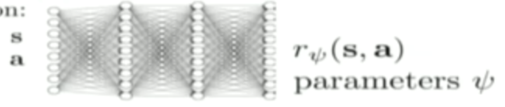
inverse RL

- what if the reward function is not obvious? but we get to observe an expert. can we learn the reward function and then run forward RL?
- look at the following situations where it is easy to see when a task has been successfully completed or not but it is hard to come up with a reward function that does that.

- lets assume here that we have sufficient expert demonstrations in all of the above cases.
- we did see imitation learning earlier and learning from experts but the important difference there is that we did not try to recover the objective of the expert, we just learnt a supervised model, which doesn't necessarily capture the salient parts of the expert's behavior.
- the baseline here is both RL with a hand designed reward and imitation learning (behavior cloning). We want to show that with inverse RL we can better than both hand designed reward RL and imitation learning at least on some tasks.
- inverse RL (IRL) is also called inverse optimal control by the way.
- ok, so the problem statement is given some observations, find the reward function.
  - This is actually an underdefined problem because for a given set of sample data of observations, there are a million rewards that are being maximized without any prior data. if you are human, because of *prior experience/knowledge* you might know what they are going for but ML algorithms don't have this prior knowledge.
  - the other challenge is that the demonstrations may not be precisely optimal for the intended reward (because of what we saw in the notes on control vs inference).
- we will learn a reward function that maps states and actions to rewards. we will parametrize our functions with ψ. (we used θ for policy params and ϕ for q/value function params)
- this could be a simple linear/quadratic function or a complex neural net for deep IRL.

linear reward function:
$r_\psi(\mathbf{s},\mathbf{a}) = \sum_i \psi_i f_i(\mathbf{s},\mathbf{a}) = \psi^T \mathbf{f}(\mathbf{s},\mathbf{a})$

neural net reward function:



- In the linear case above, there are a bunch of feature functions that describe "i care about this vs this vs this" and we learn the relative importance of each by learning weights for those feature functions. Each state and action gets put thru the sum of these feature functions.
- There is a set of classical SVM based IRL approaches that address the problem of which of the many rewards that fit this action should we pick" problem by framing it as a maximum margin problem. But that formulation is kind of arbitrary and does not account for suboptimality of expert. More importantly, it leads to an optimization problem that's messy without giving us the expressivity and ability to model complex rewards like we can using deep learning.
- Recall the probabilistic graphical model we used in the notes on control vs inference → we will use that as a model of how the expert is behaving and fit the observed expert data to that model.
- The high level idea is this: we will assume that the reward function is some function of a set of parameters ϕ, r_ϕ. Now recall the formula for $p(\tau|O_{1:T})$, the probability of a trajectory given that its soft optimality. We want to do maximize the joint likelihood of the observed expert trajectories given this model. In order to do the maximization, we will take the gradient of the joint likelihood (by doing sum of log likelihood) w.r.t ϕ and use a gradient based optimization strategy like SGD. The math is in the following two slides.
- The first mostly comes from the notes on control vs inference and is the problem formulation. The second slide gives a trick to estimate the gradient w.r.t log Z, the probability normalizer in the denominator. This log Z is called the IRL partition function.

$p(\mathcal{O}_t|\mathbf{s}_t,\mathbf{a}_t,\psi) \propto \exp(r_\psi(\mathbf{s}_t,\mathbf{a}_t))$

given:

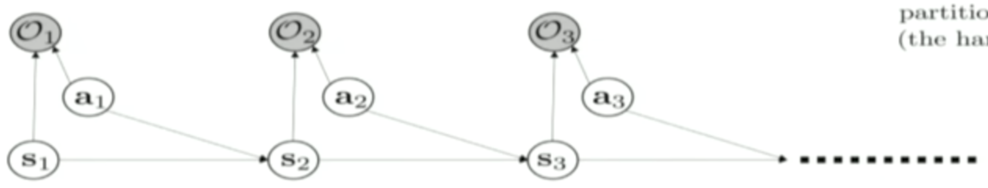
samples $\{\tau_i\}$ sampled from $\pi^\star(\tau)$

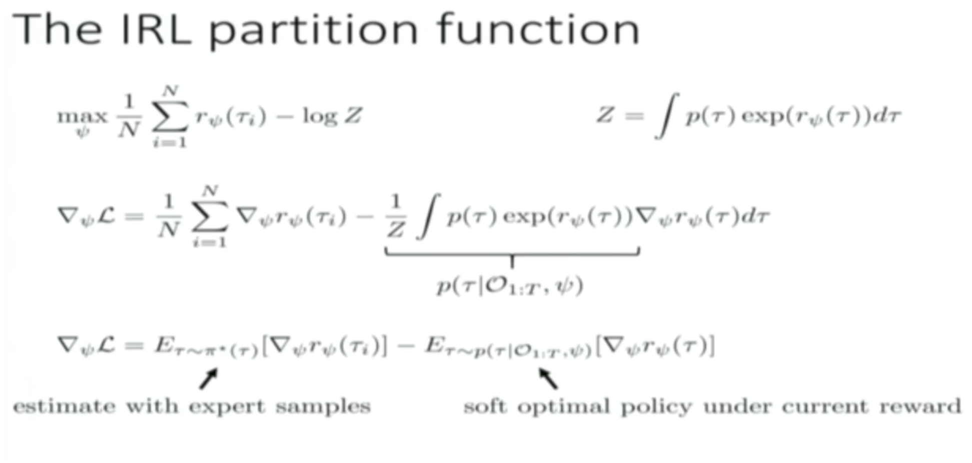
$p(\tau|\mathcal{O}_{1:T},\psi) \propto \cancel{p(\tau)} \exp\left(\sum_t r_\psi(\mathbf{s}_t,\mathbf{a}_t)\right)$

can ignore (independent of ψ)

maximum likelihood learning: $\max_\psi \frac{1}{N}\sum_{i=1}^{N} \log p(\tau_i|\mathcal{O}_{1:T},\psi) = \max_\psi \frac{1}{N}\sum_{i=1}^{N} r_\psi(\tau_i) - \log Z$

partition function (the hard part)





The IRL partition function

$$\max_{\psi} \frac{1}{N} \sum_{i=1}^{N} r_{\psi}(\tau_i) - \log Z \qquad Z = \int p(\tau) \exp(r_{\psi}(\tau)) d\tau$$

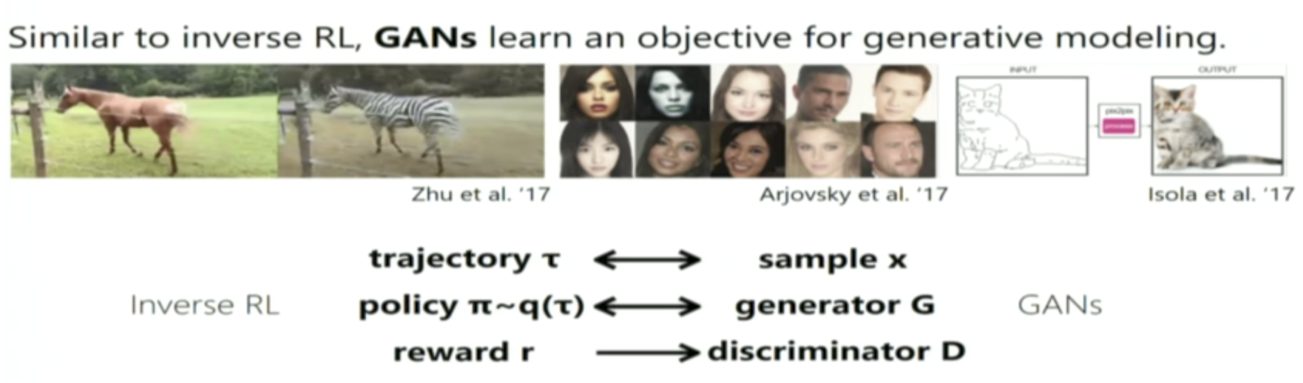
$$\nabla_{\psi} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\psi} r_{\psi}(\tau_i) - \underbrace{\frac{1}{Z} \int p(\tau) \exp(r_{\psi}(\tau))}_{p(\tau|\mathcal{O}_{1:T}, \psi)} \nabla_{\psi} r_{\psi}(\tau) d\tau$$

$$\nabla_{\psi} \mathcal{L} = E_{\tau \sim \pi^*(\tau)}[\nabla_{\psi} r_{\psi}(\tau_i)] - E_{\tau \sim p(\tau|\mathcal{O}_{1:T}, \psi)}[\nabla_{\psi} r_{\psi}(\tau)]$$

estimate with expert samples   soft optimal policy under current reward

The trick is to factor it in a way that the probability math discussed in the control vs inference can be used.

- The above algorithm, that learns the reward function by using math from the control-as-inference framework is called MaxENT IRL.
- Further mathematical details have been left out here → it is just algebra, calculus and probability manipulation → nothing too crazy. If you'd like to implement this, look up the math in the slides on inverse RL for cs 294.
- There are some inverse RL algorithms that learn the policy along with the reward function, in alternating training steps.
- Note this interesting connection between inverse RL and GANs:



Similar to inverse RL, **GANs** learn an objective for generative modeling.

Zhu et al. '17   Arjovsky et al. '17   Isola et al. '17

Inverse RL			GANs
	trajectory τ ⟷	**sample x**	
Inverse RL	**policy π~q(τ)** ⟷	**generator G**	GANs
	reward r ⟶	**discriminator D**	

The discriminator is the reward function where its saying is the expert likely to do this? Does this look like the expert? The policy is trying to produce samples (trajectories) that pass the discriminator's (reward function's) test.