Auto-Encoding Inverse Reinforcement Learning

Nicolas Zuo, Youssef Kadhi, Hector Kohler

Github: https://github.com/KohlerHECTOR/aeirl-pytorch
Article: https://openreview.net/pdf?id=OCgCYv7KGZe

Abstract

- Article addresses the problem of learning representations from noisy expert demonstrations in Adversarial Imitation Learning.
- Proposes an Adversarial Inverse Reinforcement Learning architecture built on top of GAIL. In this work, the discriminator is replaced with an auto-encoder.
- The empirical analysis show that the learned reward function can be more informative and robust to noisy data.

Auto-Encoding Inverse Reinforcement Learning

Inverse Reinforcement Learning

- Given an expert policy $\pi_{\mathbb{E}}$, IRL fits a reward function from a family of functions $\mathcal{R}$ with the optimization problem:

$$\min_{r\in\mathcal{R}} \left(\max_{\pi\in\Pi} \mathbb{E}_{\pi}[r(s,a)] \right) - \mathbb{E}_{\pi_E}[r(s,a)]$$

- IRL looks for a reward function $r(s,a)$ that assigns high values to the expert policy and low values to other policies. Then classical Reinforcement Learning is performed to learn a policy using the learned reward function:

$$\mathrm{RL}(r) = \arg\max_{\pi\in\Pi} \mathbb{E}_{\pi}[r(s,a)]$$

Adversarial Imitation Learning

- Adversarial Imitation Learning such as Generative Adversarial Imitation Learning (GAIL) formulates the learned reward function as a discriminator that learns to differentiate expert transitions from non-expert ones.
- This leads to an Adversarial IRL formulation:

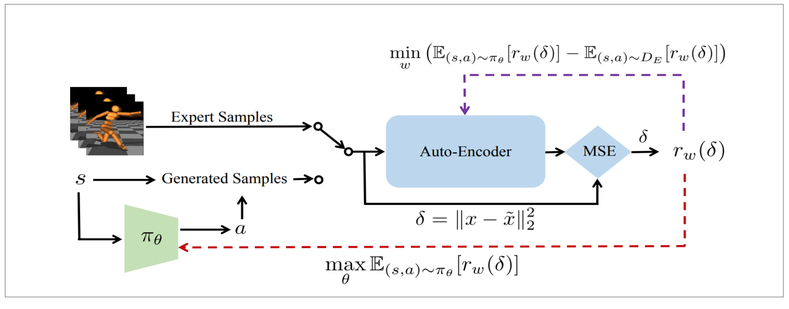
$$\min_{\pi_\theta} \max_{r_w} \mathbb{E}_{(s,a)\sim D_E}[r_w(s,a)] - \mathbb{E}_{(s,a)\sim \pi_\theta}[r_w(s,a)]$$

- D_E are expert trajectories. In the case of GAIL, $r_w(s,a) = \log D_w(s,a)$, with $D_w(s,a)$ the output of a discriminator network.

Auto-Encoding Inverse Reinforcement Learning

- AEIRL models the reward by a function of the reconstruction error of an auto-encoder network. This gives the same Adversarial IRL formulation as for GAIL with:

$$r_w(s,a) = 1/\left(1 + \mathrm{AE}_w(s,a)\right) \text{ , with}$$
$$\mathrm{AE}(x) = \| \mathrm{Dec} \circ \mathrm{Enc}(x) - x \|_2^2, \text{ the reconstruction error}$$



Practical algorithm

Algorithm 1: Auto-Encoding Inverse Reinforcement Learning (AEIRL)

Initial parameters of policy, auto-encoder θ_0, w_0; Expert trajectories D_E
for $i = 0$ *to* N **do**
  Sample state-action pairs $(s_i, a_i) \sim \pi_{\theta_i}$ and $(s_E, a_E) \sim \mathrm{D}_E$
  Update w_i to w_{i+1} by decreasing the gradient:

$$\mathbb{E}_{(s_i,a_i)}\left[\nabla_{w_i} 1/\left(1 + \mathrm{AE}_{w_i}(s_i,a_i)\right)\right] - \mathbb{E}_{(s_E,a_E)}\left[\nabla_{w_i} 1/\left(1 + \mathrm{AE}_{w_i}(s,a)\right)\right]$$

  Take a policy step from θ_i to θ_{i+1}, using the TRPO update rule with the reward function $1/(1 + \mathrm{AE}_{w_i}(s,a)))$, and the objective function for TRPO is:

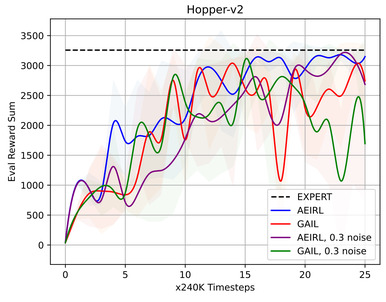
$$\mathbb{E}_{(s,a)}\left[-1/\left(1 + \mathrm{AE}_{w_i}(s,a)\right)\right]$$

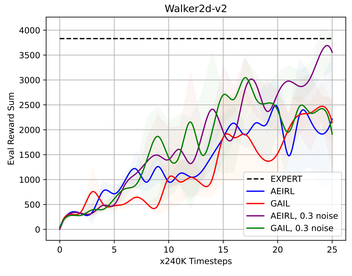
The practical algorithm uses Truncated Region Policy Optimization for the classical Reinforcement Learning step RL(r). To get GAIL from AEIRL, change the w_i to w_{i+1} update to : $\mathbb{E}_{(s_i,a_i)}\left[\nabla_{w_i} \log\left(D_{w_i}(s,a)\right)\right] + \mathbb{E}_{(s_E,a_E)}\left[\nabla_{w_i} \log\left(1 - D_{w_i}(s,a)\right)\right]$. Change the reward function in TRPO to : $\log(D_{w_i}(s,a))$. And the TRPO objective to : $\mathbb{E}_{(s,a)}[-\log(D_{w_i}(s,a))]$

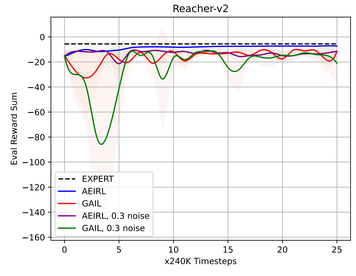
Experimental setup

- Experts are PPO agents from Stable Baselines 3 zoo.
- Benchmarking on mujoco tasks: Hopper-v2, Walker2d-v2, Reacher-v2 (Reacher-v2 was not used in the article).
- For each task: 5 runs of 6×10^6 steps for both GAIL and AEIRL. Learned policy evaluated every 240×10^3 steps on 10 rollouts (same as in article with slightly less steps).
- TRPO hyper-parameters are default (from original TRPO article).

Performance analysis







Policy	Walker2d	Hopper	Reacher
GAIL $\neg noisy$	2286.67 ± 953.97	2663.00 ± 795.05	-11.37 ± 4.23
GAIL noisy	1136.44 ± 470.18	556.80 ± 5.54	-10.09 ± 2.35
AEIRL $\neg noisy$	**3408.57** $\pm$ 502.86	**3196.67** $\pm$ 4.61	**-5.84** $\pm$ 1.91
AEIRL noisy	3406.07 ± 1110.57	3179.67 ± 9.90	-11.58 ± 2.93
Performance loss	Walker2d	Hopper	Reacher
GAIL	**50.3%**	**79.09%**	-11.32%
AEIRL	0.07%	9.90%	**98.29%**

Figure 2: Summary table of performances for learned agents with GAIL and AEIRL on non-noisy or noisy expert data

Learned reward functions analysis

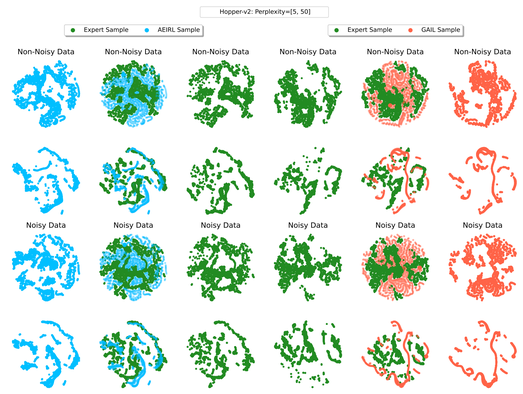
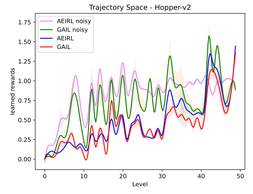


Figure 3: t-SNE visualisation of learned reward functions latent spaces



- To compare the robustness of different reward signals of different methods, authors use t-SNE to visualize the latent space of the reward function network.
- Data overlapping transcribes a good imitation process.
- Independently of perplexity and noise, AEIRL data seems to overlap with the Expert data more than GAIL data do.
- In a noisy context, GAIL data and Expert data almost never overlap unlike AEIRL data.
- Authors claim the auto-encoder should provide more information to the agent than a discriminator based reward function, to demonstrate, we evaluate agent of different levels of performances using both learned rewards.
- The ideal would be to have smooth increasing curves showing that the learned reward functions discriminate correctly between the different levels of performances.
- We don't have as much smoothness nor as much increase as in the article whether it is for the auto-encoder based reward or for the discriminator based reward.

Conclusions

- In terms of solving the IRL problem, AEIRL outperformed GAIL on all benchmarks.
- In terms of robustness to noise AEIRL outperformed GAIL on 2 out of 3 benchmarks.
- However it is not that clear that the reward signal from the auto-encoder is more informative. Refs: TRPO 2015, GAIL 2016 , AEIRL 2021