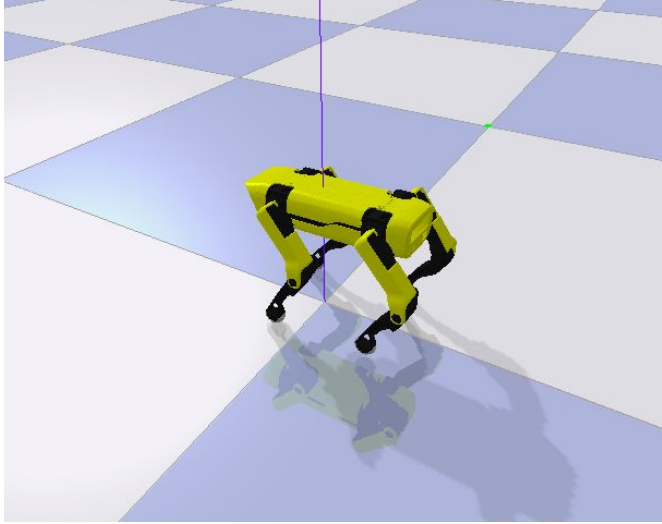
Maximum Entropy Inverse Reinforcement Learning (2008)

Paper by Brian D. Ziebart, Andrew Maas, J.Andrew Bagnell, and Anind K. Dey at CMU

Presentation by Tyler Ingebrand on Oct. 13

Problem - Reward tuning in RL

- Given an MDP with a reward function, assume we can use RL to generate an optimal policy
- MDP can model a problem we want to solve, but how do you generate a reward function?
- Slight variations in reward function components and weights lead to wildly different policies

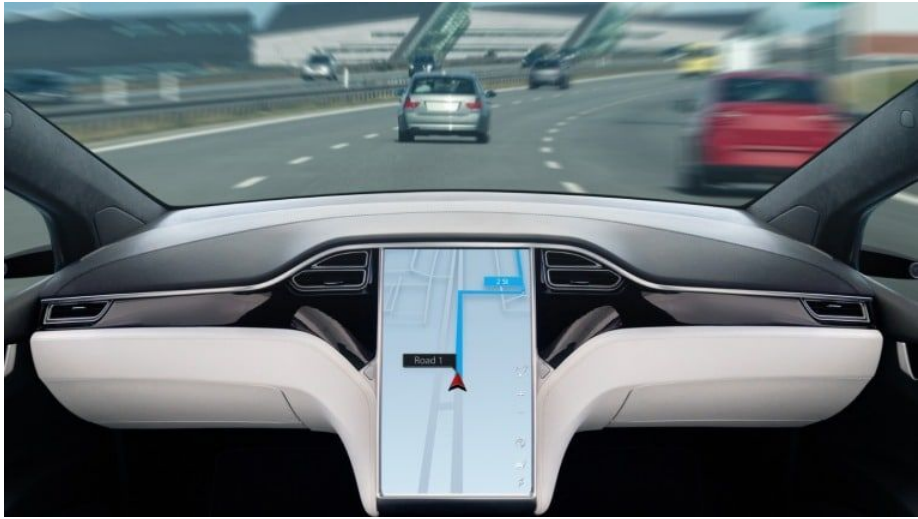


$$r_t = v_x + 25\frac{T_s}{T_f} - 50\,\hat{y}^2 - 20\,\theta^2 - 0.02\sum_i {u_{t-1}^i}^2$$

- Maximize forward velocity
- Reward for every time step the robot does not fall
- Keep the torso at the desired height
- Keep the torso parallel to the ground
- Minimize torque applied to the joints

Solution - Learn a reward function

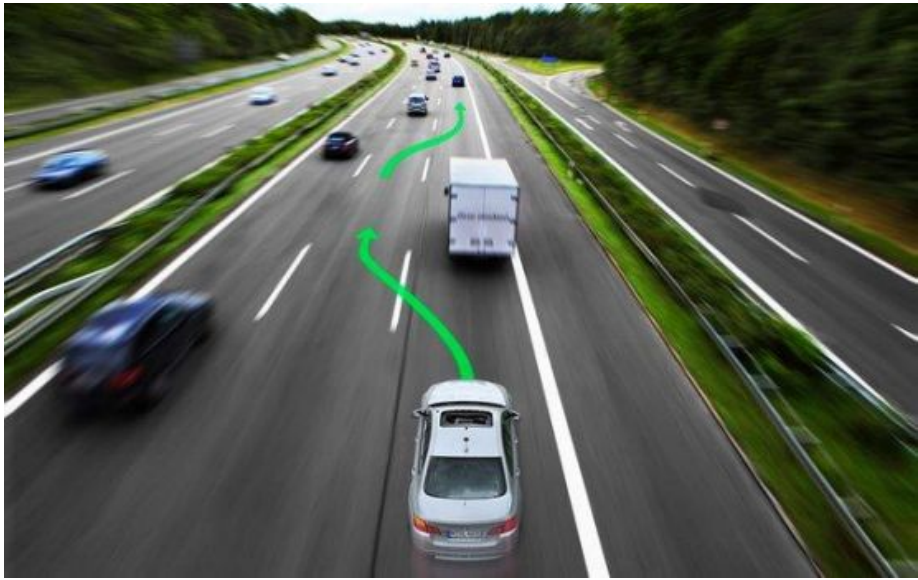
- In many cases, we can treat humans as optimal agents and try to mimic humans
- If we can learn a reward function from examples, then we can use that reward function to generate a policy using RL



Self-driving cars are a good example. Tuning a reward function would be nearly impossible. But we have thousands of hours of data on human driving behavior.

Formal problem specification

- Given trajectories of an optimal agent in the form (s_0, a_0), (s_1, a_1), (s_2, a_2)...
- Output a reward function such that an optimal agent would reproduce those actions in those states (as closely as possible)
- Extremely difficult - there are infinite possible reward functions to generate a given policy



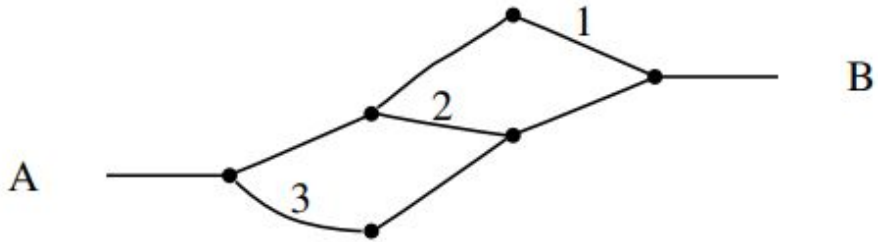
An example trajectory for a self-driving car

Prior work

- Learning agents for uncertain environments (1998)
  - Original problem definition - Reward function from expertise
- Apprenticeship learning via inverse reinforcement learning (2004)
  - Assumes the reward function is a linear combination of some features with some weights
  - Chooses the weights such that the difference in value between the expert policy and the current policy is greatest; then re-optimizes the policy. This makes the policy get iteratively closer to the expert
- Maximum margin planning (2006)
  - Similar to the above but applies gradient descent
- Bayesian Inverse Reinforcement Learning (2007)
  - Uses hill climbing over reward functions
  - Has to train a policy via value iteration every iteration - very slow

Prior work limitations

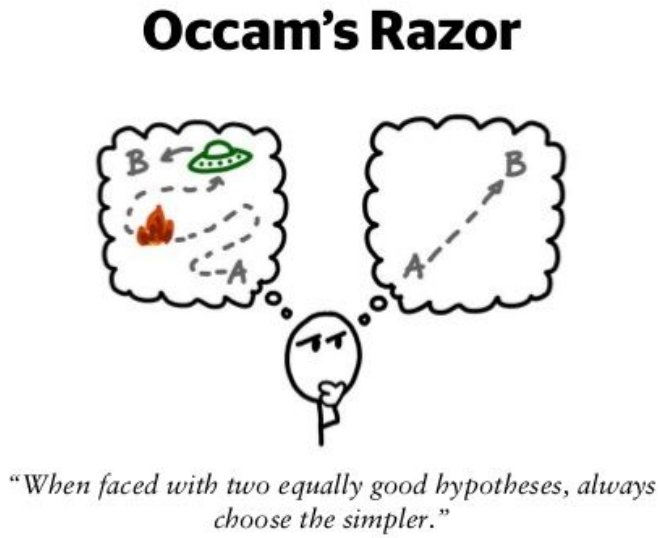
- Fails when the expert agent is not optimal, which is typical in human generated data, because it leads to ambiguity on which reward function is optimal
- Many policies may represent the same trajectories generated via the expert. There is no good way to disambiguate previously.
- Trajectories that branch earlier are favored



Given 3 paths from A to B, we should consider each path separately in terms of total reward. Prior work looks at the action taken at each step, and thus fails by favoring paths that branch soonest.

Principal of maximum entropy

- Given prior information about a distribution, the best approximation is the distribution matching the data with the largest uncertainty
- This distribution makes the least assumptions about the true distribution
- Application of Occam's Razor:
  - "The simpler explanation is to be preferred"
- Another interpretation is to not overfit the approximate distributribution to the data



Main Contribution - Apply the principle of maximum entropy to inverse RL

Probability of

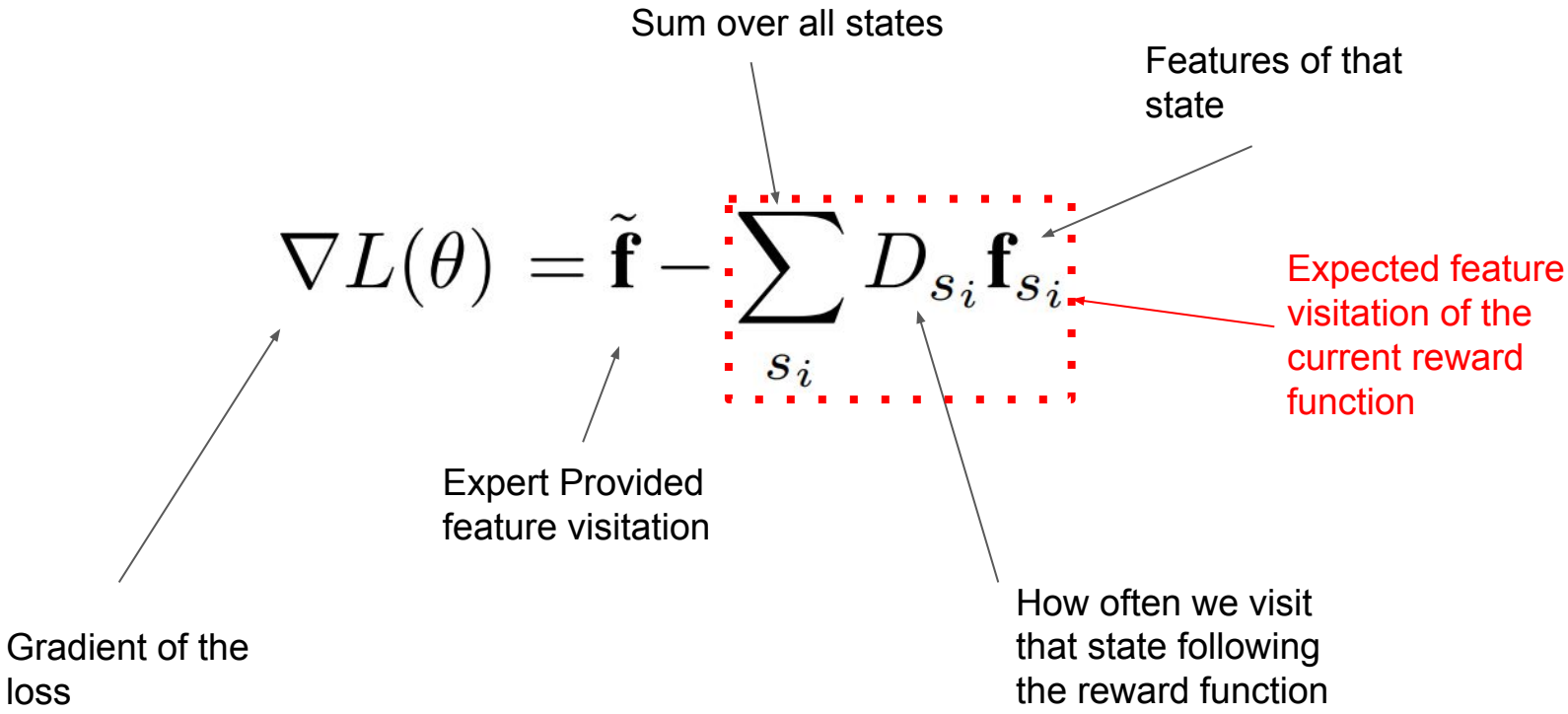
$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{\text{examples}} \log P(\tilde{\zeta}|\theta, T)$$

Optimal reward function weights

Expert Trajectory (Zeta)

Given the reward function weights and Transition function

Main Contribution - Solve via gradient descent



Minimize difference between expert feature visitation and our feature visitation

Main Contribution - Solve via gradient descent

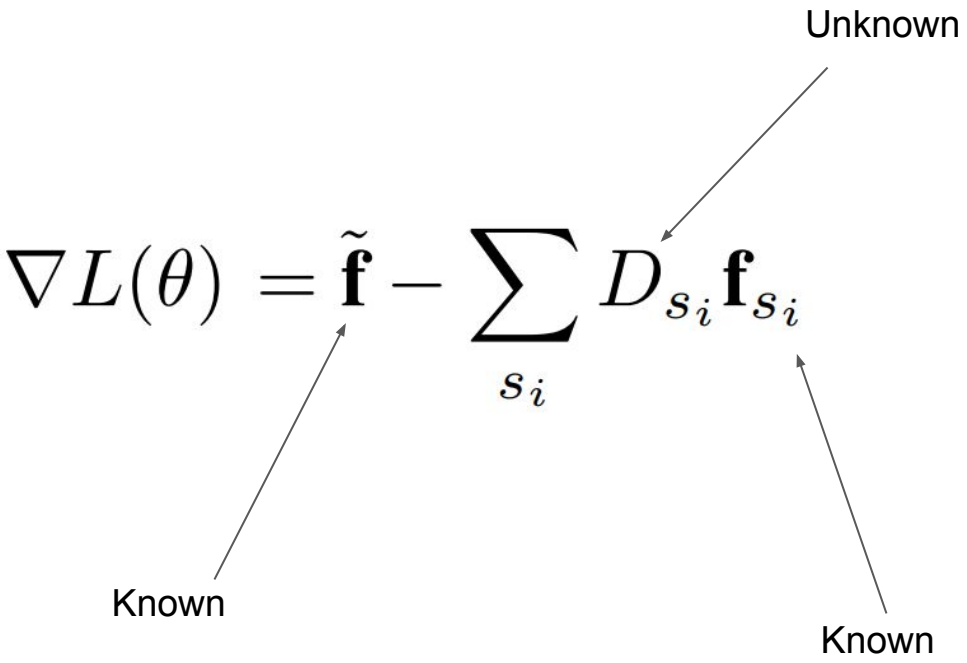
$$\nabla L(\theta) = \tilde{\mathbf{f}} - \sum_{s_i} D_{s_i} \mathbf{f}_{s_i}$$

Expected feature visitation of the current reward function

Expert Provided feature visitation

Why use the difference in feature visitation rather than the difference in state visitation? State visitation is subject to the transition function, whereas feature visitation is the “Goal” of the agent. Even with “unlucky” transitions, the agent should return to the correct features.

Main Contribution - Solve via gradient descent



Implementation details

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

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

Summing frequencies

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

Computes probability mass of each branch

Computes action probabilities for each state

Given initial states and action probabilities, calculates the probabilities of all states at all timesteps

Sum the state probabilities over all timesteps to get state visitation

Results - Taxi data from Pittsburgh

Results

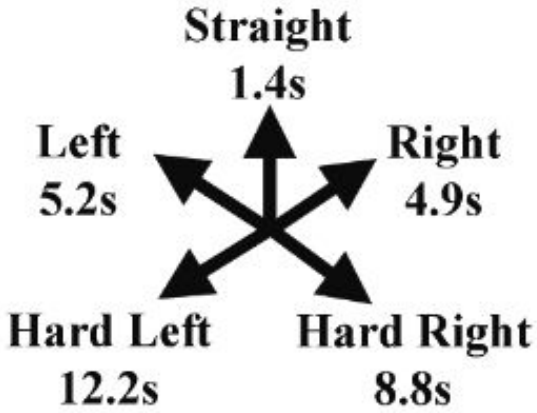
Given a goal from the expert dataset, what percent of the route follows the expert?

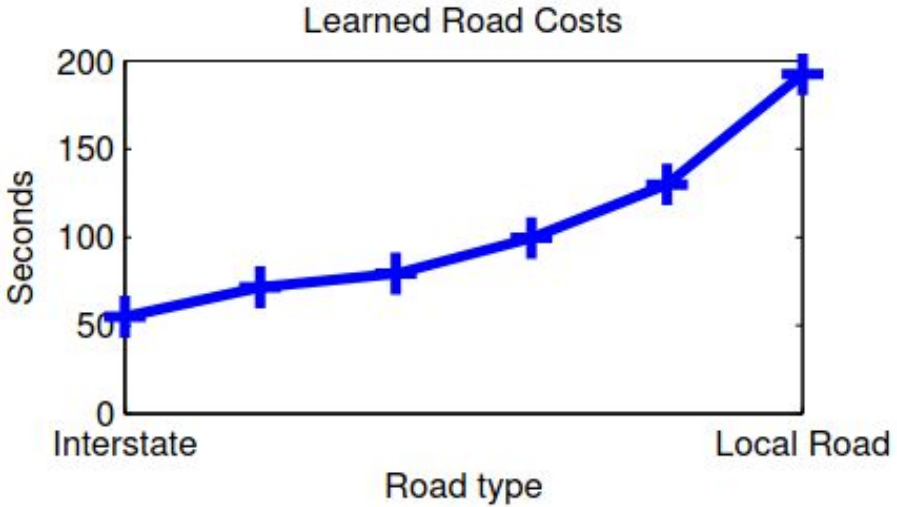
What percent of those paths are 90% the same as an expert?

	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**

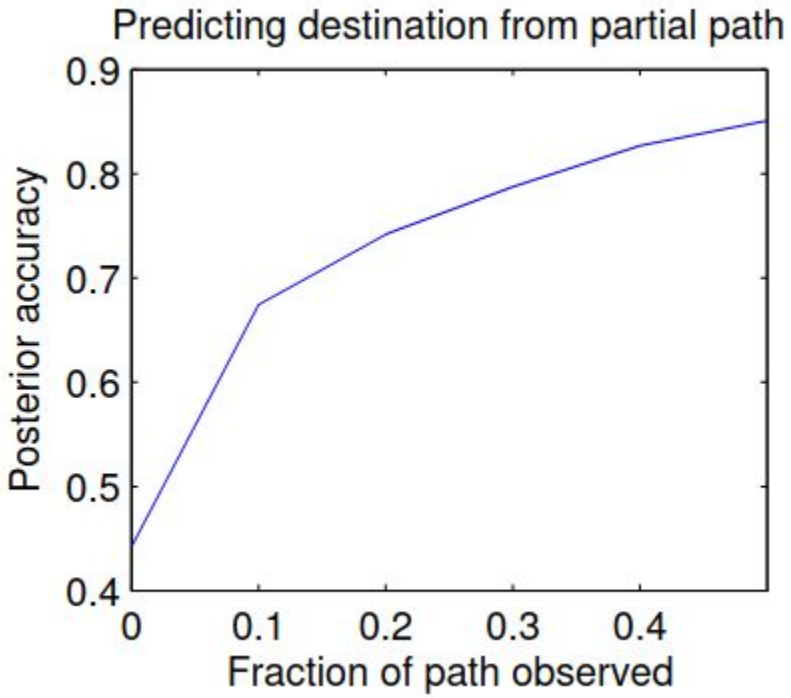
How likely are expert paths in their model?

Results - Costs of actions (in seconds)





Results - Using the model to predict destinations



Strengths

- Learns the most general reward function to explain the expert trajectories
- Can handle sub-optimal expert trajectories
- Handles stochastic environments

Limitations

- Iterates over all states and actions, therefore only works on discrete problems
- Can only produce action probabilities for visited states - cannot produce a policy for unseen states
- Requires human expertise - How to get this for robotics?

Future work

- Continuous Spaces
- Extrapolate to unseen states
- More example datasets to test on - Driving dataset alone is not very convincing
- Generate a dense reward function from a sparse reward function

Extended readings

- *A Survey of Inverse Reinforcement Learning: Challenges, Methods and Progress*
  - Provides an overview of IRL methods as of 2020
- *Maximum Entropy Deep Inverse Reinforcement Learning*
  - Extends this work to neural networks
- *Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals*
  - Extends this work to continuous state spaces

Summary

- Problem: Reward tuning is hard and boring
- Solution: Learn reward functions from experts
- Prior work cannot handle sub-optimal experts
- Improve reward function approximation by using the principle of maximum entropy
- **Key Takeaway** - Principle of maximum entropy can improve many solutions that involve uncertainty