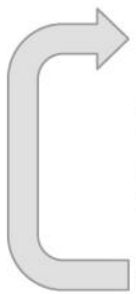
- For deterministic MDPs, this function is convex and can be optimized using gradient descent:

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

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

State visitation distribution

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

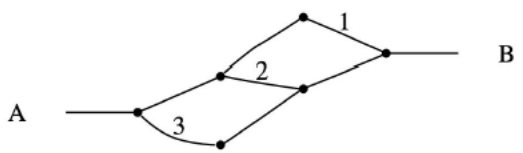
- Interested in **predicting driver behavior** and **route recommendation**
- Pittsburgh's **road network as an MDP**
  - >300,000 states or road segments
  - >900,000 actions or transitions at intersections
- Destination is represented as an absorbing state with zero-cost. Thus, trips with different 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

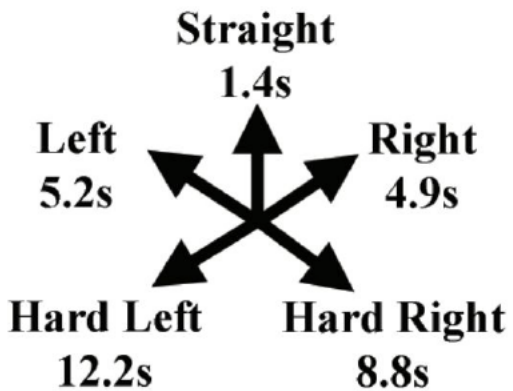
Application: Driver Route Modelling

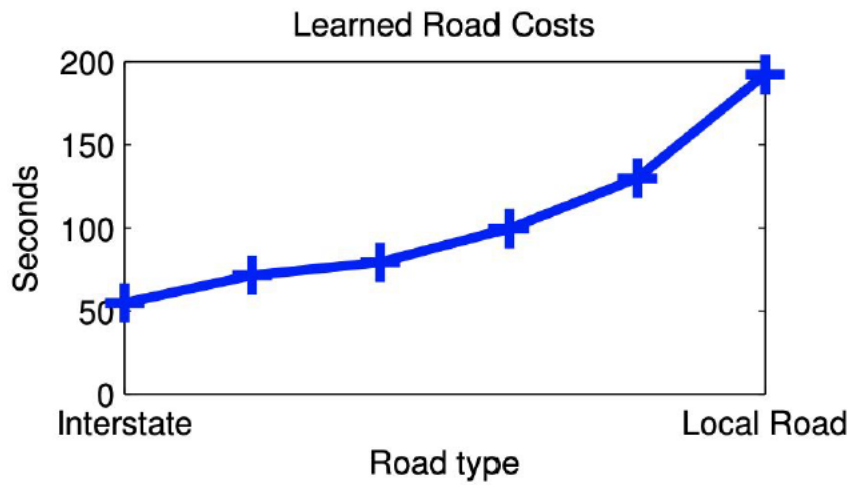
	Matching	90% Match	Log Prob
Time-based	72.38%	43.12%	N/A
Max Margin	75.29%	46.56%	N/A
Action	77.30%	50.37%	-7.91
Action (costs)	77.74%	50.75%	N/A
MaxEnt paths	**78.79%**	**52.98%**	**-6.85**

- **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



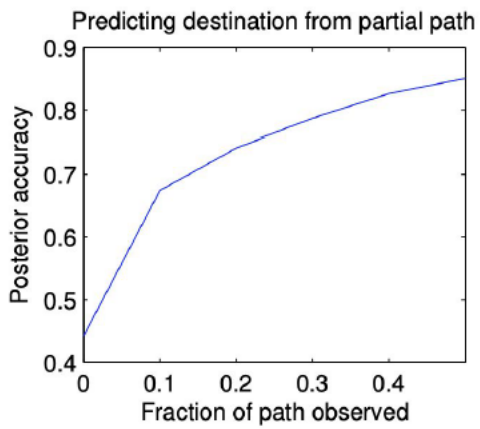


Application: Driver Route Modelling

- **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

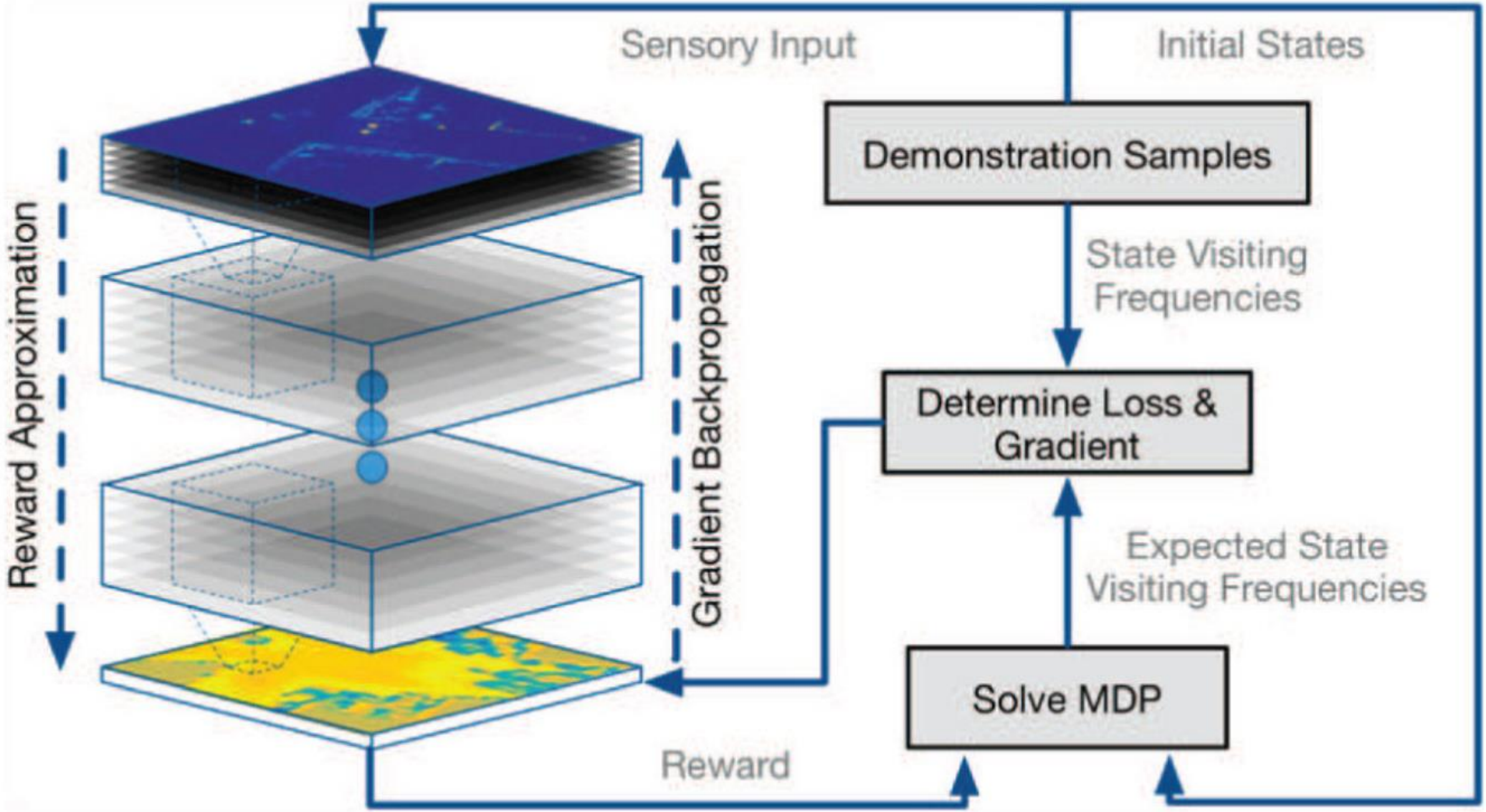
Today's agenda

- Learning linear rewards from trajectory demonstrations in 2D
- Learning nonlinear rewards from trajectory demonstrations in 2D
- Guided cost learning in any D
- Updating distributions over reward parameters using preference elicitation

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

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

Wulfmeier et. al (IJRR 2017)


Sensory Input
Initial States
Demonstration Samples
State Visiting Frequencies
Determine Loss & Gradient
Expected State Visiting Frequencies
Solve MDP
Reward
Reward Approximation
Gradient Backpropagation

Deep Maximum Entropy IRL [Wulfmeier et al. 2017]

$$p(\tau|\theta) = \frac{\exp(R_\theta(\tau))}{Z(\theta)}$$

$$p(\tau|\theta) = p(x_0) \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t)\, \pi_\theta(u_t|x_t) = \frac{\exp(R_\theta(\tau))}{Z(\theta)}$$

Assumption: known and deterministic dynamics

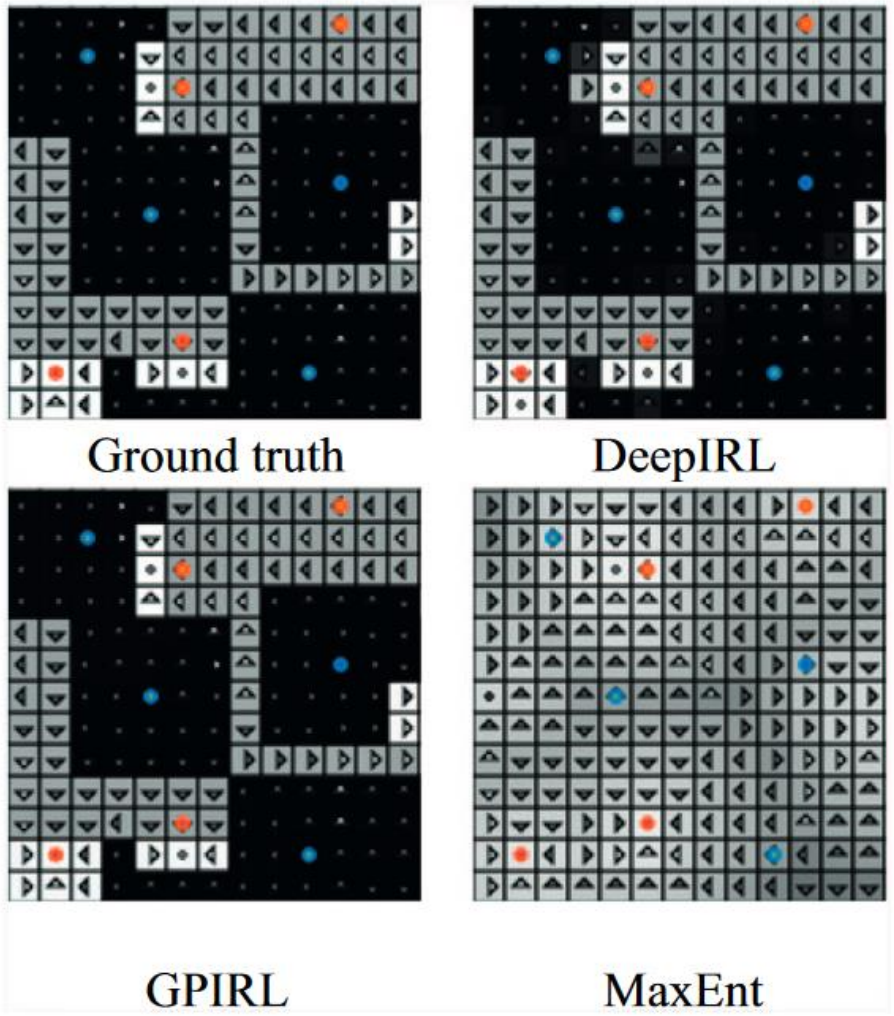
Log-likelihood of observed dataset D of trajectories

$$L(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \log p(\tau|\theta) = \frac{1}{|D|} \sum_{\tau \in D} R_\theta(\tau) - \log Z(\theta)$$

$$\nabla_\theta L(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \nabla_\theta R_\theta(\tau) - \sum_{\tau} p(\tau|\theta) \nabla_\theta R_\theta(\tau)$$

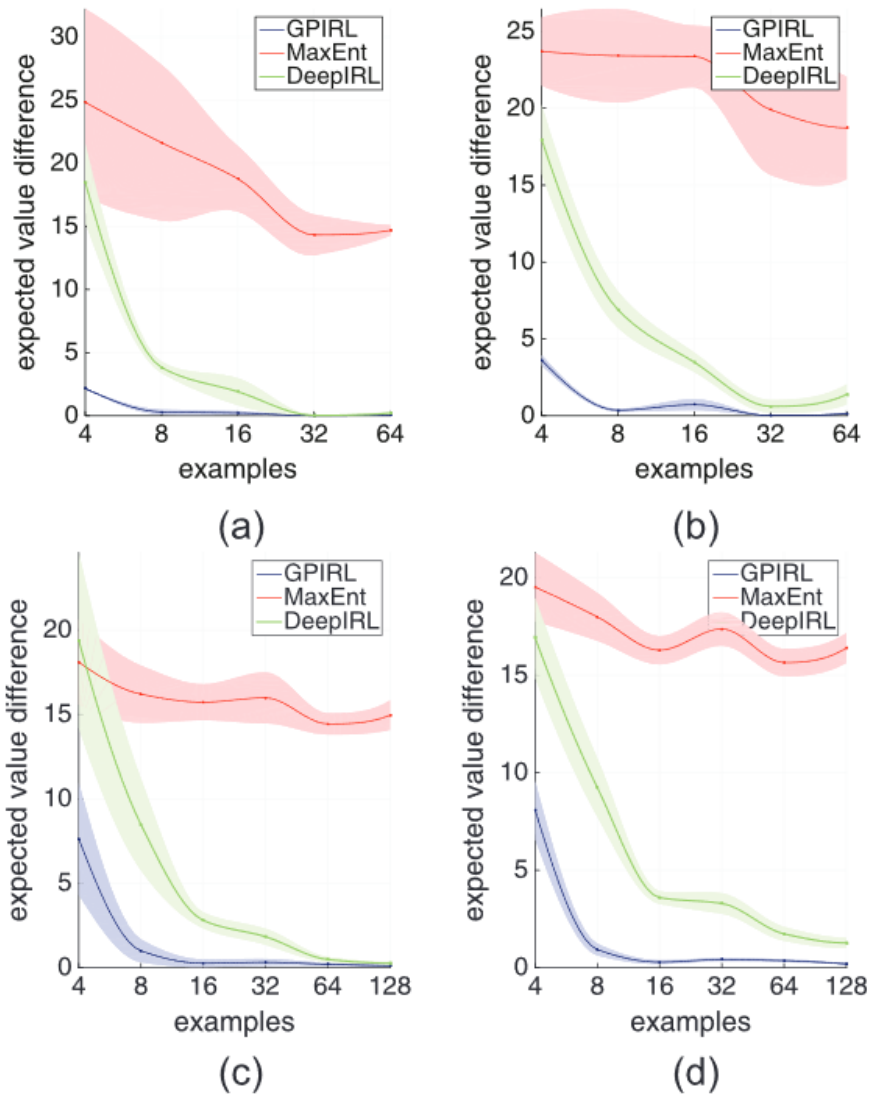
Benchmarking

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

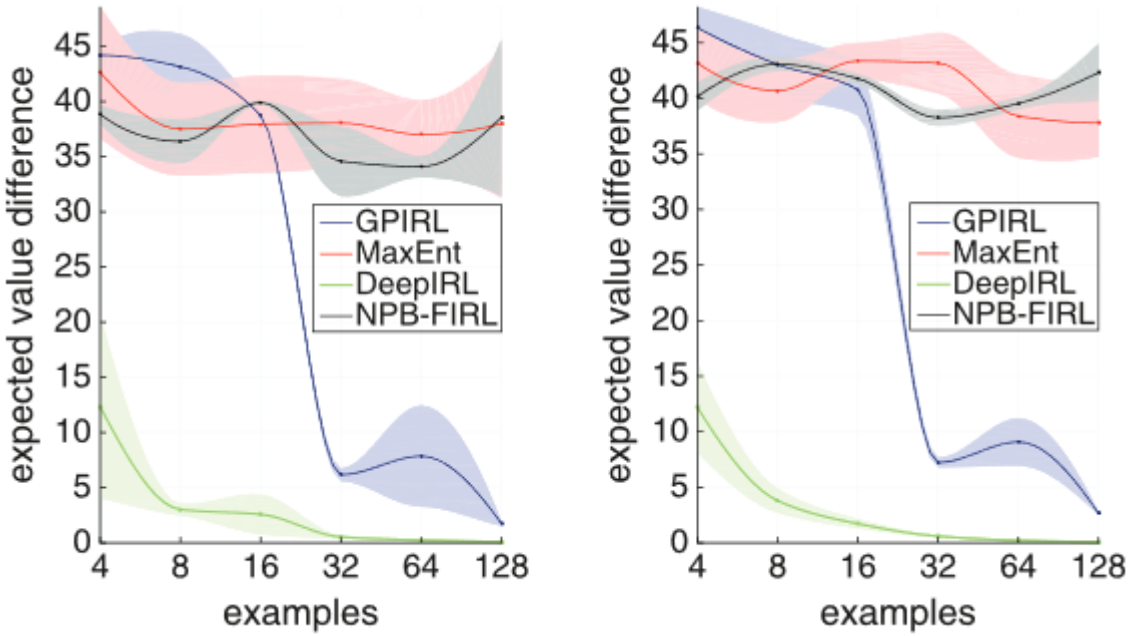


ObjectWorld

Benchmarking

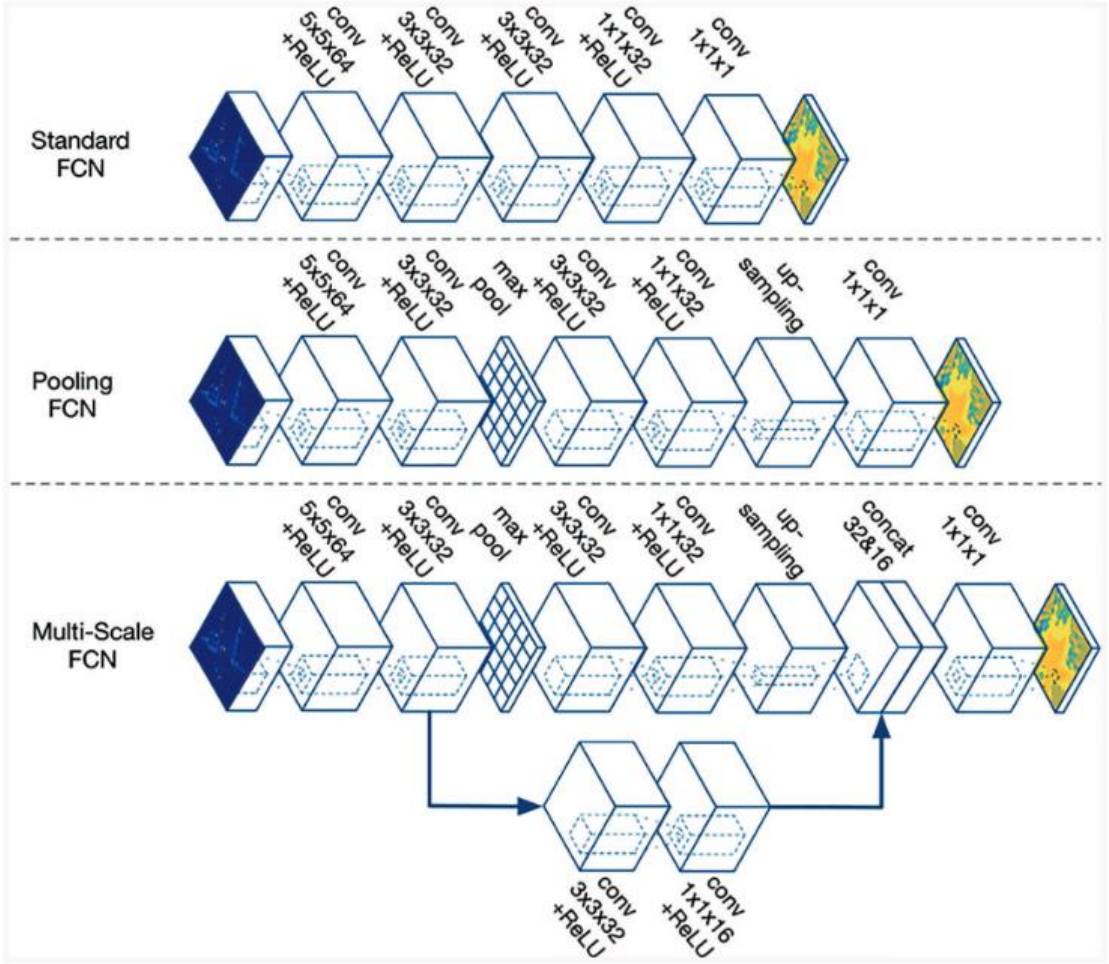


ObjectWorld



BinaryWorld

Proposed Network Architectures



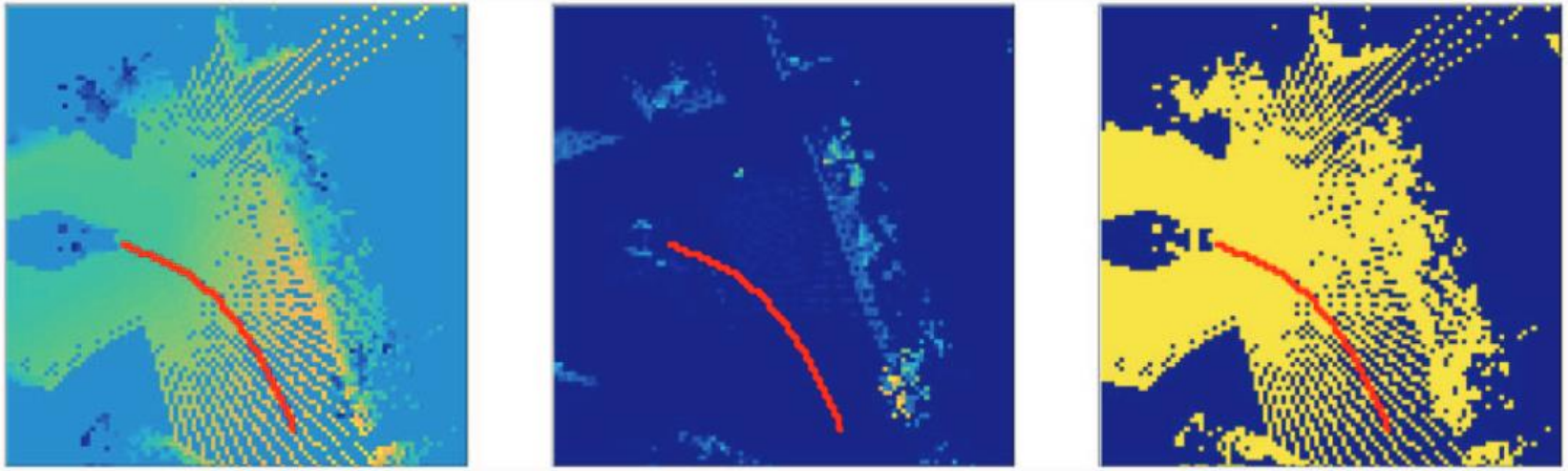
Large-scale Demonstration

- 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.

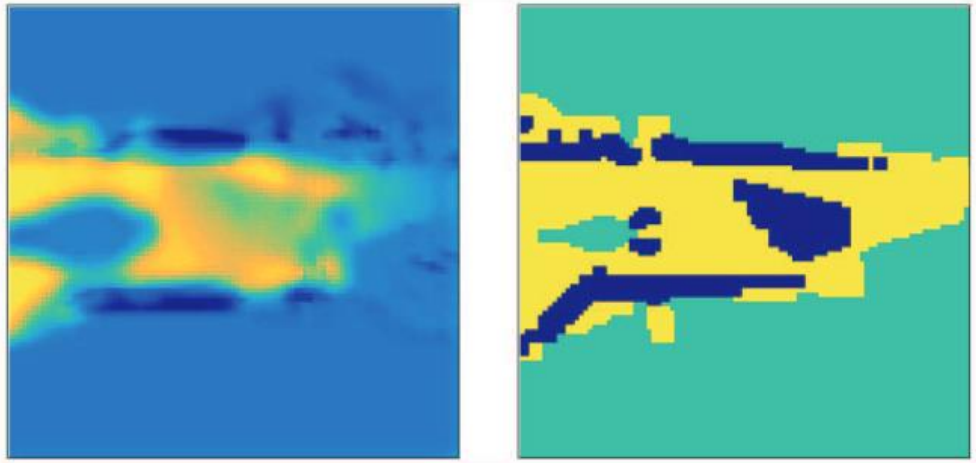
Evaluation

- 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

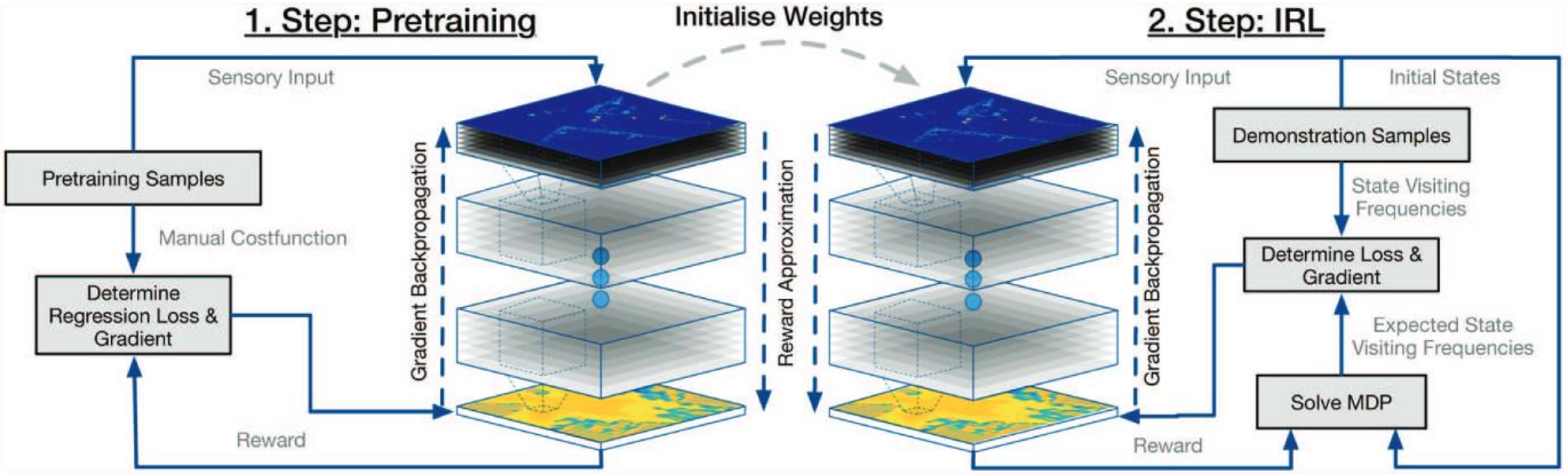
Robustness to Systematic Noise



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

Today's agenda

- Learning linear rewards from trajectory demonstrations in 2D
- Learning nonlinear rewards from trajectory demonstrations in 2D
- Guided cost learning in any D
- Updating distributions over reward parameters using preference elicitation

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

Guided Cost Learning [Finn, Levine, Abbeel et al. 2016]

$$p(\tau|\theta) = \frac{\exp(-c_\theta(\tau))}{Z(\theta)}$$

Nonlinear Reward Function
Learned Features

$$p(\tau|\theta) = p(x_0) \prod_{t=0}^{T-1} \underbrace{p(x_{t+1}|x_t, u_t)}\ \pi_\theta(u_t|x_t) = \frac{\exp(-c_\theta(\tau))}{Z(\theta)}$$

True and stochastic dynamics (unknown)

Log-likelihood of observed dataset D of trajectories

$$L(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \log p(\tau|\theta) = \frac{1}{|D|} \sum_{\tau \in D} -c_\theta(\tau) - \log Z(\theta)$$

Guided Cost Learning [Finn, Levine, Abbeel et al. 2016]

$$p(\tau|\theta) = \frac{\exp(-c_\theta(\tau))}{Z(\theta)}$$

Nonlinear Reward Function
Learned Features

$$p(\tau|\theta) = p(x_0) \prod_{t=0}^{T-1} \underbrace{p(x_{t+1}|x_t, u_t)}\ \pi_\theta(u_t|x_t) = \frac{\exp(-c_\theta(\tau))}{Z(\theta)}$$

True and stochastic dynamics (unknown)

Log-likelihood of observed dataset D of trajectories

$$L(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \log p(\tau|\theta) = \frac{1}{|D|} \sum_{\tau \in D} -c_\theta(\tau) - \log Z(\theta)$$

$$\nabla_\theta L(\theta) = -\frac{1}{|D|} \sum_{\tau \in D} \nabla_\theta c_\theta(\tau) + \sum_{\tau} p(\tau|\theta) \nabla_\theta c_\theta(\tau)$$

Serious problem remains

Approximating the gradient of the log-likelihood

$$\nabla_\theta L(\theta) = -\frac{1}{|D|}\sum_{\tau \in D} \nabla_\theta c_\theta(\tau) + \sum_{\tau} p(\tau|\theta)\, \nabla_\theta c_\theta(\tau)$$

How do you approximate this expectation?

$$p(\tau|\theta) = \frac{\exp(-c_\theta(\tau))}{Z(\theta)}$$

Nonlinear Reward Function
Learned Features

Approximating the gradient of the log-likelihood

$$\nabla_\theta L(\theta) = -\frac{1}{|D|}\sum_{\tau \in D} \nabla_\theta c_\theta(\tau) + \sum_{\tau} p(\tau|\theta)\, \nabla_\theta c_\theta(\tau)$$

How do you approximate this expectation?

Idea #1: sample from $p(\tau|\theta)$
(can you do this?)

$$p(\tau|\theta) = \frac{\exp(-c_\theta(\tau))}{Z(\theta)}$$

Nonlinear Reward Function
Learned Features

Approximating the gradient of the log-likelihood

$$\nabla_\theta L(\theta) = -\frac{1}{|D|}\sum_{\tau \in D} \nabla_\theta c_\theta(\tau) + \sum_{\tau} p(\tau|\theta)\, \nabla_\theta c_\theta(\tau)$$

How do you approximate this expectation?

Idea #1: sample from $p(\tau|\theta)$
(don't know the dynamics ☹)

$$p(\tau|\theta) = \frac{\exp(-c_\theta(\tau))}{Z(\theta)}$$

Nonlinear Reward Function
Learned Features

Approximating the gradient of the log-likelihood

$$\nabla_\theta L(\theta) = -\frac{1}{|D|}\sum_{\tau \in D} \nabla_\theta c_\theta(\tau) + \sum_{\tau} p(\tau|\theta)\, \nabla_\theta c_\theta(\tau)$$

How do you approximate this expectation?

Idea #1: sample from $p(\tau|\theta)$
(don't know the dynamics ☹)

Idea #2: sample from an easier distribution $q(\tau|\theta)$ that approximates $p(\tau|\theta)$

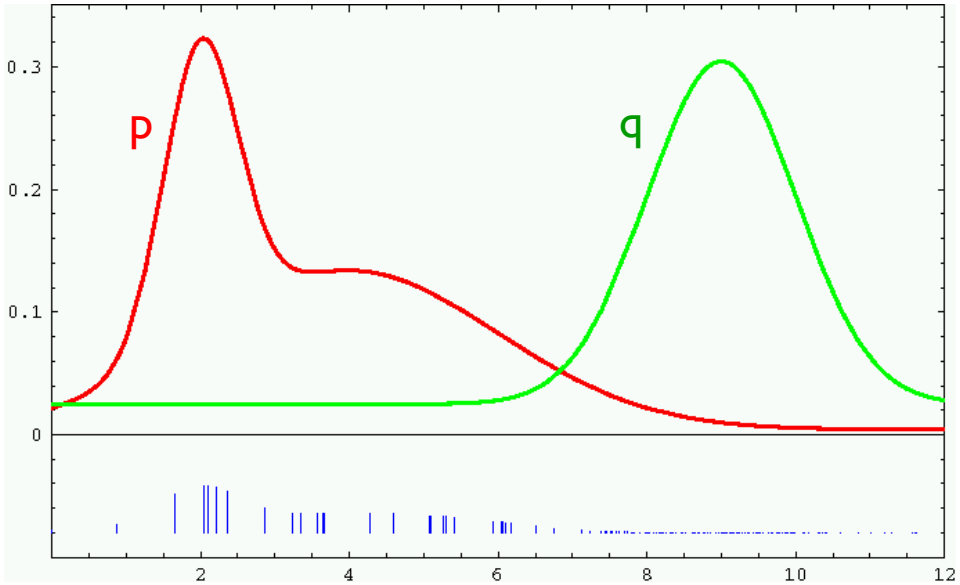
Importance Sampling
see Relative Entropy Inverse RL
by Boularias, Kober, Peters

$$p(\tau|\theta) = \frac{\exp(-c_\theta(\tau))}{Z(\theta)}$$

Nonlinear Reward Function
Learned Features

Importance Sampling

How to estimate properties/statistics of one distribution (p) given samples from another distribution (q)

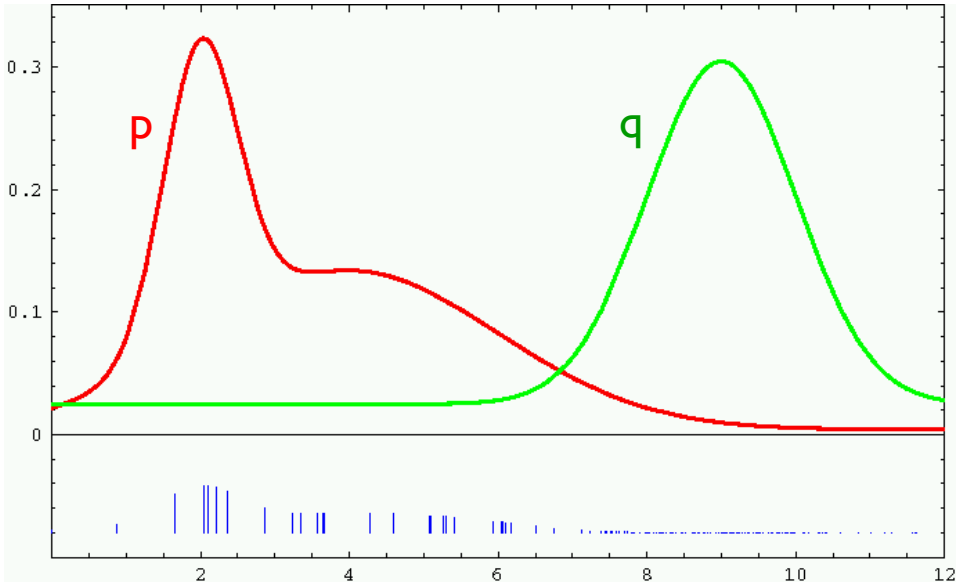


$$
\begin{aligned}
\mathbb{E}_{x\sim p(x)}[f(x)] &= \int f(x)p(x)dx \\
&= \int \frac{q(x)}{q(x)} f(x)p(x)dx \\
&= \int \frac{f(x)p(x)}{q(x)} q(x)dx \\
&= \mathbb{E}_{x\sim q(x)}\left[f(x)\frac{p(x)}{q(x)}\right] \\
&= \mathbb{E}_{x\sim q(x)}\left[f(x)\ w(x)\right]
\end{aligned}
$$

Weights = likelihood ratio,
i.e. how to reweigh samples to obtain statistics of p from samples of q

Importance Sampling: Pitfalls and Drawbacks

What can go wrong?



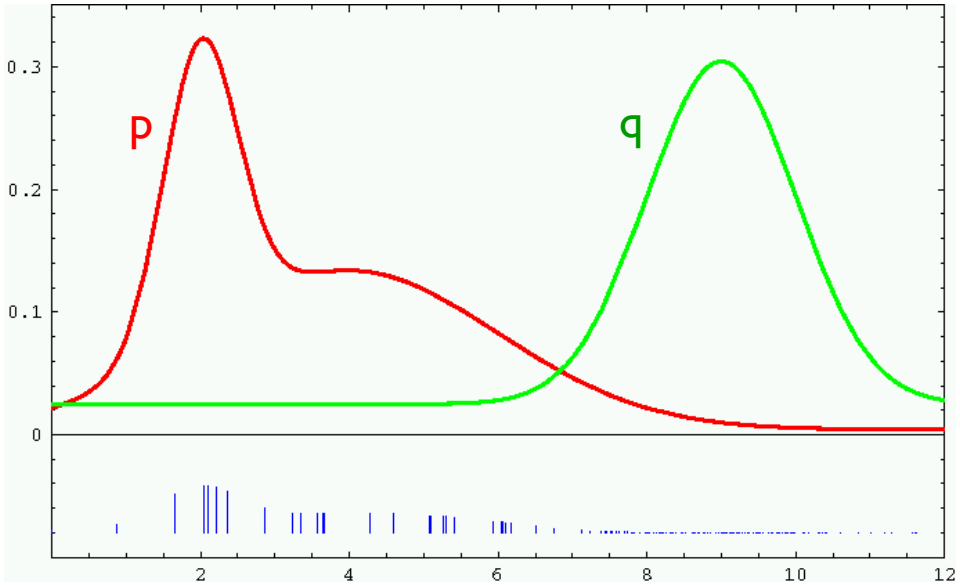
$$
\begin{aligned}
\mathbb{E}_{x\sim p(x)}[f(x)] &= \int f(x)p(x)dx \\
&= \int \frac{q(x)}{q(x)} f(x)p(x)dx \\
&= \int \frac{f(x)p(x)}{q(x)} q(x)dx \\
&= \mathbb{E}_{x\sim q(x)}\left[f(x)\frac{p(x)}{q(x)}\right] \\
&= \mathbb{E}_{x\sim q(x)}\left[f(x)\ w(x)\right]
\end{aligned}
$$

Problem #1:

If q(x) = 0 but f(x)p(x) > 0 for x in non-measure-zero set then there is estimation bias

Importance Sampling: Pitfalls and Drawbacks

What can go wrong?



$$
\begin{aligned}
\mathbb{E}_{x\sim p(x)}[f(x)] &= \int f(x)p(x)dx \\
&= \int \frac{q(x)}{q(x)} f(x)p(x)dx \\
&= \int \frac{f(x)p(x)}{q(x)} q(x)dx \\
&= \mathbb{E}_{x\sim q(x)}\left[f(x)\frac{p(x)}{q(x)}\right] \\
&= \mathbb{E}_{x\sim q(x)}\left[f(x)\ w(x)\right]
\end{aligned}
$$

Problem #1:

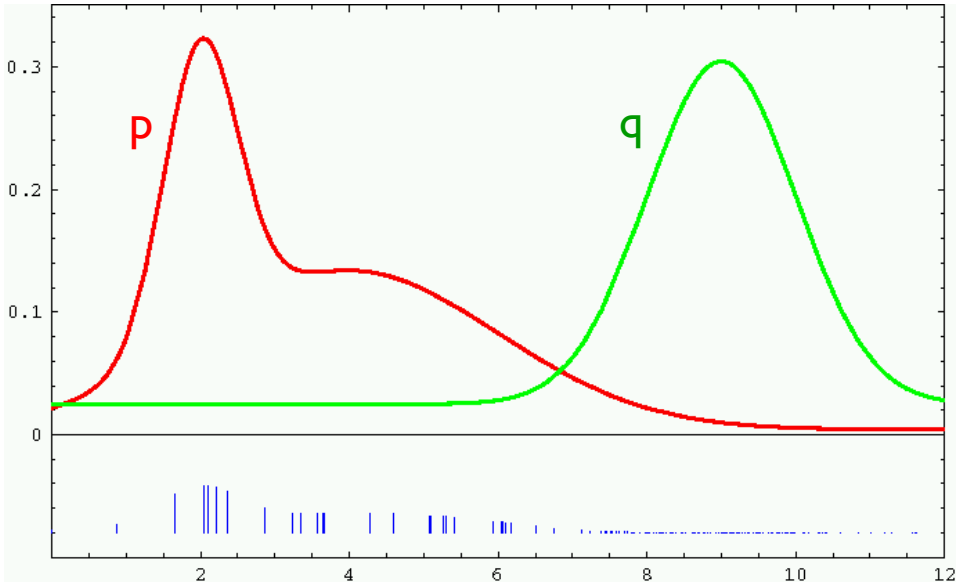
If q(x) = 0 but f(x)p(x) > 0 for x in non-measure-zero set then there is estimation bias

Problem #2:

Weights measure mismatch between q(x) and p(x). If mismatch is large then some weights will dominate. If x lives in high dimensions a single weight may dominate

Importance Sampling: Pitfalls and Drawbacks

What can go wrong?



$$
\begin{aligned}
\mathbb{E}_{x\sim p(x)}[f(x)] &= \int f(x)p(x)dx \\
&= \int \frac{q(x)}{q(x)} f(x)p(x)dx \\
&= \int \frac{f(x)p(x)}{q(x)} q(x)dx \\
&= \mathbb{E}_{x\sim q(x)}\left[f(x)\frac{p(x)}{q(x)}\right] \\
&= \mathbb{E}_{x\sim q(x)}\left[f(x)\ w(x)\right]
\end{aligned}
$$

Problem #1:

If q(x) = 0 but f(x)p(x) > 0 for x in non-measure-zero set then there is estimation bias

Problem #2:

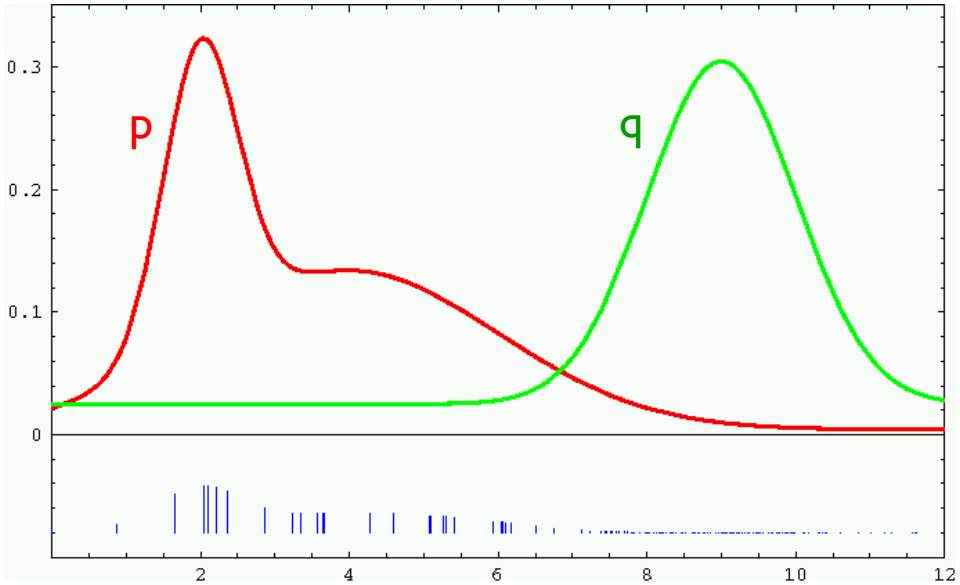
Weights measure mismatch between q(x) and p(x). If mismatch is large then some weights will dominate. If x lives in high dimensions a single weight may dominate

Problem #3:

Variance of estimator is high if (q – fp)(x) is high

Importance Sampling: Pitfalls and Drawbacks

What can go wrong?



$$
\begin{aligned}
\mathbb{E}_{x\sim p(x)}[f(x)] &= \int f(x)p(x)dx \\
&= \int \frac{q(x)}{q(x)} f(x)p(x)dx \\
&= \int \frac{f(x)p(x)}{q(x)} q(x)dx \\
&= \mathbb{E}_{x\sim q(x)}\left[f(x)\frac{p(x)}{q(x)}\right] \\
&= \mathbb{E}_{x\sim q(x)}\left[f(x)\ w(x)\right]
\end{aligned}
$$

Problem #1:

If q(x) = 0 but f(x)p(x) > 0 for x in non-measure-zero set then there is estimation bias

Problem #2:

Weights measure mismatch between q(x) and p(x). If mismatch is large then some weights will dominate. If x lives in high dimensions a single weight may dominate

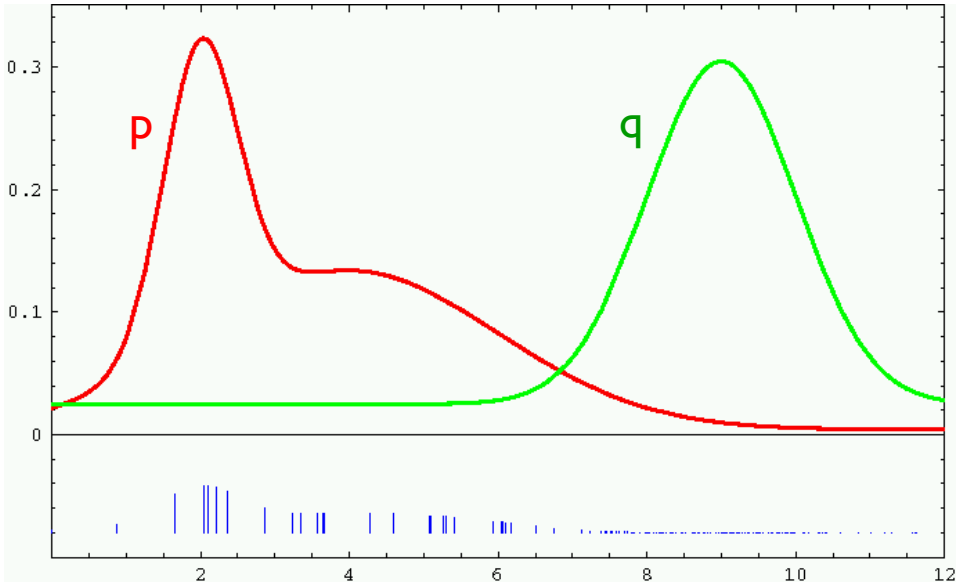
Problem #3:

Variance of estimator is high if (q – fp)(x) is high

For more info see:
#1, #3: Monte Carlo theory, methods, and examples, Art Owen, chapter 9
#2: Bayesian reasoning and machine learning, David Barber, chapter 27.6 on importance sampling

Importance Sampling

What is the best approximating distribution q?

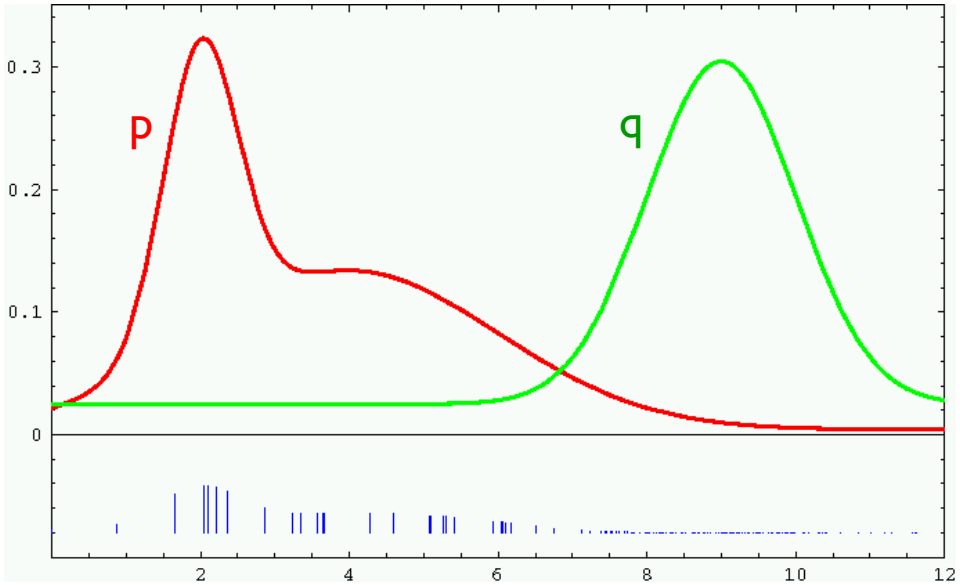


$$
\begin{aligned}
\mathbb{E}_{x\sim p(x)}[f(x)] &= \int f(x)p(x)dx \\
&= \int \frac{q(x)}{q(x)} f(x)p(x)dx \\
&= \int \frac{f(x)p(x)}{q(x)} q(x)dx \\
&= \mathbb{E}_{x\sim q(x)}\left[f(x)\frac{p(x)}{q(x)}\right] \\
&= \mathbb{E}_{x\sim q(x)}\left[f(x)\ w(x)\right]
\end{aligned}
$$

Best approximation $q(x) \propto f(x)p(x)$

Importance Sampling

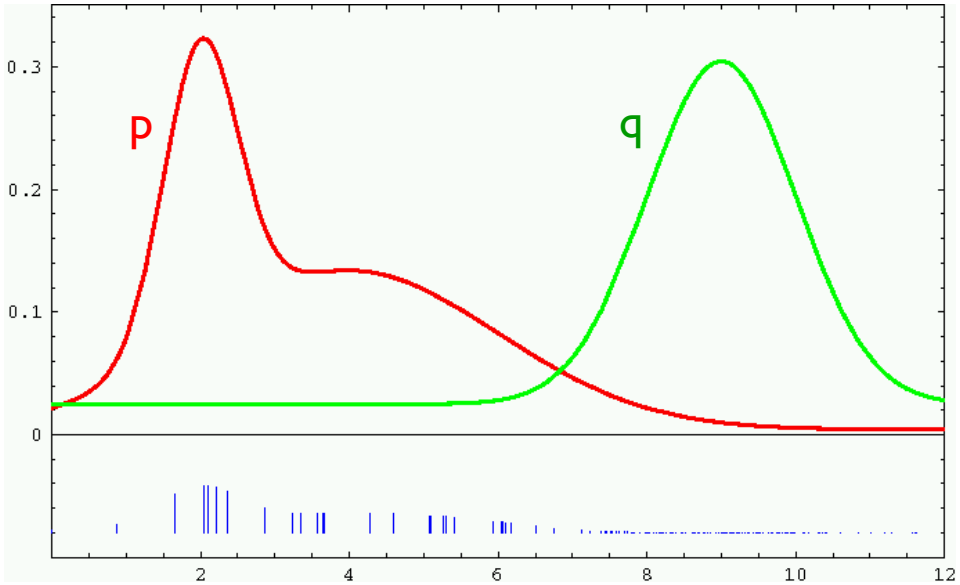
How does this connect back to partition function estimation?



$$
\begin{aligned}
Z(\theta) &= \sum_{\tau} \exp(-c_\theta(\tau)) \\
&= \sum_{\tau} \exp(-c_\theta(\tau)) \\
&= \sum_{\tau} \frac{q(\tau|\theta)}{q(\tau|\theta)} \exp(-c_\theta(\tau)) \\
&= \mathbb{E}_{\tau \sim q(\tau|\theta)} \left[\frac{\exp(-c_\theta(\tau))}{q(\tau|\theta)} \right]
\end{aligned}
$$

Importance Sampling

How does this connect back to partition function estimation?

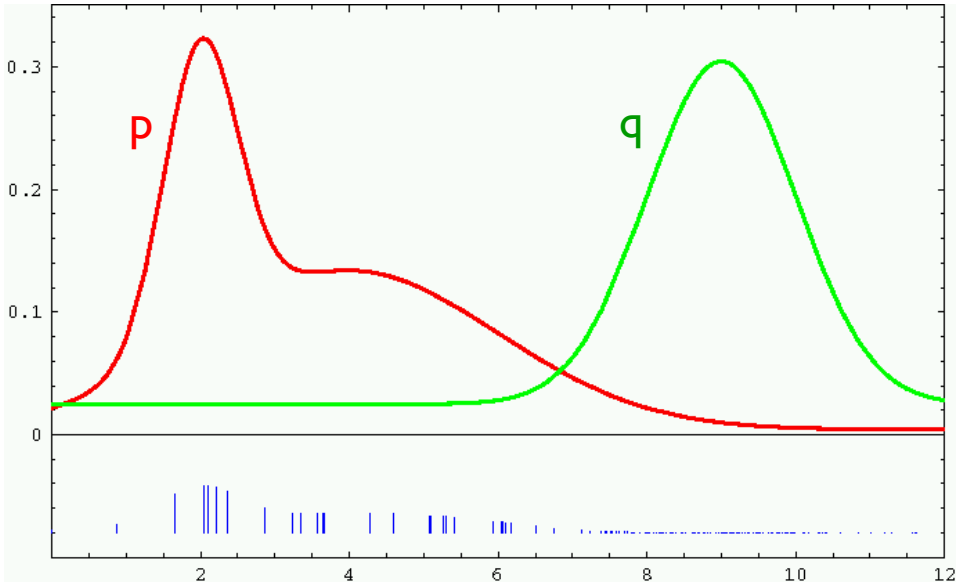


$$
\begin{aligned}
Z(\theta) &= \sum_{\tau} \exp(-c_\theta(\tau)) \\
&= \sum_{\tau} \exp(-c_\theta(\tau)) \\
&= \sum_{\tau} \frac{q(\tau|\theta)}{q(\tau|\theta)} \exp(-c_\theta(\tau)) \\
&= \mathbb{E}_{\tau \sim q(\tau|\theta)} \left[\frac{\exp(-c_\theta(\tau))}{q(\tau|\theta)} \right]
\end{aligned}
$$

Best approximating distribution $q(\tau|\theta) \propto \exp(-c_\theta(\tau))$

Importance Sampling

How does this connect back to partition function estimation?



$$
\begin{aligned}
Z(\theta) &= \sum_{\tau} \exp(-c_\theta(\tau)) \\
&= \sum_{\tau} \exp(-c_\theta(\tau)) \\
&= \sum_{\tau} \frac{q(\tau|\theta)}{q(\tau|\theta)} \exp(-c_\theta(\tau)) \\
&= \mathbb{E}_{\tau \sim q(\tau|\theta)} \left[\frac{\exp(-c_\theta(\tau))}{q(\tau|\theta)} \right]
\end{aligned}
$$

Best approximating distribution $q(\tau|\theta) \propto \exp(-c_\theta(\tau))$

Cost function estimate changes at each gradient step
Therefore the best approximating distribution should change as well

Approximating the gradient of the log-likelihood

$$\nabla_\theta L(\theta) = -\frac{1}{|D|}\sum_{\tau \in D} \nabla_\theta c_\theta(\tau) + \sum_{\tau} p(\tau|\theta)\, \nabla_\theta c_\theta(\tau)$$

How do you approximate this expectation?

Idea #1: sample from $p(\tau|\theta)$
(don't know the dynamics ☹)

Idea #2: sample from an easier distribution $q(\tau|\theta)$ that approximates $p(\tau|\theta)$

Importance Sampling
see Relative Entropy Inverse RL
by Boularias, Kober, Peters

$$p(\tau|\theta) = \frac{\exp(-c_\theta(\tau))}{Z(\theta)}$$

Nonlinear Reward Function
Learned Features

Approximating the gradient of the log-likelihood

$$\nabla_\theta L(\theta) = -\frac{1}{|D|}\sum_{\tau \in D} \nabla_\theta c_\theta(\tau) + \sum_\tau p(\tau|\theta)\, \nabla_\theta c_\theta(\tau)$$

How do you approximate this expectation?

Idea #1: sample from $p(\tau|\theta)$
(don't know the dynamics ☹)

Idea #2: sample from an easier distribution $q(\tau|\theta)$ that approximates $p(\tau|\theta)$

Previous papers used a fixed $q(\tau|\theta)$

Importance Sampling
see Relative Entropy Inverse RL by Boularias, Kober, Peters

$$p(\tau|\theta) = \frac{\exp(-c_\theta(\tau))}{Z(\theta)}$$

Nonlinear Reward Function
Learned Features

Approximating the gradient of the log-likelihood

$$\nabla_\theta L(\theta) = -\frac{1}{|D|}\sum_{\tau \in D} \nabla_\theta c_\theta(\tau) + \underbrace{\sum_\tau p(\tau|\theta)\, \nabla_\theta c_\theta(\tau)}$$

How do you approximate this expectation?

$$p(\tau|\theta) = \frac{\exp(-c_\theta(\tau))}{Z(\theta)}$$

Nonlinear Reward Function
Learned Features

Idea #1: sample from $p(\tau|\theta)$
(don't know the dynamics ☹)

Idea #2: sample from an easier distribution $q(\tau|\theta)$ that approximates $p(\tau|\theta)$

Previous papers used a fixed $q(\tau|\theta)$

Importance Sampling
see Relative Entropy Inverse RL
by Boularias, Kober, Peters

This paper uses adaptive $q(\tau|\theta)$

Adaptive Importance Sampling
see Guided Cost Learning
By Finn, Levine, Abbeel

Guided Cost Learning

How do you select q?

How do you adapt it as the cost c changes?

Guided Cost Learning: the punchline

How do you select q?

How do you adapt it as the cost c changes?

Given a fixed cost function c, the distribution of trajectories that Guided Policy Search computes is close to $\frac{\exp(-c(\tau))}{Z}$
i.e. it is good for importance sampling of the partition function Z

Recall: Finite-Horizon LQR

$$P_0 = Q$$

// n is the # of steps left

for n = 1…N

$$K_n = -(R + B^T P_{n-1} B)^{-1} B^T P_{n-1} A$$

$$P_n = Q + K_n^T R K_n + (A + BK_n)^T P_{n-1} (A + BK_n)$$

Optimal control for time t = N – n is $\mathbf{u}_t = K_t \mathbf{x}_t$ with cost-to-go $J_t(\mathbf{x}) = \mathbf{x}^T P_t \mathbf{x}$

where the states are predicted forward in time according to linear dynamics

Recall: LQG = LQR with stochastic dynamics

Assume $\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t + \mathbf{w}_t$ and $c(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_t^T Q \mathbf{x}_t + \mathbf{u}_t^T R \mathbf{u}_t$

zero mean Gaussian

Then the form of the optimal policy is the same as in LQR $\mathbf{u}_t = K_t \hat{\mathbf{x}}_t$ ← estimate of the state

No need to change the algorithm, as long as you observe the state at each step (closed-loop policy)

Linear Quadratic Gaussian
LQG

Deterministic Nonlinear Cost & Deterministic Nonlinear Dynamics

$$
\begin{aligned}
u_0^*, \ldots, u_{N-1}^* = \quad & \underset{u_0, \ldots, u_N}{\operatorname{argmin}} && \sum_{t=0}^{N} c(\mathbf{x}_t, \mathbf{u}_t) \\
& \text{s.t.} \\
\mathbf{x}_1 &= && f(\mathbf{x}_0, \mathbf{u}_0) \\
\mathbf{x}_2 &= && f(\mathbf{x}_1, \mathbf{u}_1) \\
& \ldots \\
\mathbf{x}_N &= && f(\mathbf{x}_{N-1}, \mathbf{u}_{N-1})
\end{aligned}
$$

Arbitrary differentiable functions c, f

iLQR: iteratively approximate solution by solving linearized versions of the problem via LQR

Deterministic Nonlinear Cost & Stochastic Nonlinear Dynamics

$$
\begin{aligned}
u_0^*, \ldots, u_{N-1}^* = \quad & \underset{u_0, \ldots, u_N}{\operatorname{argmin}} && \sum_{t=0}^{N} c(\mathbf{x}_t, \mathbf{u}_t) \\
& \text{s.t.} \\
\mathbf{x}_1 &= f(\mathbf{x}_0, \mathbf{u}_0) + \mathbf{w}_0 \\
\mathbf{x}_2 &= f(\mathbf{x}_1, \mathbf{u}_1) + \mathbf{w}_1 \\
& \ldots \\
\mathbf{x}_N &= f(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) + \mathbf{w}_{N-1}
\end{aligned}
$$

Arbitrary differentiable functions c, f

$\mathbf{w}_t \sim \mathcal{N}(0, W_t)$

iLQG: iteratively approximate solution by solving linearized versions of the problem via LQG

Recall from Guided Policy Search

$$\begin{aligned} \underset{q(\tau)}{\text{argmin}} \quad & \mathbb{E}_{\tau \sim q(\tau)} \left[c(\tau) \right] \\ \text{subject to} \quad & q(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{x}_{t+1}; f_{xt}\mathbf{x}_t + f_{ut}\mathbf{u}_t, \mathbf{F}_t) \\ & \mathrm{KL}(q(\tau) \,||\, q_{\text{prev}}(\tau)) \leq \epsilon \end{aligned}$$

Learn linear Gaussian dynamics

Recall from Guided Policy Search

$$\begin{aligned} \underset{q(\tau)}{\text{argmin}} \quad & \mathbb{E}_{\tau \sim q(\tau)} \left[c(\tau) \right] \\ \text{subject to} \quad & q(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{x}_{t+1}; f_{xt}\mathbf{x}_t + f_{ut}\mathbf{u}_t, \mathbf{F}_t) \\ & \mathrm{KL}(q(\tau) \,||\, q_{\text{prev}}(\tau)) \leq \epsilon \end{aligned}$$

⇐ Learn linear Gaussian dynamics

⇓

$$\begin{aligned} q_{\text{gps}}(\tau) = \underset{q(\tau)}{\text{argmin}} \quad & \mathbb{E}_{\tau \sim q(\tau)} \left[c(\tau) \right] - \mathcal{H}(q(\tau)) \\ \text{subject to} \quad & q(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{x}_{t+1}; f_{xt}\mathbf{x}_t + f_{ut}\mathbf{u}_t, \mathbf{F}_t) \end{aligned}$$

Recall from Guided Policy Search

$$\begin{aligned} \underset{q(\tau)}{\text{argmin}} \quad & \mathbb{E}_{\tau \sim q(\tau)} \left[c(\tau) \right] \\ \text{subject to} \quad & q(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{x}_{t+1}; f_{xt}\mathbf{x}_t + f_{ut}\mathbf{u}_t, \mathbf{F}_t) \\ & \mathrm{KL}(q(\tau) \,||\, q_{\text{prev}}(\tau)) \leq \epsilon \end{aligned}$$

⇐ Learn linear Gaussian dynamics

⇓

$$q_{\text{gps}}(\tau) = \underset{q(\tau)}{\text{argmin}} \quad \mathbb{E}_{\tau \sim q(\tau)} \left[c(\tau) \right] - \mathcal{H}(q(\tau))$$

$$\text{subject to} \quad q(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{x}_{t+1}; f_{xt}\mathbf{x}_t + f_{ut}\mathbf{u}_t, \mathbf{F}_t)$$

⇓

$$q_{\text{gps}}(\tau) = q(\mathbf{x}_0) \prod_{t=0}^{T-1} q(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) q(\mathbf{u}_t | \mathbf{x}_t)$$

↑ ↑

Linear Gaussian dynamics and controller

Recall from Guided Policy Search

$$\begin{aligned} \underset{q(\tau)}{\text{argmin}} \quad & \mathbb{E}_{\tau \sim q(\tau)} \left[c(\tau) \right] \\ \text{subject to} \quad & q(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{x}_{t+1}; f_{xt}\mathbf{x}_t + f_{ut}\mathbf{u}_t, \mathbf{F}_t) \\ & \mathrm{KL}(q(\tau) \,||\, q_{\text{prev}}(\tau)) \leq \epsilon \end{aligned}$$

Learn linear Gaussian dynamics

$$q_{\text{gps}}(\tau) = \underset{q(\tau)}{\text{argmin}} \quad \mathbb{E}_{\tau \sim q(\tau)} \left[c(\tau) \right] - \mathcal{H}(q(\tau))$$

$$\text{subject to} \quad q(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{x}_{t+1}; f_{xt}\mathbf{x}_t + f_{ut}\mathbf{u}_t, \mathbf{F}_t)$$

$$q_{\text{gps}}(\tau) = q(\mathbf{x}_0) \prod_{t=0}^{T-1} q(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) q(\mathbf{u}_t | \mathbf{x}_t)$$

Linear Gaussian dynamics and controller

$q_{\text{prev}} = q_{\text{gps}}$

Run controller on the robot
Collect trajectories

Recall from Guided Policy Search

$$\begin{aligned} \underset{q(\tau)}{\text{argmin}} \quad & \mathbb{E}_{\tau \sim q(\tau)} \left[c(\tau) \right] \\ \text{subject to} \quad & q(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{x}_{t+1}; f_{xt}\mathbf{x}_t + f_{ut}\mathbf{u}_t, \mathbf{F}_t) \\ & \mathrm{KL}(q(\tau) \,||\, q_{\text{prev}}(\tau)) \leq \epsilon \end{aligned}$$

⇐ Learn linear Gaussian dynamics

⇓

$$\begin{aligned} q_{\text{gps}}(\tau) = \underset{q(\tau)}{\text{argmin}} \quad & \mathbb{E}_{\tau \sim q(\tau)} \left[c(\tau) \right] - \mathcal{H}(q(\tau)) \\ \text{subject to} \quad & q(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{x}_{t+1}; f_{xt}\mathbf{x}_t + f_{ut}\mathbf{u}_t, \mathbf{F}_t) \end{aligned}$$

$$\mathrm{KL}\left(q(\tau) \,||\, \frac{\exp(-c(\tau))}{Z} \right)$$

Given a fixed cost function c, the linear Gaussian controllers that GPS computes induce a distribution of trajectories close to

$$\frac{\exp(-c(\tau))}{Z}$$

i.e. good for importance sampling of the partition function Z

Guided Cost Learning [rough sketch]

Collect demonstration trajectories D
Initialize cost parameters θ_0

Do forward optimization using Guided Policy Search for cost function $c_{\theta_t}(\tau)$
and compute linear Gaussian distribution of trajectories $q_{\text{gps}}(\tau)$

$$\nabla_\theta \, L(\theta) = -\frac{1}{|D|} \sum_{\tau \in D} \nabla_\theta \, c_\theta(\tau) + \underbrace{\sum_{\tau} \, p(\tau|\theta) \, \nabla_\theta c_\theta(\tau)}$$

Importance sample trajectories from $q_{\text{gps}}(\tau)$

$$\theta_{t+1} = \theta_t + \gamma \nabla_\theta L(\theta_t)$$

Regularization of learned cost functions

$$g_{\text{lcr}}(\tau) = \sum_{x_t \in \tau} [(c_\theta(x_{t+1}) - c_\theta(x_t)) - (c_\theta(x_t) - c_\theta(x_{t-1}))]^2$$

$$g_{\text{mono}}(\tau) = \sum_{x_t \in \tau} [\max(0, c_\theta(x_t) - c_\theta(x_{t-1}) - 1)]^2$$

Source: https://www.youtube.com/watch?v=hXxaepw0zAw&ab_channel=RAIL

Today's agenda

- Learning linear rewards from trajectory demonstrations in 2D
- Learning nonlinear rewards from trajectory demonstrations in 2D
- Guided cost learning in any D
- Updating distributions over reward parameters using preference elicitation

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

Active Preference-Based Learning of Reward Functions

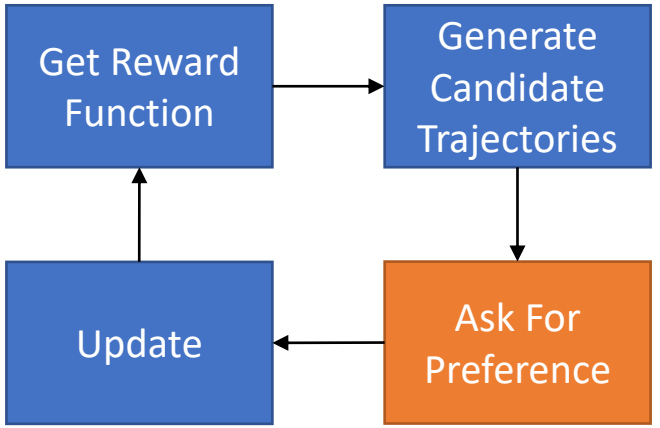
By: Dorsa Sadigh, Anca D. Dragan, Shankar Sastry, and Sanjit A. Seshia

Active Preference-Based Learning of Reward Functions

Dorsa Sadigh, Anca Dragan, Shankar Sastry, Sanjit Seshia

Source: https://rss2017.lids.mit.edu/static/slides/04.mp4

Preference Based Learning



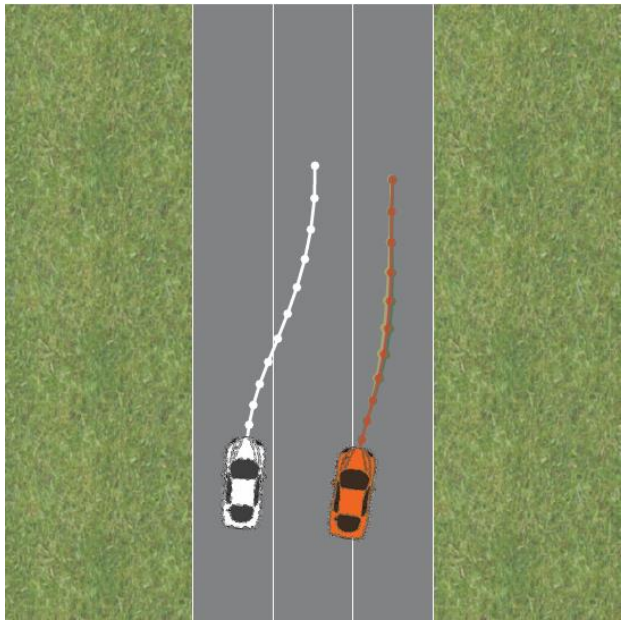
Preference Based Learning Diagram

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

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

- Reward at a state:

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

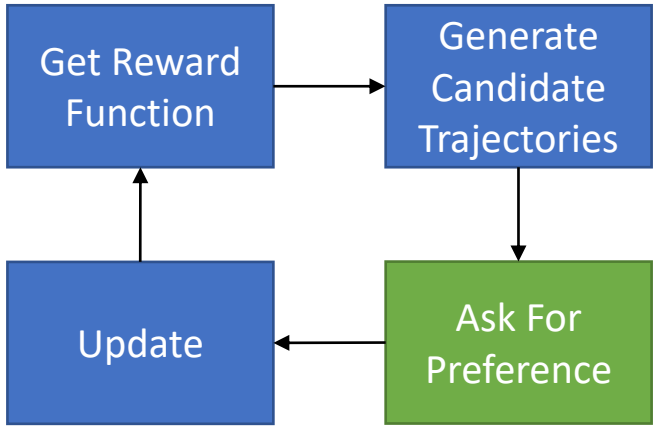
- Reward over a finite trajectory:

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

Preference

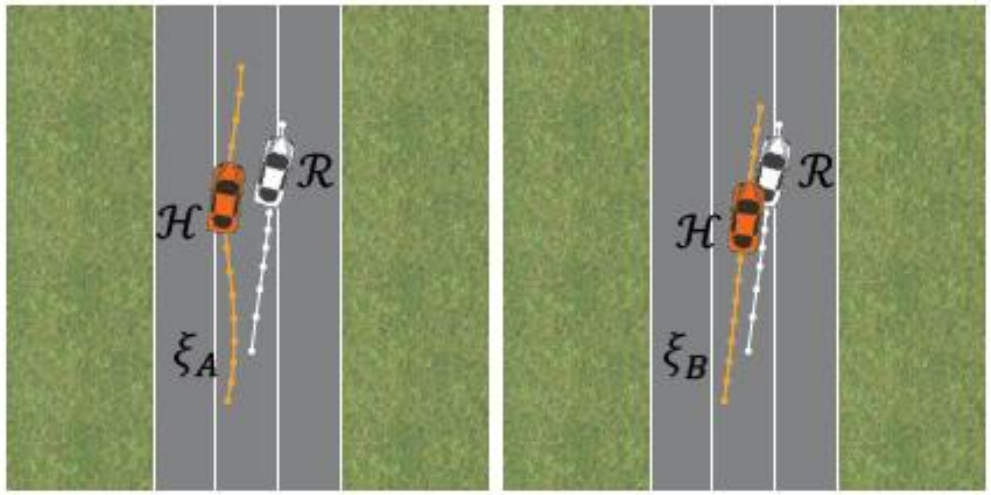
- Given 2 trajectories ξ_A and ξ_B
- Preference variable I

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

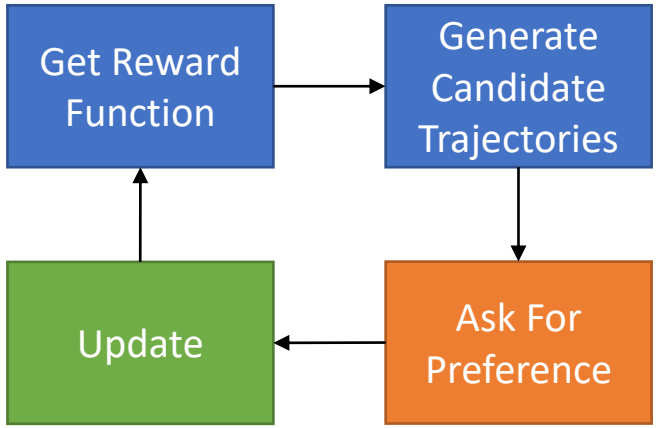


Preference Based Learning Diagram

ξ_A or $\xi_B \rightarrow I$



Weight Update



Preference Based Learning Diagram

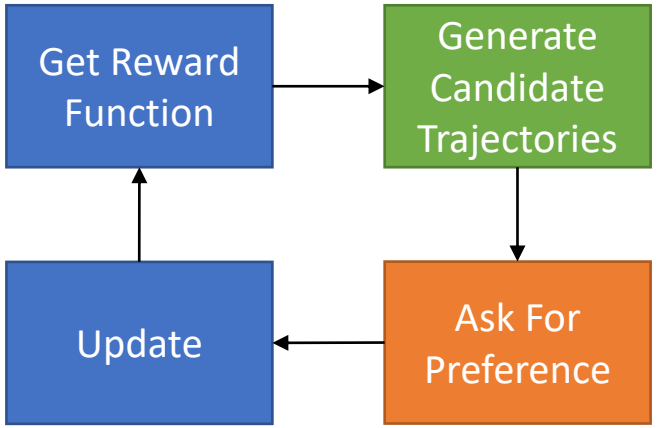
- Assume probabilistic model: weights come from a distribution
- Preference is noisy:

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

- Some simplification:

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

Generate Trajectories

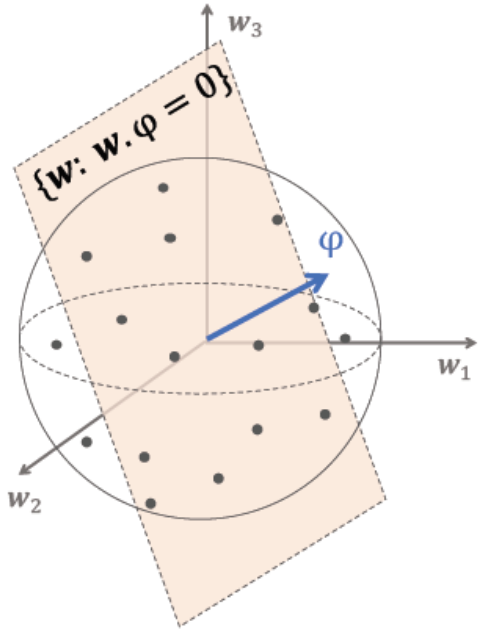


Preference Based Learning Diagram

- Two feasible trajectories: ξ_A, ξ_B
- Want each update to give most information
  - Maximize minimum volume removed with a query:

$$\max_{\xi_A, \xi_B} \min(\mathbb{E}_w[1 - f_\varphi(w)], \mathbb{E}_w[1 - f_{-\varphi}(w)])$$

  - A binary query corresponds to selecting sides of hyperplane $\boldsymbol{w}^T \varphi = 0$
  - Response increases probability of weights on one side of hyperplane and decreases the other side.



$* = E_{w\sim p(w)}\left[1 - p(I=1|w,\phi)\right]$

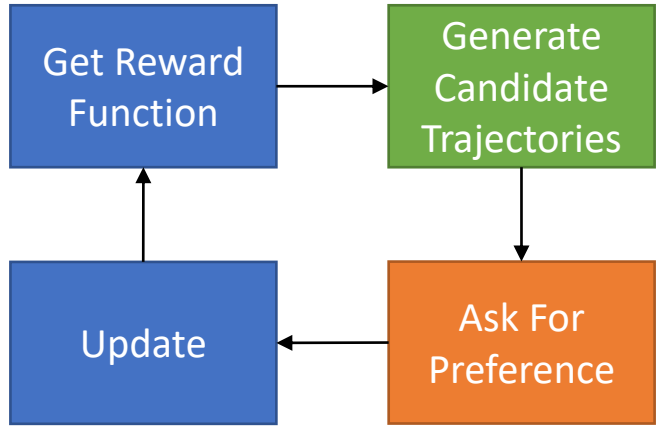
Generate Trajectories

$= 1 - \int p(I=1|w,\phi)p(w)dw$

$= 1 - \int p(I=1,w|\phi)\,dw$

$= 1 - p(I=1|\phi)$

$= p(I=1|\phi)$



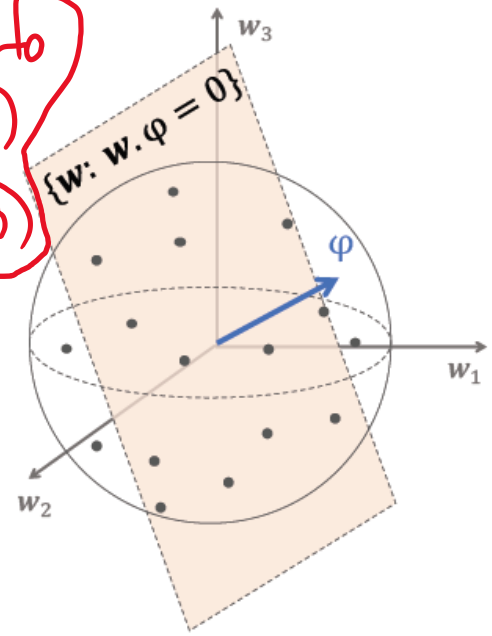
Preference Based Learning Diagram

- Two feasible trajectories: ξ_A, ξ_B
- Want each update to give most information
  - Maximize minimum volume removed with a query:

$$\max_{\xi_A,\xi_B} \min(\mathbb{E}_w[1 - f_\varphi(w)], \mathbb{E}_w[1 - f_{-\varphi}(w)])$$

$*$

Preferable to query (0.5, 0.5) than (0.1, 0.9)

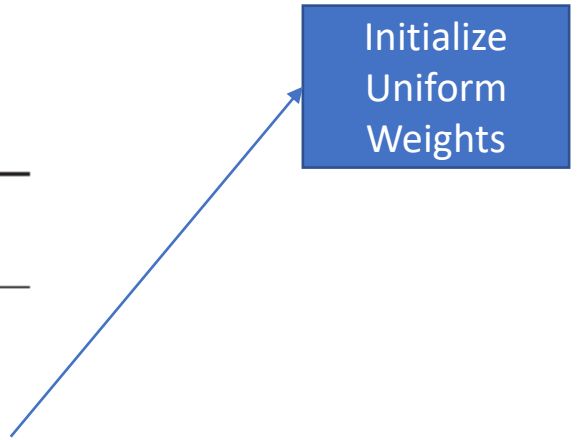


  - A binary query corresponds to selecting sides of hyperplane $\boldsymbol{w^T}\varphi = 0$
  - Response increases probability of weights on one side of hyperplane and decreases the other side.

Algorithm Summary

Algorithm 1 Preference-Based Learning of Reward Functions

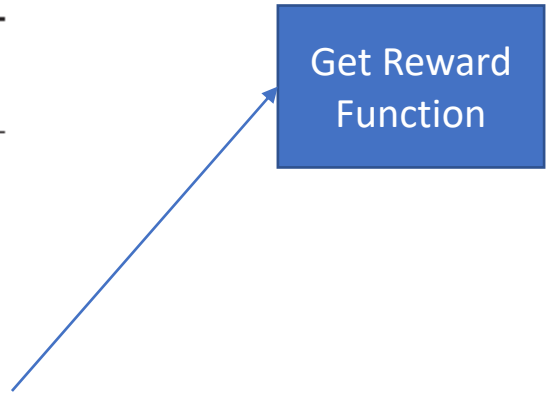
Input: Features ϕ, horizon N, dynamics f, *iter*
Output: Distribution of $\mathbf{w}$: $p(\mathbf{w})$
Initialize $p(\mathbf{w}) \sim \text{Uniform}(B)$, for a unit ball B
While $t < iter$:
  $W \leftarrow M$ samples from AdaptiveMetropolis($p(\mathbf{w})$)
  $(x^0, \mathbf{u}_R, \mathbf{u}_H^A, \mathbf{u}_H^B)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

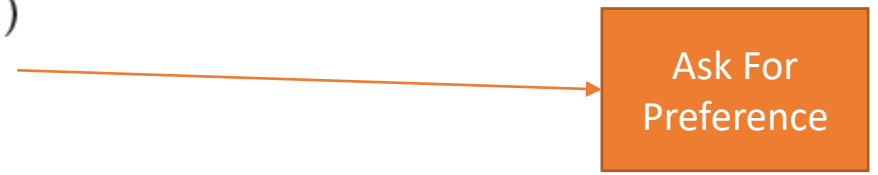
Algorithm 1 Preference-Based Learning of Reward Functions

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

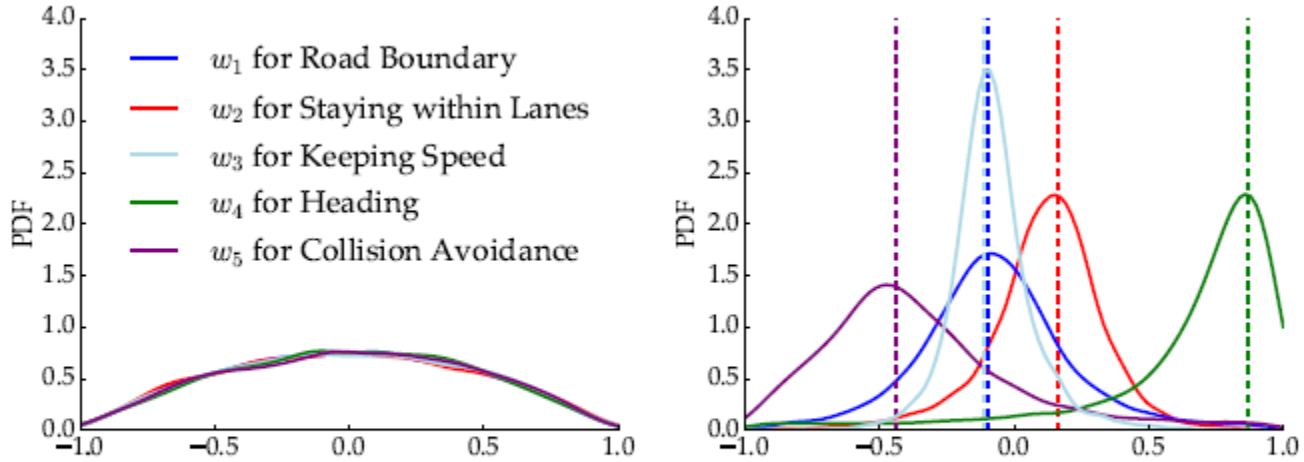
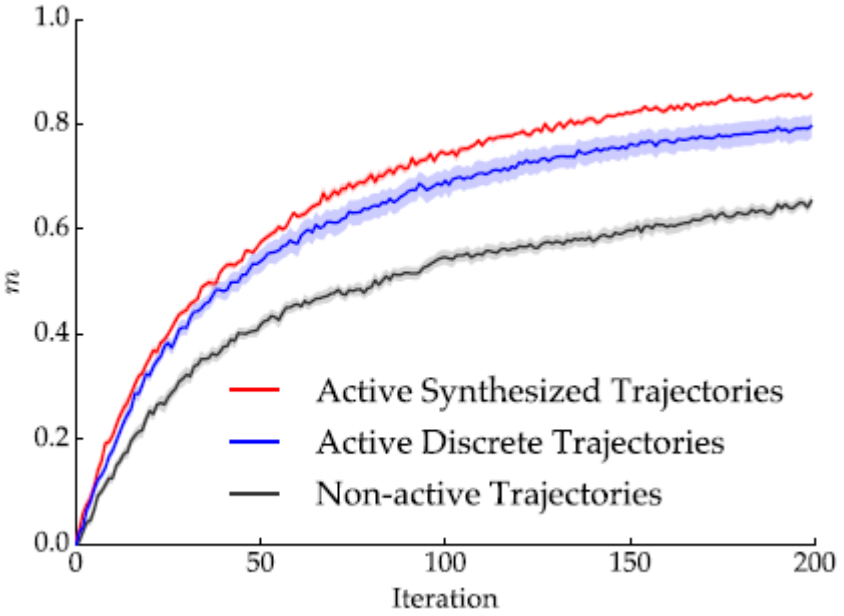


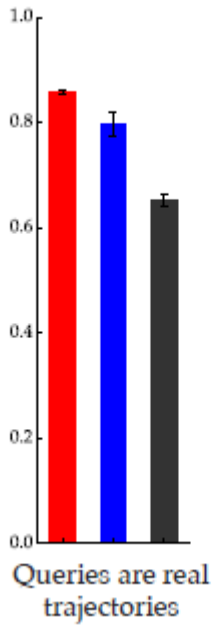
Fig. 3: Distribution over all weights before/after convergence. The dotted lines show the ground truth of the weights.

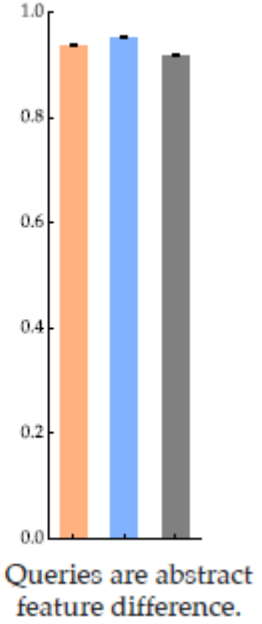
Results

- 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

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

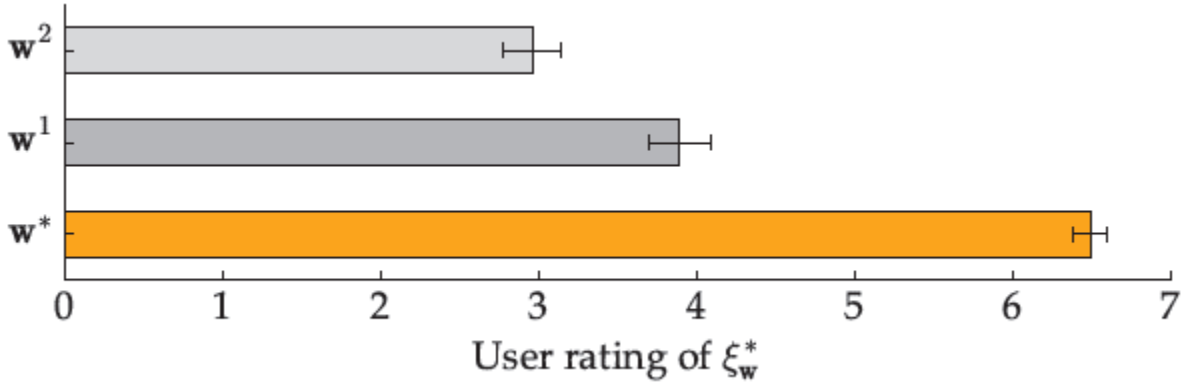


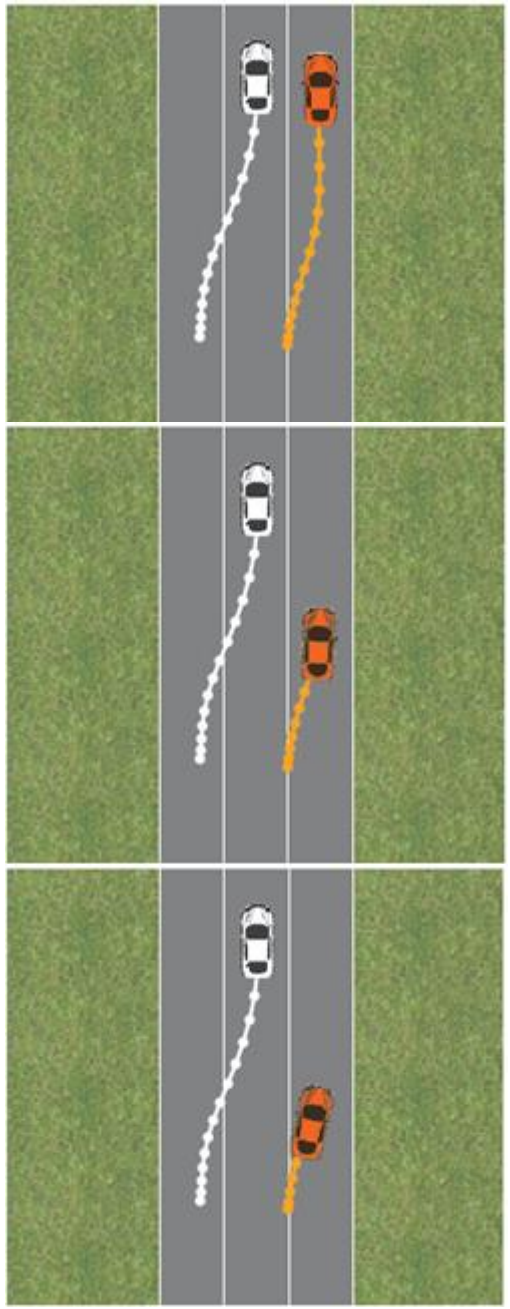




Results

- Perturbation of weights
  - Learned weights: w^*
  - Slightly perturbed weights: w^1
  - Largely perturbed weights: w^2
- Users prefer w^*





Check out their library for preference learning

https://github.com/Stanford-ILIAD/APReL