Connections Between Inverse Reinforcement Learning, Imitation Learning, and Apprenticeship Learning

With Algorithms

Jonathan Wilder Lavington

December 20, 2018

University of British Columbia

Table of contents

Introduction

Quick note before we start

Everything I will talk about is discussed in [3, 4, 2, 1].

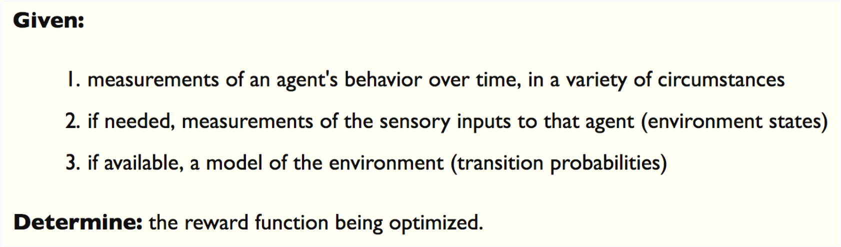
The last citation is especially illuminating if your interested in this more, check out: “A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models”.

I ended up making these slides without realizing it existed, so I’m sure it would supplement this topic nicely!

What is Inverse Reinforcement Learning (IRL)?

Inverse Reinforcement Learning (IRL) is defined as the problem of attempting to determine the reward function being optimized by an agent.

More explicitly,



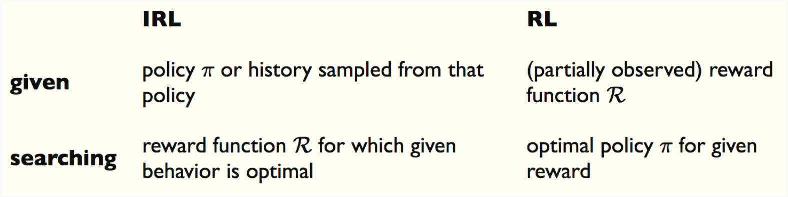
Given:

1. measurements of an agent's behavior over time, in a variety of circumstances
2. if needed, measurements of the sensory inputs to that agent (environment states)
3. if available, a model of the environment (transition probabilities)

Determine: the reward function being optimized.

What is the connection to RL?

We can tie this back reinforcement learning,



	IRL	**RL**
given	policy π or history sampled from that policy	(partially observed) reward function $\mathcal{R}$
searching	reward function $\mathcal{R}$ for which given behavior is optimal	optimal policy π for given reward

Why is IRL interesting?

Why would we want to search for a reward function?

In most RL problems there is not a natural source for the reward signal, and researchers using the method must painstakingly engineer there own to represent the task.

A better route to fitting a reward function might be to watch an expert complete the trajectory, and directly infer the reward from observations.

IRL Is Not Behavioral Cloning.

When given trajectories sampled from a policy, we could attempt to infer the policy itself, and thus ”clone” the behavior of the agent.

This is not robust to changes in the environments dynamics, and is thus a bad idea for attempting to create agents that are generalize to other environments.

In many ways, the reward function is a transferable description of the task.

What are the Applications to RL

The biggest argument for IRL is it allows for an automated method to specify a reward function for a task.

This could mean inferring a reward strictly based upon expert trajectories, or using IRL on spaces that exhibit sparse reward structures to improve training.

In many cases it might be useful to infer a non sparse reward from an expert, train a robust policy on the approximate reward, and then apply an RL algorithm over the actual observed rewards given the better approximation of an expert policy.

IRL as a Linear Programming Problem

Basic MDP Setup

One basic way to solve the IRL problem is to generate and solve a series of linear programming tasks as proposed by (Ng 2000). In IRL, much like RL we define our MDP using the following tuple,

$$(S,A,T,r,\gamma,\rho_0)$$

Where:

- S is the state space
- A is the action space
- γ is the discount factor
- T(s'|s,a) is the transition dynamics
- ρ_0(s) is the initial state distribution
- R(s,a) is the reward function

Basic MDP Setup

In most cases, we will not be provided with the MDP, but instead will be provided with a trajectory, or set of trajectories defined as:

$$\zeta_i = (a_0, s_0, \ \dots \ , a_T, s_T) \text{ for i = 1 … N}$$

Additionally we can define the value of a state under a policy π as:

$$V_\pi(s) = R(s) + \gamma \sum_{s' \in S} T(s'|s, \pi(s)) V_\pi(s')$$

We can then define the value of state action pairs as,

$$Q_\pi(s, a) = R(s) + \gamma \sum_{s' \in S} T(s'|s, a) V_\pi(s')$$

Solving IRL

Problem Statement:

Assume we are in an infinite state-space MDP representing some non-discrete process.

Additionally assume that we are given a set of expert trajectories $\zeta_{\pi_i^*}$

How can we approximate the reward function?

No Neural Networks Yet.

For simpler problems in continuous spaces, we can just model the reward with linear combinations of known basis functions.

For a d dimensional functional representation,

$$\hat{R}(s) = \alpha_1\phi_1(s) + \alpha_2\phi_2(s) + \ \dots \ + \alpha_d\phi_d(s)$$

Now, we want to approximate the empirical value of a trajectory under the current reward function, not the expected value of a policy.

Returning to the Value Function V^π

Because we assumed that our approximation $\hat{R}(s)$ was a linear combination of basis functions,

$$\hat{V}_i(\zeta) = \gamma\phi_i(s_1) + \gamma^2\phi_i(s_2) + \ \dots\ + \gamma^k\phi_i(s_k)$$

and thus we can write the empirical value of a trajectory under our reward function as:

$$\hat{V}(\zeta) = \alpha_1\hat{V}_1(\zeta) + \alpha_2\hat{V}_2(\zeta) + \ \dots\ + \alpha_d\hat{V}_d(\zeta)$$

Where we want to ensure that for some expert trajectories π_i^*, and some generated set of non-expert trajectories π_j,

$$\hat{V}(\zeta_{\pi^*,i}) \geq \hat{V}(\zeta_{\pi,j})$$

How do we get the trajectories π_j?

We could either generate random policies under the reward, or solve an RL problem to infer a policy.

There is an issue since we are still trying to infer rewards the yield higher values for the expert trajectories then the trained strategies.

If we run a RL algorithm to find a policy which is optimal for the current reward estimate, its likely that we will find a policy that does better then the expert.

The Final Problem Statement

Given a set of expert trajectories ζ_{π^*}, and a basis ϕ

$$\begin{aligned} \underset{\alpha}{\text{argmax}} \quad & \sum_{i=1}^{n}\left[\beta \sum_{j=1}^{m}(\hat{V}(\zeta_{\pi^*,i}) - \hat{V}(\zeta_{\pi,j}))\right] \\ \text{subject to} \quad & |\alpha_i| < 1,\ i = 1, \dots, d. \\ & \beta > 0 \end{aligned}$$

Where,

$$\begin{aligned} \hat{R}(s) &= \alpha_1\phi_1(s) + \alpha_2\phi_2(s) + \ \dots\ + \alpha_d\phi_d(s) \\ \hat{V}_i(\zeta) &= \gamma\phi_i(s_1) + \gamma^2\phi_i(s_2) + \ \dots\ + \gamma^k\phi_i(s_k) \\ \hat{V}(\zeta) &= \alpha_1\hat{V}_1(\zeta) + \alpha_2\hat{V}_2(\zeta) + \ \dots\ + \alpha_d\hat{V}_d(\zeta) \end{aligned}$$

and

$$\pi_i \sim \underset{\pi}{\text{maximize}} \quad \sum_{t=1}^{T} E_{a_t,s_t\sim\pi(a_t,s_t)}\left[\hat{R}(s_t) - log(\pi(a_t,s_t)))\right] \qquad (1)$$

Pseudo-code for the Algorithm

At a high level, we generate an optimal policy based upon the current approximation of R, and then update R so as to maximize the expression above. In pseudo-code,

Algorithm 1 Approximation of reward function via linear programming in IRL

```
1: generate random trajectory $\zeta_{\pi_0}$ from randomly generated policy $\pi_0$
2: for $iter = 0$ to $N$ do
3:     Solve the linear programming problem for $\alpha$ given expert trajectories $\{\zeta_{\pi_i^*}\}_1^n$, and generated policies $\{\zeta_{\pi_i}\}_0^{iter}$.
4:     Solve the forward reinforcement learning problem given the new reward function $R_\alpha(s)$ and store new policy $\zeta_{\pi_i ter}$
5: end for
```

Advantages and Issues

Advantages:

- Because we are finding the optimal policy at each step, provided that the reward update is not drastic, the previous policy update should act as a good guess for the next one.
- This procedure could be applied to solving an MDP with expert trajectories, and significant uncertainty about the state dependent rewards.

Disadvantages:

- We have to approximate the solution to the forward RL problem at each step, and its not clear how much this assumption can or should be relaxed.
- The trained optimal policy will tend to imitate the experts policy.

Intermission

Ideally, if we can create the true underlying reward structure of the environment then non-generalizable expert imitation should not happen.

How can we use IRL to create a robust model of the environment through approximation of the reward?

Maximum Causal Entropy IRL

IRL as Maximizing Causal Entropy

In maximum Casual entropy IRL we map a reward function to high-entropy policies that minimize the expected cumulative reward.

As a result, we can reformulate our problem as,

$$\underset{r \in R}{\operatorname{argmax}} \quad \left(\min_{\pi \in \Pi} - H(\pi) + E_{\pi} r(s,a) \right) - E_{\pi^*} r(s,a)$$

Where:

$$H(\pi) = -E_{\pi} \log(\pi(a|s))$$

$$r(s,a) = \left[\sum_{t=0}^{\infty} \gamma^t r(a_t, s_t) \right]$$

Basically the only change here is that the forward RL problem is now changed to that of Max entropy RL problem.

Regularized Maximizing Causal Entropy IRL

Under maximum Casual entropy IRL, we can additionally add a regularization term ψ:

$$\operatorname*{argmax}_{r \in R} \quad -\psi(r) + \left(\min_{\pi \in \Pi} - H(\pi) + E_{\pi} r(s,a) \right) - E_{\pi^*} r(s,a)$$

Where the only requirement on ψ is that it is convex in r. This regularization just ensures that functional representation of r doesn't become to complex, and ideally improves generalization of the reward function.

Another use for the Relationship between RL and IRL

To see the interesting relationship between RL and MCE-IRL described by (Ho, Ermon 2016), we have to define a need to define an extra term.

Occupancy measure: The unnormalized distribution of state-action pairs that an agent encounters when navigating the environment with the policy

$$\rho_\pi(s,a) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi)$$

Which allows us to write the expectation from the last slide:

$$E_\pi(r(s,a)) = \sum_{s,a} \rho_\pi(s,a) r(s,a)$$

Imitation Learning

Now we can flip the problem on its head to derive one interpretation of imitation learning.

$$RL \circ IRL_{\psi}(\pi^*) = \underset{\pi \in \Pi}{\operatorname{argmin}} - H(\pi) + \psi^*(\rho_\pi - \rho_{\pi^*})$$

Where ψ^* is the convex conjugate of ψ.

The equation above describes the vanilla maximum entropy RL objective regularized by some function ψ over a set of observed expert trajectories.

Imitation Learning

If we do a little work, we see that **IRL is in fact the dual problem to imitation learning's primal problem**.

We can note the for constant ψ there will be not reward regularization at all, then the recovered optimal policy will exactly match the expert's occupancy measure.

For now we can quickly look at how this occupancy matching approach actually works in practice.

Apprenticeship Learning

For specific regularizations ψ and sets of reward function classes R,

$$\hat{\pi} = \underset{\pi \in \Pi}{\text{argmin}} - H(\pi) + \psi^*(\rho_\pi - \rho_{\pi^*})$$

we arrive at different forms of “apprenticeship learning”, which attempts to arrive at a policy that is better then the experts, and optimizes:

$$\max_{\pi \in \Pi} \quad \min_{r \in R} E_\pi(r(s, a)) - E_{\pi^*}(r(s, a))$$

Similar the first algorithm we talked about most algorithms use convex combinations of linear basis.

Apprenticeship Learning

Two such functions are

$$\begin{aligned} R_{linear} &\in \{\sum w_j f_j,\ ||w||_2^2 \leq 1\} \\ R_{convex} &\in \{\sum w_i f_i,\ \sum w_j = 1, w_j \geq 1 \text{ for j = 1 ... n}\} \end{aligned}$$

The first leads to feature expectation matching, which minimizes $_2$ distance between expected feature vectors, while the second minimizes the worst-case excess reward among the individual basis functions:

$$\begin{aligned} \min_{r \in R_{linear}} E_\pi f(s,a) - E_{\pi^*} f(s,a) &= ||E_\pi f(s,a) - E_{\pi^*} f(s,a)||_2 \\ \min_{r \in R_{convex}} E_\pi f(s,a) - E_{\pi^*} f(s,a) &= max(E_\pi f_i(s,a) - E_{\pi^*} f_i(s,a)) \end{aligned}$$

Apprenticeship Learning

Interestingly enough, we can also show that for a specific ψ correspond to the original imitation learning scheme! Looking at the objective for Apprenticeship Learning,

$$\max_{\pi \in \Pi} \quad \min_{r \in R} \quad E_{\pi}(r(s,a)) - E_{\pi^*}(r(s,a))$$

If we let the regularizer ψ be the indicator function $\delta_C(c)$ such that $\delta_C(c) = 0$ for $c \in C$ and $\delta_C(c) = +\infty$ for $c \notin C$, then we can recover the original objective above.

If we relax the above, and let $\delta_C(c) = \alpha$ for $c \notin C$ and $\alpha > 0$, then we recover something interesting.

Apprenticeship Learning

If we look at the inner optimization function regularized by δ_c,

$$\begin{aligned} & \min_{r \in R} && E_\pi(r(s,a)) - E_{\pi^*}(r(s,a)) \\ = & \min_{r \in R} && -\delta(c) + \sum (\rho_\pi(s,a) - \rho_{\pi^*}(s,a)) r(s,a) \\ = & \min_{r \in R} && \delta(c)(\rho_\pi - \rho_{\pi^*}) \end{aligned}$$

Which means if we incorporate entropy into our objective such that:

$$\begin{aligned} & \max_{\pi \in \Pi} \; -H(\pi) + \min_{r \in R} && E_\pi(r(s,a)) - E_{\pi^*}(r(s,a)) \\ = & \max_{\pi \in \Pi} \; -H(\pi) + \min_{r \in R} && \delta(c)(\rho_\pi - \rho_{\pi^*}) \end{aligned}$$

We now have that entropy regularized Apprenticeship Learning is equivalent to performing RL following IRL with cost regularizer δ_c.

Scalable Apprenticeship Learning in Practice

Algorithm 2 Apprenticeship Learning with TRPO

```
1: generate random policy $\pi_{\theta_0}$
2: for $iter = 0$ to $N$ do
3:     Sample trajectories of the current policy $\pi_{\theta_i}$ by simulating in the
       environment, and then fit a reward function $r_i^*$ using:
           $$r^* = \underset{r \in R}{\text{argmax}} \quad E_\pi(r(s,a)) - E_{\pi^*}(r(s,a))$$
4:     Form a gradient estimate using:
           $$\nabla_\theta E_{\pi_\theta}(r(s,a)) = \quad E_{\pi_\theta}(\nabla_\theta \log \pi(a|s) E_{\pi_\theta}(r^*(a,s)|a_0,s_0))$$
5:     Using reward approximation and the sampled trajectories, take
       a trust region policy optimization (TRPO) step to produce $\pi_{\theta_{i+1}}$.
6: end for
```

Intermission

We have shown the relationship between max causal entropy IRL, IL, and AL.

How can we use this relationship to make more robust IRL/IL/AL algorithms?

For the last stretch we will look at two recent algorithms called GAIL, and AIRL that attempt to answer this.

Generative Adversarial Imitation Learning (GAIL)

Motivation

Constant regularizers leads to an imitation learning algorithm that exactly matches occupancy measures, but is intractable in large environments.

The indicator regularizers for the linear cost functions, lead to algorithms incapable of exactly matching occupancy measures without careful tuning, but are tractable in large environments.

Using this information we can create a better regularizer defined by,

$$\psi_{\mathrm{GA}}(c) \triangleq \begin{cases} \mathbb{E}_{\pi_E}[g(c(s,a))] & \text{if } c < 0 \\ +\infty & \text{otherwise} \end{cases} \quad \text{where } g(x) = \begin{cases} -x - \log(1 - e^x) & \text{if } x < 0 \\ +\infty & \text{otherwise} \end{cases}$$

Finally a Connection to GANs

It turns out that if we apply this regularization to the difference between between occupancy measures,

$$\psi^*_{\mathrm{GA}}(\rho_\pi - \rho_{\pi_E}) = \max_{D \in (0,1)^{\mathcal{S}\times\mathcal{A}}} \mathbb{E}_\pi[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))]$$

Which is the optimal negative log loss of the binary classification problem (equivalent by JS divergence) of distinguishing between state-action pairs of π and π^*.

GAIL Algorithm

This means that if we add causal entropy H as a policy regularizer, we obtain a new imitation learning algorithm, given by the objective:

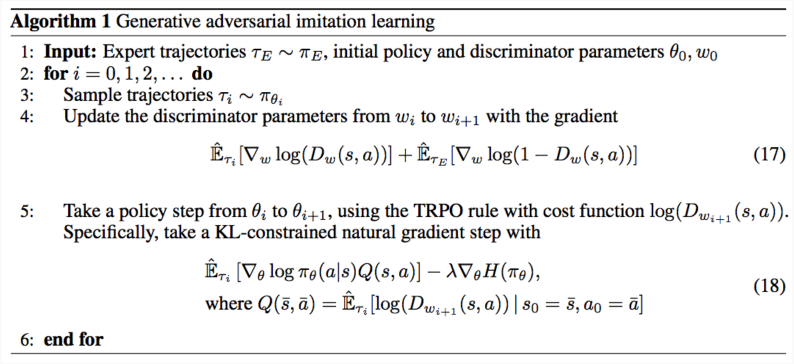
$$\underset{\pi}{\text{minimize}}\ \psi^*_{\text{GA}}(\rho_\pi - \rho_{\pi_E}) - \lambda H(\pi) = D_{\text{JS}}(\rho_\pi, \rho_{\pi_E}) - \lambda H(\pi)$$

This algorithm finds a policy whose occupancy measure minimizes Jensen-Shannon divergence to the expert’s.

Additionally, it minimizes a true metric between occupancy measures, so, unlike linear apprenticeship learning algorithms, it can imitate expert policies exactly.

GAIL Algorithm

Now that we have the objective defined, we can apply a GAN framework via the GAIL algorithm.



Algorithm 1 Generative adversarial imitation learning

```
1: Input: Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters $\theta_0, w_0$
2: for $i = 0, 1, 2, \ldots$ do
3:     Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4:     Update the discriminator parameters from $w_i$ to $w_{i+1}$ with the gradient
           $\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))]$    (17)
5:     Take a policy step from $\theta_i$ to $\theta_{i+1}$, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$.
       Specifically, take a KL-constrained natural gradient step with
           $\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta),$
           where $Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}]$    (18)
6: end for
```

In this case, The discriminator network can be interpreted as a local cost function providing learning signal to the policy, while the generator learns the policy itself.

GAIL Algorithm (Pros and Cons)

This approach is nice because,

- We have rephrased the imitation learning problem as an adversarial one, which allows us to represent both the estimated cost function, as well as the policy using complex functional representations like neural networks.

There are some issues however:

- GANs are notoriously difficult to train, especially under JS divergence where mode collapse is often an issue.
- In this case mode collapse would likely mean exact imitation of expert trajectories instead of a good approximation of expert policy.
- We also don’t explicitly learn a reward function, which means generalization will be even harder.

Intermission

We saw above that we can reformulate IL (and as a result of the section before that both AL and IRL) in a generative adversarial framework.

However this framework still has some issue: namely we dont recover a reward function,

Additionally we are also still working with full trajectories.

Ideally we could make gradient updates on a more local basis (i.e. descriminate based upon state action pairs).

Adversarial Inverse Reinforcement Learning (AIRL)

Adding a few Changes to GAIL

GAIL does not place special structure on the discriminator, so the reward cannot be recovered, however if we reformulate the discriminator as proposed by Finn et al. (2016), we can fix this!

First Fix: GAN-GCL

We can define a GAN where the discriminator is defined by,

$$D_\theta(\tau) = \frac{exp\{f_\theta(\tau)\}}{exp\{f_\theta(\tau)\} + \pi(\tau)}$$

Where f is a learned function parameters θ.

The policy π is then trained to optimize,

$$R(\tau) = log(1 - D(\tau)) - log(D(\tau))$$

In this case updating the discriminator represents updating the reward function, while updating π represents improving the sampling distribution over the partition function.

Quick note

The reason that the updating the discriminator determines the approximation of the reward follows from the fact that at optimality it can be shown:

$$f^*(\tau) = R^*(\tau) + \text{const}$$

In this case the partition function is defined as, exponential of some function f.

Full Trajectory Issues

What is the issue with this methodology?

It relies on approximating a reward based off of full trajectories instead of single state action pairs.

This leads to high variance approximations of the reward functions, and in practice makes it less generalize.

How can we fix this?

AIRL = State-Action Based GAN-GCL

We can convert our previous equations to a single state-action version,

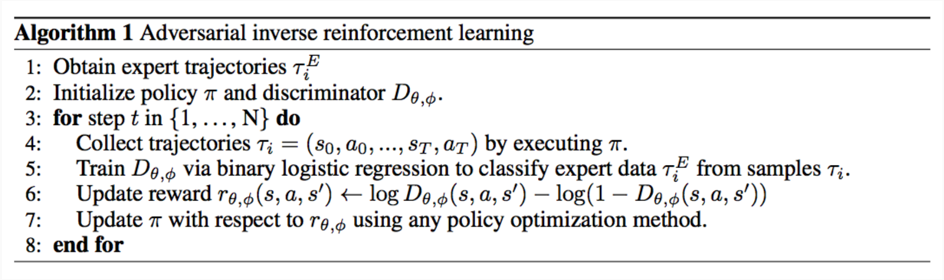
$$D_\theta(s, a) = \frac{exp\{f_\theta(s, a)\}}{exp\{f_\theta(s, a)\} + \pi(s|a)}$$

and the policy π is then trained to optimize,

$$R(s, a) = log(1 - D(s, a)) - log(D(s, a))$$

In this case it can be shown that at optimally $f^*(s, a) = A^*(s, a)$, where $A^*(s, a)$ is the advantage function of the optimal policy.

AIRL Algorithm



Algorithm 1 Adversarial inverse reinforcement learning

```
1: Obtain expert trajectories $\tau_i^E$
2: Initialize policy $\pi$ and discriminator $D_{\theta,\phi}$.
3: for step $t$ in $\{1, \ldots, \mathrm{N}\}$ do
4:     Collect trajectories $\tau_i = (s_0, a_0, ..., s_T, a_T)$ by executing $\pi$.
5:     Train $D_{\theta,\phi}$ via binary logistic regression to classify expert data $\tau_i^E$ from samples $\tau_i$.
6:     Update reward $r_{\theta,\phi}(s, a, s') \leftarrow \log D_{\theta,\phi}(s, a, s') - \log(1 - D_{\theta,\phi}(s, a, s'))$
7:     Update $\pi$ with respect to $r_{\theta,\phi}$ using any policy optimization method.
8: end for
```

Avoiding Behavioral Cloning

In most cases the reward still encourages mimicking the expert policy π^*, and fails to produce desirable results when changes to the environment are made.

How can can we recover more robust rewards?

Answer: By generating a “Disentangled” reward function.

Disentangling Rewards from Dynamics

What is a disentangled reward?

If we define $Q^*_{r,T}$ as the optimal Q function with respect to a reward function r and dynamics T, and $\pi^*_{r,T}$ in the same way, then we can define disentangled rewards in the following way:

Definition: (Disentangled Rewards) A reward function $r'(s, a, s')$ is disentangled with respect to the ground truth reward $r(s, a, s')$ and dynamics T such that under all dynamics $T' \in T$, the optimal policy is the same $(\pi^*_{r',T}(a|s) = \pi^*_{r,T}(a|s))$.

Disentangling Rewards from Dynamics

When can we find “Disentangled Rewards”?

As it turns out, when the true underlying reward is only a function of the current state.

That is to say if the ground truth reward is a function of the current state, we can find a disentangled reward that is also a function of state such that we have recovered the ground truth reward up to a constant.

Disentangling Rewards from Dynamics

With a little work we can show that this is true, provided we change the form of our discriminator slightly:

$$D_{\theta,\phi}(s,a,s') = \frac{\exp\{f_{\theta,\phi}(s,a,s')\}}{\exp\{f_{\theta,\phi}(s,a,s')\} + \pi(a|s)},$$

Where

$$f_{\theta,\phi}(s,a,s') = g_\theta(s,a) + \gamma h_\phi(s') - h_\phi(s).$$

Where g and h is determined based upon decomposability conditions that we don't have time to go over!

A Few Final Notes

It turns out that under the decomposability conditions that I didn't describe above, we find the following:

- Under deterministic environments with a state-only ground truth reward:

$$g^*(s) = r^*(s) + \text{const},$$
$$h^*(s) = V^*(s) + \text{const},$$

- One view $f_{\theta,\phi}$ as the advantage under deterministic dynamics:

$$f^*(s,a,s') = \underbrace{r^*(s) + \gamma V^*(s')}_{Q(s,a)} - \underbrace{V^*(s)}_{V(s)} = A^*(s,a)$$

- Under stochastic dynamics, $f_{\theta,\phi}$ can be seen as a single sample approximate of the advantage function $A^*(s,a)$

Questions?

References i

C. Finn, P. Christiano, P. Abbeel, and S. Levine.
A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models.
arXiv preprint arXiv:1611.03852, 2016.

C. Finn, T. Yu, J. Fu, P. Abbeel, and S. Levine.
Generalizing skills with semi-supervised reinforcement learning.
arXiv preprint arXiv:1612.00429, 2016.

J. Fu, K. Luo, and S. Levine.
Learning robust rewards with adversarial inverse reinforcement learning.
arXiv preprint arXiv:1710.11248, 2017.

References ii

J. Ho and S. Ermon.
Generative adversarial imitation learning.
In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.