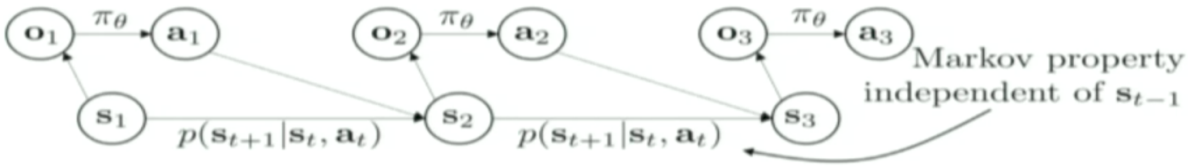
imitation learning

- we can formulate a supervised learning problem as RL by saying something like we want to learn $\pi_\theta(a|o)$ → distribution over labels (actions) given input (observation) → θ is the policy (the weights in the network)
- for a rl problem with fully observable state, policies are $\pi_\theta(a_t|o_t)$ = $\pi_\theta(a_t|s_t)$ where s_t is state that produces o_t. depending on the context (fully observable env or not), s_t and o_t can be different.
- probabilistic graphical model for states, actions, observations:

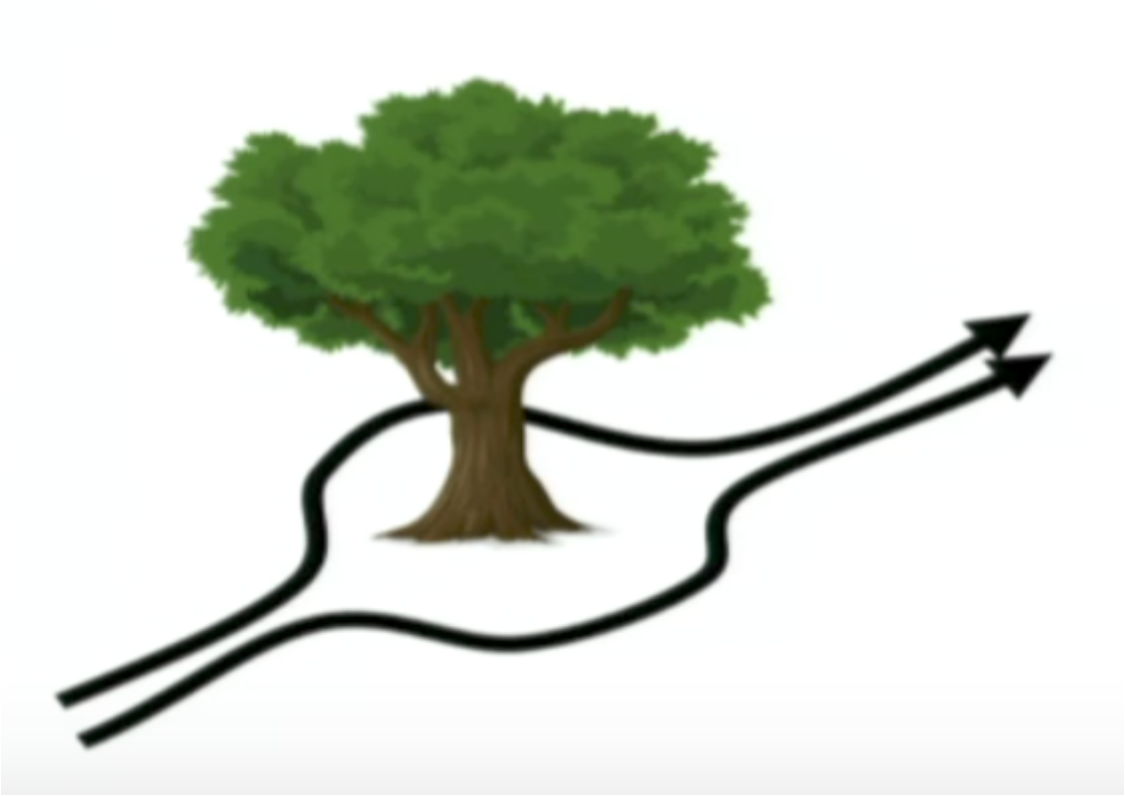


  note that states have the markovian property, but observations don't have this property (observations might have this property in fully observable environments)
- imitation learning is framing these series of actions kind of like a supervised learning problem where we map input to action based on training data we've seen. eg in the Alvinn self driving system we saw in cs 229.
- self driving framed as a supervised learning problem with observation → action mapping doesn't work well because it makes small mistakes each time and over time deviates from the path quite a bit. the sequential over time property of this problem makes the supervised learning approach work not so well.
- the fundamental problem with the above is that the distribution seen during training is quite different from the distribution seen during testing. to get around this, there is an algorithm called Dagger (Dataset Aggregation):
  do the following a few times in a **loop**:

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

  this forces the training data to encounter situations the policy would see.
- one of the fundamental disadvantages of dagger is that it needs a large manual labeling process → deep neural nets work well with lots of data so this is a real bummer.
- additionally, even if we have humans to do the labeling, we might not be able to get labels because it is sometimes hard for a human to say what he/she would have done in that situation (like steering angle in driving) just by looking at an image.
- we don't have a lot of theory that explains when and why imitation learning does and doesn't work.
- why might we fail to the expert?

- the expert's behavior might be non-markovian. to address this, we can consider a good amount of history instead of basing actions only based off of latest observations. RNNs might be useful here.
- multimodal behavior:



ways to get around the multimodal behavior problem:

- learn a mixture of gaussians output. here we will have the network output the parameters of a mixture of gaussians, but the downside is we have to pick the number of modes. if we pick it wrong, we encounter the same problem again.
- implicit density models: these estimate the distribution directly. sergey doesn't talk too much about these other than saying they are more complex and harder to train. ¯_(ツ)_/¯
- autoregressive discretization: here we discretize the space. discretizing a high dimensional space still falls to the curse of dimensionality → too many dimensions. so instead we use a neural net to predict the output of first dimension, and then conditioned on that we use another net to get output on second dimension etc.

- imitation learning sometimes works well, not always. we don't have good theoretical understanding of why. you can do some things like dagger to try to see if it works better.
- cost and reward are similar things (just negatives of each other). RL folks use reward and optimal control folks use the term "cost".
- look at the following RL problem:

$$r(\mathbf{x}, \mathbf{u}) = \begin{cases} 1 \text{ if object at target} \\ 0 \text{ otherwise} \end{cases}$$

$$\begin{aligned} r(\mathbf{x}, \mathbf{u}) = &- w_1 \|p_{\text{gripper}}(\mathbf{x}) - p_{\text{object}}(\mathbf{x})\|^2 + \\ &- w_2 \|p_{\text{object}}(\mathbf{x}) - p_{\text{target}}(\mathbf{x})\|^2 + \\ &- w_3 \|\mathbf{u}\|^2 \end{aligned}$$

- we care about the objective at the top, but we write down the reward function as in the bottom so that it is better suited for our optimization technique → continuous, differentiable and provides more feedback rather than just 0, 1. This is called reward shaping, where we slightly tune our reward/cost function to suit our optimization problem.
- The other trouble with reward functions, other than being hard to express/derive, is that its sometimes hard to evaluate it. if you're building a robot to put water into a glass, you'd have to build a vision system that says is the water in the glass? if you were to build this vision system, might as well use that to solve the problem instead of doing the RL approach. So reward function evaluation shouldn't be too crazy like this.