learning policies by imitating optimal control

- we looked at how to learn global and local models (local models with constraints) and do optimal control on these. what if we want to learn policies? can we backprapagate to learn policies?
- why learn policies?
  - policies can be much cheaper to execute computationally.
  - more interestingly, it seems like humans learn strategies to perform tasks. they don't make up dynamics and have cost functions and solve cost functions each time they perform a task. and this sometimes generalizes a lot better than learning a model of the physics eg:

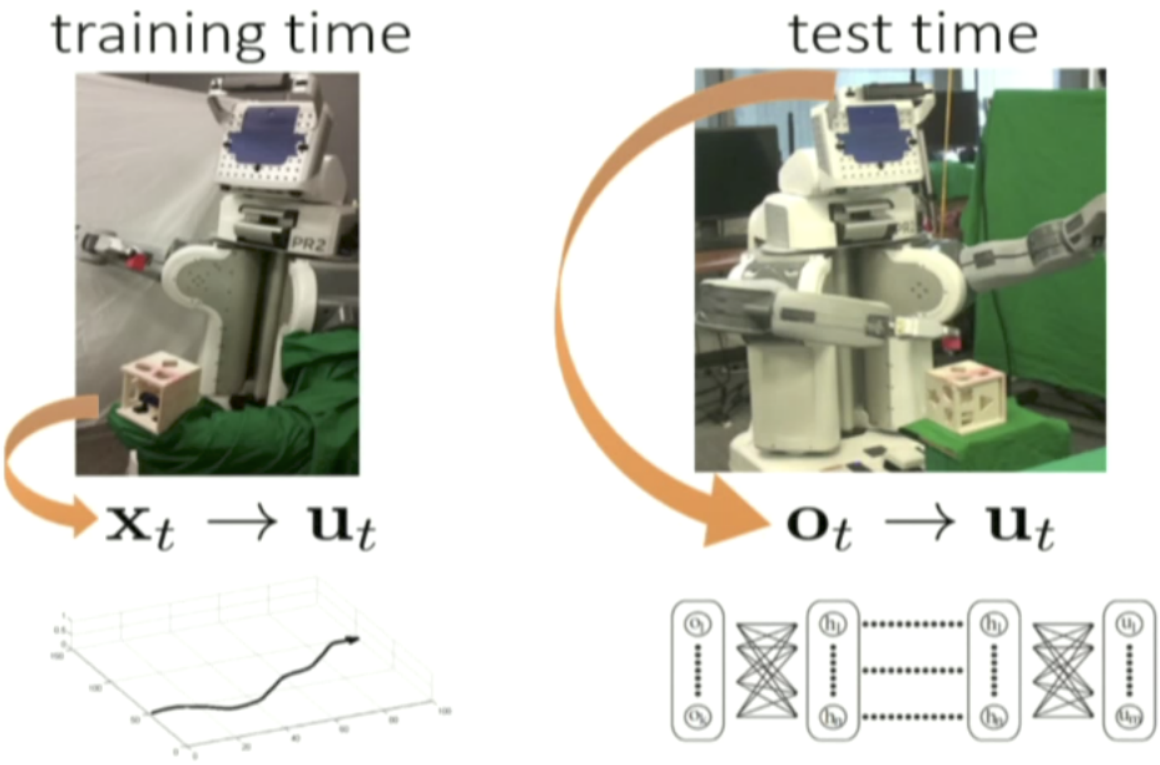
you are not gonna solve physics equation of motions while catching!

- training the parameters of a policy with backprop is like training the parameters in an RNN with backprop thru time (same parameters applied at each time step in both RNN and in control policy). except that, in the case of RNNs, we have LSTMs with convenient numerical properties that behave well and navigate around vanishing and exploding gradient problems. here we don't have that convenience. dynamics is dictated by nature, not picked by us.
- the policy equivalent of the collocation view is:

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T,\mathbf{x}_1,\ldots,\mathbf{x}_T,\theta} \sum_{t=1}^{T} c(\mathbf{x}_t,\mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1},\mathbf{u}_{t-1}), \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$

- the part upto the first constraint can be boxed as a trajectory optimization problem (that can be solved shooting way (like LQR) or collocation way → basically some planning/control black box). now we ask the question is how can we impose constraints on our trajectory optimization/planner? if we can do that, we can solve the above problem.

- how do we do this? Lagrangians! we can convert our constrained optimization problem into an unconstrained optimization problem using lagrangians. this is also called dual gradient descent.
- in practice, we add another term to the above unconstrained optimization problem which is $\rho||C(x)^2|| = \rho||\pi_\theta(x_t) - u_t||^2$. This term penalizes actions that the policy cannot imitate. If the constraint of taking actions according to policy is perfectly satisfied, then this term has no effect but in practice it makes it work better.
- what we really do as we solve this above problem is that we are finding good actions (trajectory optimization) and fitting the parameters of the policy (like supervised learning) to take the actions that the expert computed during trajectory optimization took. This is why this strategy is called learning a policy by imitating an optimal controller.
- Because of the above-mentioned term $\rho||C(x)^2|| = \rho||\pi_\theta(x_t) - u_t||^2$, the teacher tries to take actions that the policy can implement and avoids actions the learner (the policy) can't mimic.
- the above is called **guided policy search.**
- But! All the math we did is in terms of state and actions. In the real world, policies often operate with observations, not necessarily state. A trick (called input remapping trick) is to train with full state (using simulators?), map state to observation and train policy with observations instead of state, as in:



- here we saw how to use imitation learning with optimal LQR controllers. similarly, we can use variants of imitation learning with variants of controllers/planners like MCTS etc.
- Note that dagger only works when it is possible for the policy to match the expert's behavior, up to a bounded loss (on training set). This assumption can be violated easily, especially in partially observed domains.
- While using dagger with guided policy search, the expert can adapt to produce actions that the policy can mimic so we sort of get around the above mentioned constraint (as long as said adaption of expert is possible)

- the big downside of the model based RL techniques discussed so far is that we need a model! sometimes we don't have one and it can be hard to learn (globally or locally).
- model based RL methods rely on physics of the system (dynamics) + optimal control/trajectory planning/policy learning
- model free RL is more like trial and error learning. it often ends up being slower to learn (because there isn't an expert to guide us?), but often times more general.
- the broad way model free RL we will learn about works is by just trying a bunch of trajectories, seeing which ones work and increasing the probability of those. no dynamics model, no optimal control/trajectory optimization etc.