Apprenticeship Learning via Inverse Reinforcement Learning

Presenter: Vanya Cohen

10/26/2021

Motivation and Main Problem

1-5 slides

High-level description of problem being solved

Why is the problem important?

- its significance towards general-purpose robot autonomy
- its potential application and societal impact of the problem

Technical challenges arising from the problem

- the role of the AI and machine learning in tackling this problem

High-level idea of why prior approaches didn't already solve

Key insight(s) (try to do in 1-3) of the proposed work

Learning From a Human Expert

AI Apprentice

Human Experts

(Left) https://medium.com/waymo/simulation-how-one-flashing-yellow-light-turns-into-thousands-of-hours-of-experience-a7a1cb475565
(Right) https://medium.com/waymo/simulation-how-one-flashing-yellow-light-turns-into-thousands-of-hours-of-experience-a7a1cb475565

Learning From a Human Expert

- It's often difficult to explicitly specify a reward function for a given task.
- But expert behavior attempts to maximize an implicit reward function for a task.
- Key Insight: Instead of specifying a reward function, learn to recover an approximation of the expert's reward function from some demonstrations – Inverse Reinforcement Learning (Ng and Russell, 2000).
- Apprenticeship Learning: umbrella term for "learning by watching, imitation learning, or learning from demonstration" (Abbeel and Ng, 2004).

Why Learn Reward Functions?

- For example, we could learn to mimic trajectories directly…
- Or the policy π or value function V^{π}…
- But we can always recover the policy or value function from the reward function.
- The reward function is compact, and potentially transferable.

http://www.esa.int/spaceinimages/ESA_Multimedia/Copyright_Notice_Images

Problem Setting

1 or more slides

Problem formulation, key definitions and notations

- Be precise -- should be as formal as in the paper

Problem Setting: Learning in a MDP\R

- S — States
- A — Actions
- T — Transition matrix
- γ — Discount factor
- D — Initial state distribution

MDP\R (MDP without a Reward Function)

Problem Setting: Learning in a MDP\R

- S — States
- A — Actions
- T — Transition matrix
- γ — Discount factor
- D — Initial state distribution

→ MDP\R (MDP without a Reward Function)

- R* — Reward function

→ The “true” reward function; will learn an approximation from a demonstration set.

|R*| <= 1

Context / Related Work / Limitations of Prior Work

1 or more slides

Which other papers have tried to tackle this problem or a related problem?

- The paper's related work is a good start, but there may be others
- What is the key limitations of prior work(s)?

Related Work

Approach or Topic	Papers	Difference or Limitation
Policy “Mimicking” Through Supervised Learning	Sammut et al. (1992); Kuniyoshi et al. (1994); Demiris & Hayes (1994); Amit & Mataric (2002); Pomerleau (1989)	Cannot be easily transferred as the policy is directly learned. Limited performance guarantees.
Trajectory Replication with Feedback from a Fixed Reward Function	Atkeson & Schaal (1997)	Directly optimizes the trajectory, not the underlying reward function that specifies the trajectory.
Inverse Reinforcement Learning	(Ng & Russell, 2000)	Presents IRL and derives algorithms for learning. In IRL, reward learning is the end goal, not learning an apprentice policy.

Proposed Approach / Algorithm / Method

1-5 slides

Describe algorithm or framework (pseudocode and flowcharts can help)

- What is the optimization objective?
- What are the core technical innovations of the algorithm/framework?

Implementation details should be left out here, but may be discussed later if its relevant for limitations / experiments

Methods

Methods

- Given an initial set of expert demonstrations, we need to learn a reward function which can then be used to find a high-performing policy for the given task.
- The paper develops two algorithms for this problem, both using the same underlying reward function model.

Methods (High-level Algorithm)

- Start with an initial set of expert demonstrations and a random policy for the agent.
- Calculate state feature expectations for the expert and the initial policy.
- Optimize the reward function to maximize the performance of the expert (subject to constraints), and then train an improved policy using RL.
- Repeat algorithm with this new policy, and run until convergence.

Theory (if relevant)

What are the assumptions made for the theory? Are these reasonable? Realistic?

If the theory build strongly on other prior theory / results, reference those and state them here.

Theory (if relevant, continued)

State main results formally

Give proof sketches

Refer students to the full proofs in paper

Methods: Definitions

- Formally, given a set of expert demonstrations, an MDP/R, a state featurization function ϕ(s), and expert feature expectations μ_E, need to learn to approximate R*.
- The learning algorithm will guarantee the reward function can be used to generate a good policy, but makes no guarantees about similarity to R*.

Methods: Reward Model

R* is approximated using a linear model.

$$R^*(s) = w^* \cdot \phi(s)$$

$\phi(s)$ → State feature vectors

w^* → w* is a weight vector

$$w^* \in \mathbb{R}^k \quad \|w^*\|_1 \leq 1$$

Algorithms: Expert Feature Expectations

- Compute the feature expectation of the expert policy

$$m \text{ trajectories } \{s_0^{(i)}, s_1^{(i)}, \ldots\}_{i=1}^{m}$$

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)})$$

Mean over all demonstrations

Discount-factor weighted sum of feature vectors

Methods: Two Algorithms

- First, a straightforward algorithm for optimizing the reward function R. The authors call this algorithm the Max-Margin method.
    - Requires a quadratic programming (QP) solver.
- As an optimization, the authors present a second algorithm “the Projection method.” This resembles the Max-Margin method, with one part changed.
    - Removes the need for a QP solver.

Algorithms: Max-Margin

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Algorithms: Max-Margin

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Initialize random policy, compute feature expectations

Algorithms: Max-Margin

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Optimize w to maximize the margin between the expert and the best policy found thus far.

Algorithms: Max-Margin

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Terminate when desired performance delta is reached

Algorithms: Max-Margin

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

→ Learn a new policy with R by value iteration.

Algorithms: Max-Margin

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Compute feature expectations of the new policy

Algorithms: Max-Margin

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Repeat from the optimization step

Algorithms: Max-Margin

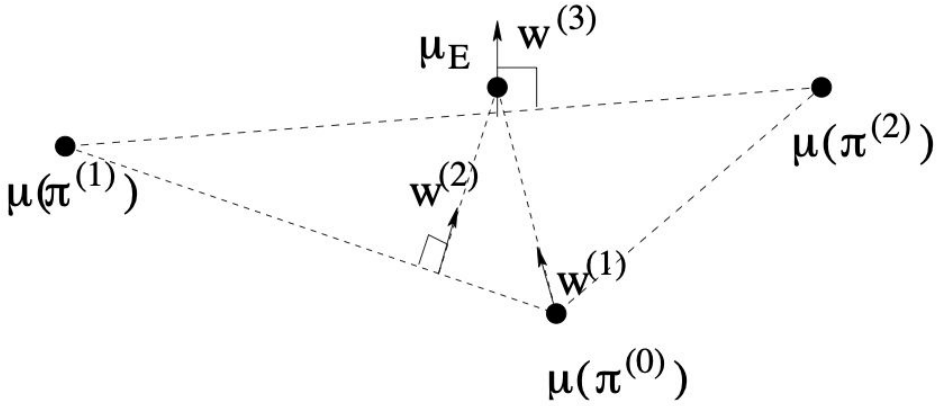


Figure 1. Three iterations for max-margin algorithm.

Algorithms: Projection (Details in Supplement)

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

- Set $\bar{\mu}^{(i-1)} = \bar{\mu}^{(i-2)} + \frac{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T(\mu_E - \bar{\mu}^{(i-2)})}{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T(\mu^{(i-1)} - \bar{\mu}^{(i-2)})}(\mu^{(i-1)} - \bar{\mu}^{(i-2)})$
  (This computes the orthogonal projection of μ_E onto the line through $\bar{\mu}^{(i-2)}$ and $\mu^{(i-1)}$.)
- Set $w^{(i)} = \mu_E - \bar{\mu}^{(i-1)}$
- Set $t^{(i)} = \|\mu_E - \bar{\mu}^{(i-1)}\|_2$

Algorithms: Projection

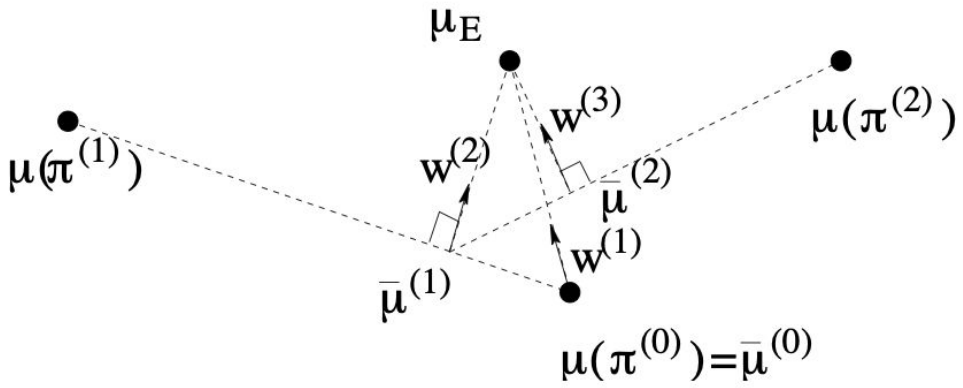


Figure 2. Three iterations for projection algorithm.

Theorem 1 (Convergence) (Proofs in Appendix A)

Theorem 1. *Let an MDP\R, features $\phi : S \mapsto [0,1]^k$, and any $\epsilon > 0$ be given. Then the apprenticeship learning algorithm (both max-margin and projection versions) will terminate with $t^{(i)} \leq \epsilon$ after at most*

$$n = O\left(\frac{k}{(1-\gamma)^2\epsilon^2} \log \frac{k}{(1-\gamma)\epsilon}\right) \tag{14}$$

iterations.

Theorem 2 (Bounding Demonstrations) (Proofs in Appendix A)

Theorem 2. *Let an MDP\R, features $\phi : S \mapsto [0,1]^k$, and any $\epsilon > 0, \delta > 0$ be given. Suppose the apprenticeship learning algorithm (either max-margin or projection version) is run using an estimate $\hat{\mu}_E$ for μ_E obtained by m Monte Carlo samples. In order to ensure that with probability at least $1 - \delta$ the algorithm terminates after at most a number of iterations n given by Eq. (14), and outputs a policy $\tilde{\pi}$ so that for any true reward $R^*(s) = {w^*}^T \phi(s)$ ($\|w^*\|_1 \leq 1$) we have*

$$E[\textstyle\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\tilde{\pi}] \geq E[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\pi_E] - \epsilon, \quad (15)$$

it suffices that

$$m \geq \frac{2k}{(\epsilon(1-\gamma))^2} \log \frac{2k}{\delta}.$$

Experimental Setup

1-3 slides

Description of the experimental evaluation setting

- What is the domain(s), e.g., datasets, tasks, robot hardware setups?
- What are the baseline(s)?
- What scientific hypotheses are tested?

How did the authors evaluate the success of their approach?

- Clear description of the metrics that will be used

Experimental Results

>1 slide

Present the quantitative and qualitative results

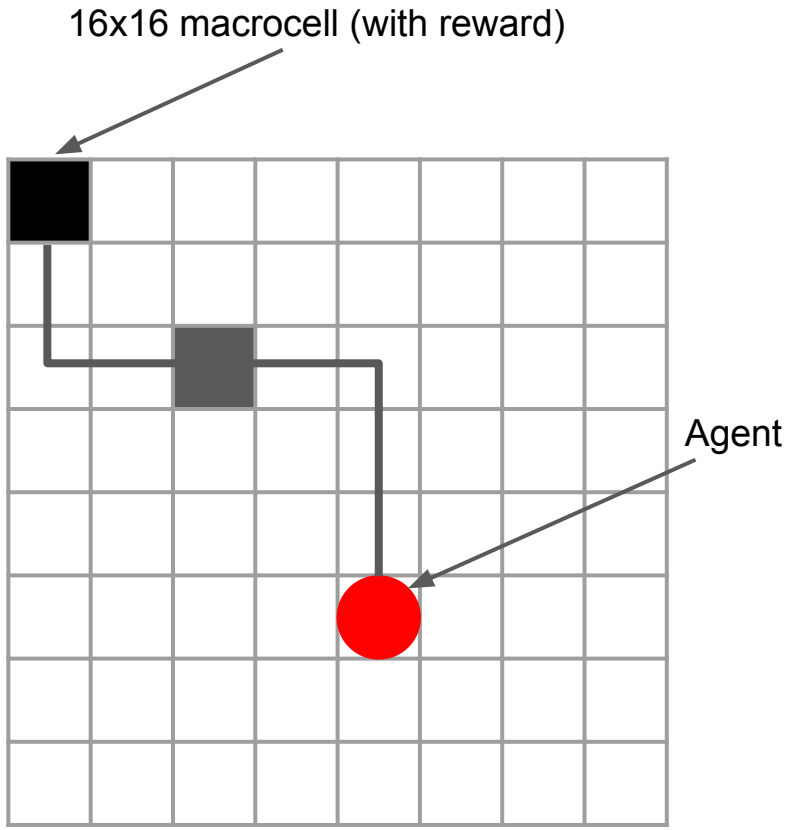
Show figures / tables / plots / robot demos

Pinpoint the most interesting / significant results

Experiments

GridWorld Experiments

- GridWorld Navigation
  - 128x128 GridWorld.
  - Divided into 64 16x16 “macrocells”
    - Some macrocells have positive rewards.
  - 4 Actions (Up, Down, Left, Right) with 30% success rate (moves randomly otherwise).
  - Expert policy provided to sample demonstrations.
  - Multiple and sparse rewards in the environment.



GridWorld Experiments

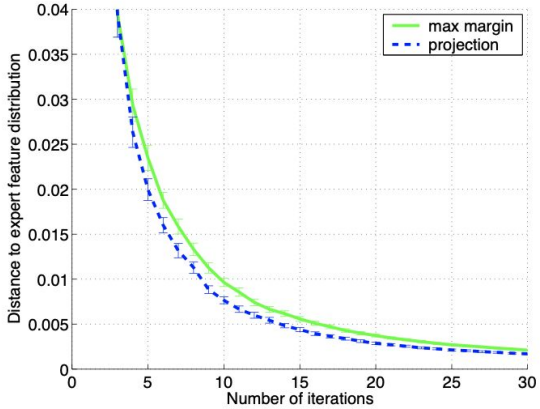


Figure 3. A comparison of the convergence speeds of the max-margin and projection versions of the algorithm on a 128x128 grid. Euclidean distance to the expert's feature expectations is plotted as a function of the number of iterations. We rescaled the feature expectations by $(1-\gamma)$ such that they are in $[0,1]^k$. The plot shows averages over 40 runs, with 1 s.e. errorbars.

GridWorld Experiments

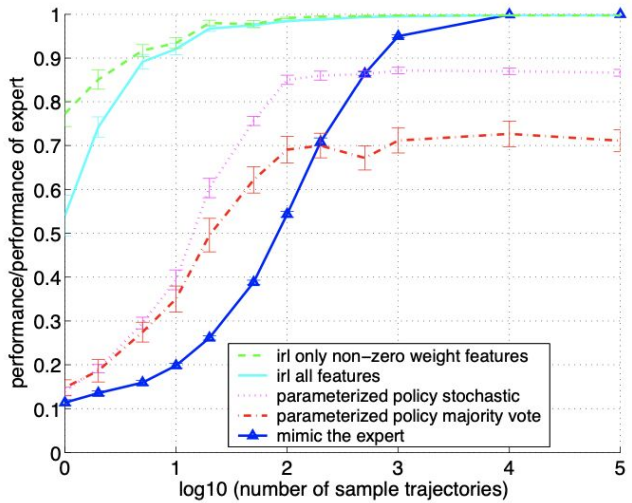


Figure 4. Plot of performance vs. number of sampled trajectories from the expert. (Shown in color, where available.) Averages over 20 instances are plotted, with 1 s.e. errorbars. Note the base-10 logarithm scale on the x-axis.

irl only non-zero features	64 features, pruned of non-zero features
irl all features	all 64 state features
parameterized policy stochastic	sample from distribution of expert actions at each macro state
parameterized policy majority vote	most common expert action at each macro state
mimic the expert	select action of expert if in same state, random otherwise

2D Driving Simulator

- Drives forward at a constant speed, 3 lanes and 2 off-road “lanes.”
- Five actions: steer to any of the 3 lanes, or the two off-road lanes.
- State features:
  - Five variables to indicate whether the car is in a lane
  - Distance to the closest oncoming car in the current lane.

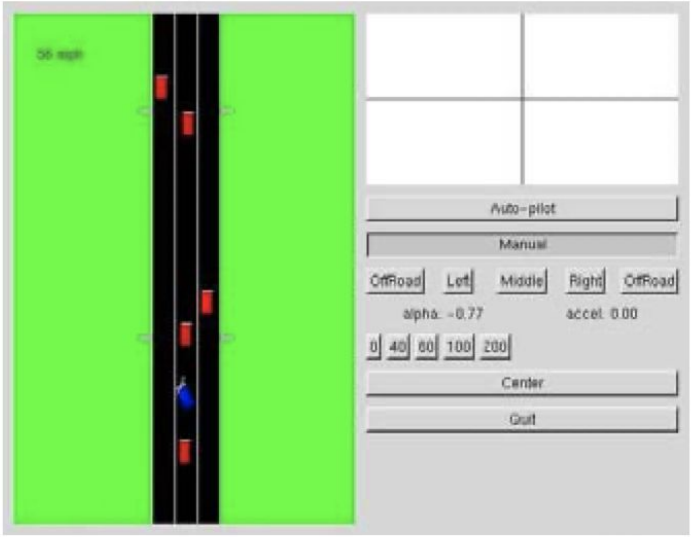


Figure 5. Screenshot of driving simulator.

2D Driving Simulator

- Demonstrations for 5 types of driving.
  - Nice: avoid collisions.
  - Nasty: maximize collisions.
  - Right lane nice: drive in the right lane, but go off-road to avoid collisions.
  - Right lane nasty: drive off-road, but swerve into the left lane to collide.
  - Middle lane: drive in the middle lane and avoid cars.
- Expert demonstrations consist of 2 minutes of driving for each type (1200 environment steps).

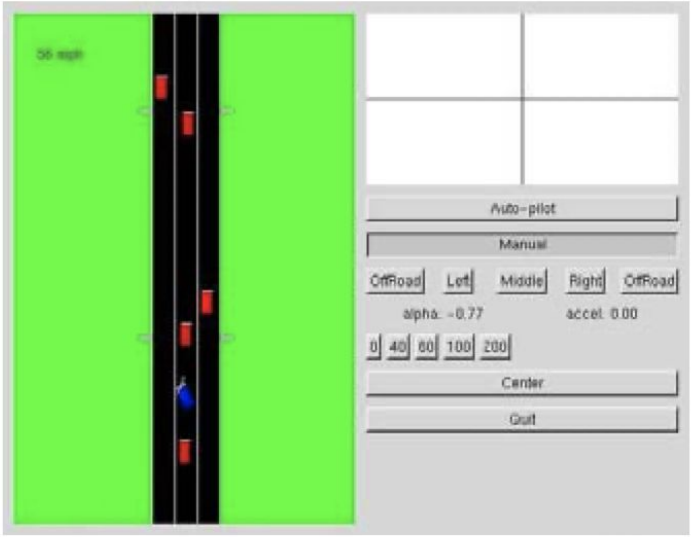
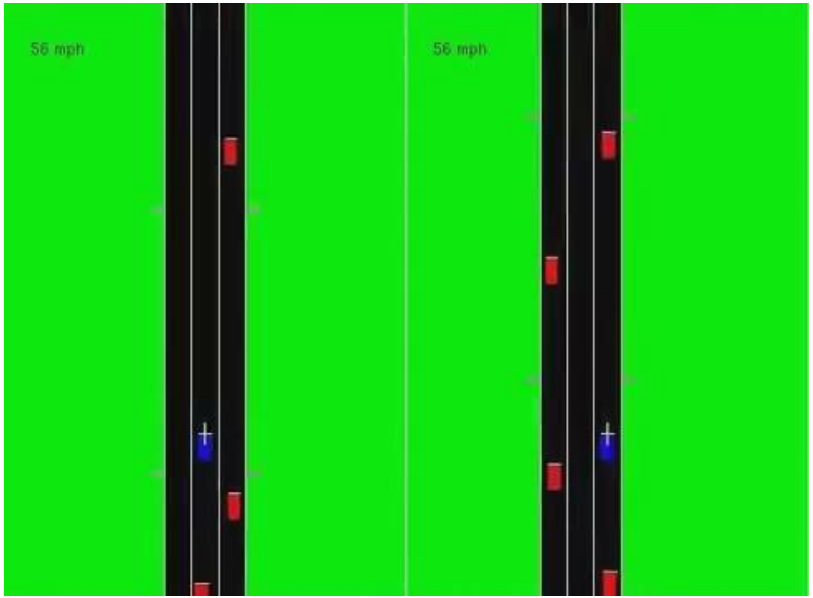
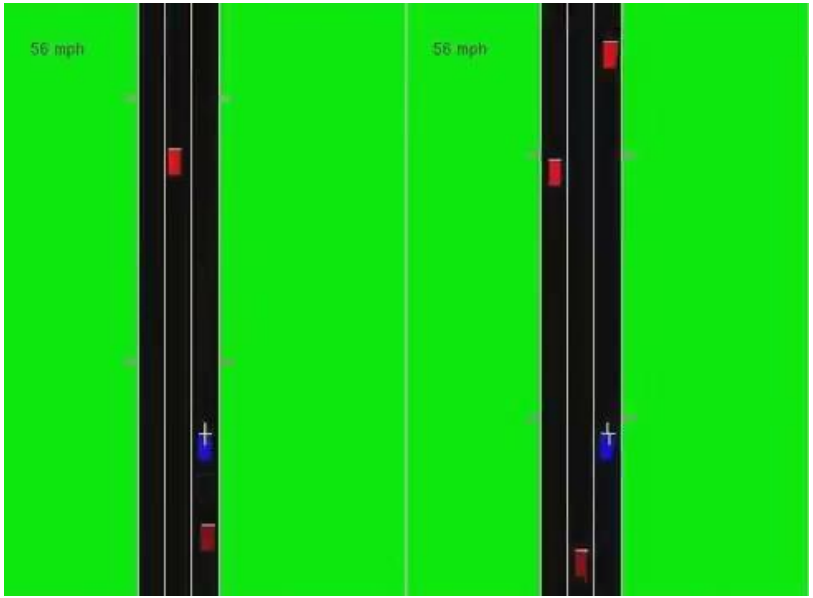


Figure 5. Screenshot of driving simulator.

Driving Experiments: Qualitative





“Nice” Driving (Left) “Nasty” Driving (Right)

Driving Experiments: Learned Reward Weights

Table 1. Feature expectations of teacher $\hat{\mu}_E$ and of selected/learned policy $\mu(\tilde{\pi})$ (as estimated by Monte Carlo). and weights w corresponding to the reward function that had been used to generate the policy shown. (Note for compactness, only 6 of the more interesting features, out of a total of 15 features, are shown here.)

		Collision	Offroad Left	LeftLane	MiddleLane	RightLane	Offroad Right
1	$\hat{\mu}_E$	0.0000	0.0000	0.1325	0.2033	0.5983	0.0658
	$\mu(\tilde{\pi})$	0.0001	0.0004	0.0904	0.2287	0.6041	0.0764
	$\tilde{w}$	-0.0767	-0.0439	0.0077	0.0078	0.0318	-0.0035
2	$\hat{\mu}_E$	0.1167	0.0000	0.0633	0.4667	0.4700	0.0000
	$\mu(\tilde{\pi})$	0.1332	0.0000	0.1045	0.3196	0.5759	0.0000
	$\tilde{w}$	0.2340	-0.1098	0.0092	0.0487	0.0576	-0.0056
3	$\hat{\mu}_E$	0.0000	0.0000	0.0000	0.0033	0.7058	0.2908
	$\mu(\tilde{\pi})$	0.0000	0.0000	0.0000	0.0000	0.7447	0.2554
	$\tilde{w}$	-0.1056	-0.0051	-0.0573	-0.0386	0.0929	0.0081
4	$\hat{\mu}_E$	0.0600	0.0000	0.0000	0.0033	0.2908	0.7058
	$\mu(\tilde{\pi})$	0.0569	0.0000	0.0000	0.0000	0.2666	0.7334
	$\tilde{w}$	0.1079	-0.0001	-0.0487	-0.0666	0.0590	0.0564
5	$\hat{\mu}_E$	0.0600	0.0000	0.0000	1.0000	0.0000	0.0000
	$\mu(\tilde{\pi})$	0.0542	0.0000	0.0000	1.0000	0.0000	0.0000
	$\tilde{w}$	0.0094	-0.0108	-0.2765	0.8126	-0.5099	-0.0154

Discussion of Results

1-2 slides

What conclusions are drawn from the results by the authors?

- What insights are gained from the experiments?
- What strengths and weaknesses of the proposed method are illustrated by the results?

Are the stated conclusions fully backed by the results and references?

- If so, why? (Recap the relevant supporting evidences from the given results + refs)
- If not, what are the additional experiments / comparisons that can further support/repudiate the conclusions of the paper?

Conclusion

Discussion

- In two experimental domains, with a reasonably small set of expert demonstrations, the method learns a reward function which can be used to quantitatively and qualitatively match expert performance.
- Validates that reward function weights make sense.
- Provides theoretical bounds for the number of iterations and demonstrations needed to attain a given performance differential from the expert.

Critique / Limitations / Open Issues

1-2 slides

What are the key limitations of the proposed approach / ideas? (e.g. does it require strong assumptions that are unlikely to be practical? Computationally expensive? Require a lot of data?)

Are there any practical challenges in deploying the approach on physical robots in the real world? Are there any safety or ethical concerns of using such approach?

If follow-up work has addressed some of these limitations, include pointers to that. But don't limit your discussion only to the problems / limitations that have already been addressed.

Discussion

- Limiting model assumptions (may impact applicability to other domains)
  - The reward function is a linear function of known features.
  - These features are hand-crafted by human experts to effectively parameterize the domain.
  - Only validated in domains with discrete actions and states.
  - States are fully observed.
- The technique assumes access to expert demonstrations, which may be difficult to provide for certain tasks.

Future Work for Paper / Reading

1-2 slides

What interesting questions does it raise for future work?

- Your own ideas for future work
- Others' ideas (if others have already built on this idea)

Extended Readings

1-2 slides

Pointers to papers that use this paper as a reference and/or other very related papers that others may want to read

Point classmates to where they can go for further reading on this paper/reading

Further Reading

- In the last 21 (!) years a lot of papers have built on the original formulation of IRL (Ng and Russell, 2000) and Apprenticeship Learning via Inverse Reinforcement Learning (Abbeel and Ng, 2004).
  - An Application of Reinforcement Learning to Aerobatic Helicopter Flight (Abbeel et al., 2006).
    - https://youtu.be/M-QUkgk3HyE?t=104
  - Bayesian Inverse Reinforcement Learning (Ramachandran & Amir, 2007).
    - Bayesian formulation of IRL and apprenticeship learning.
  - Active Preference-Based Learning of Reward Functions (Sadigh et al, 2017).
    - Demonstrations may be hard to provide; replaces the need for explicit demonstrations with “preferences” over automatically generated (active-learning) demonstrations.

Summary

1 slide

Approximately one bullet for each of the following

- Problem the reading is discussing
- Why is it important and hard
- What is the key limitation of prior work
- What is the key insight(s) (try to do in 1-3) of the proposed work
- What did they demonstrate by this insight? (tighter theoretical bounds, state of the art performance on X, etc)

Summary

- Learns a reward function from expert demonstrations, and then a policy from that reward function.
- Provides a foundational formulation and implementation of the proposed learning problem.
- Important because reward functions form a compact description of correct behavior for solving a task, but are often hard to specify directly for complex tasks.
- Demonstrates tractable learning of reward functions in few iterations, while attaining high task performance.
- Provides theoretical bounds based for the number of iterations and demonstrations.