Last Time:

- implementing uncertainty quant.

Lecture 14

SP '26

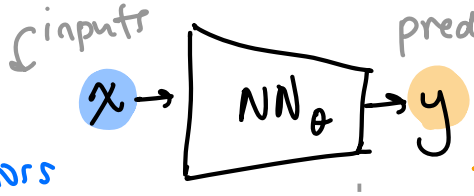
Andrea Bajcsy

This Time:

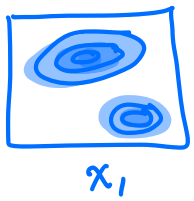
- System-level failures

Component vs. System-level Failures

So far, we have discussed a suite of UQ techniques for a data-driven model:



→ Density estimators

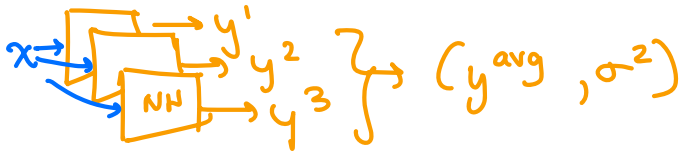
$\tilde{p}(x|\mathcal{D})$: 

→ Cosine similarity:

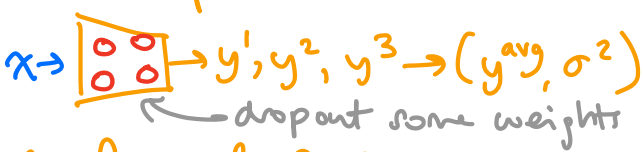
$$s(x;\mathcal{D}) = \max_{\bar{x}\in\mathcal{D}} \frac{\bar{x}^T x}{\|\bar{x}\|\,\|x\|}$$

"gets most similar prior experience from $\mathcal{D}$ to construct score"

→ Ensembles

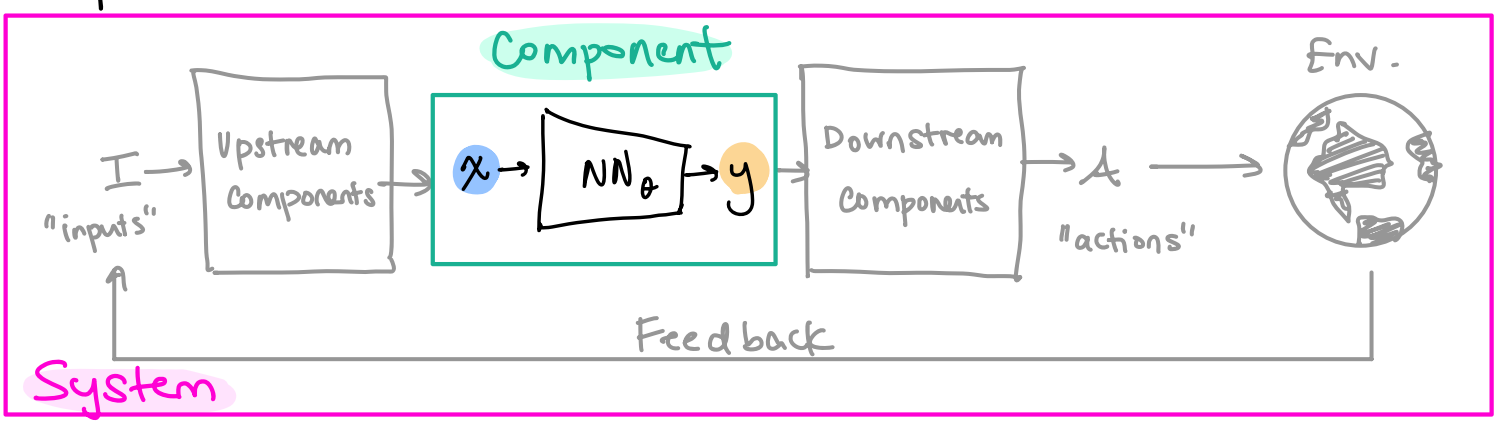


→ MC Dropout



→ Conformal Pred.

But, all these models live within a complex "autonomy pipeline" — ex. the NN could be perception module, trajectory forecaster, LLM task planner...



Q How can we model, identify, & repair component-level errors that cause system-level degradation?

MODEL

How do we mathematically model component-level errors that cause system-level degradation?

Broadly, all methods that study this problem end up tackling something that looks like this:

trajectory resulting from executing system when component sees input x

$$\text{find} \quad x \in \mathcal{X} \quad \text{s.t.} \quad \text{Cost}\left[\xi^{NN(x)}\right] \geq \delta$$

x: inputs into the learned component

Cost: function that evaluates performance / safety / risk / .... of closed-loop behavior.

Intuitively, we want to find inputs into our component that cause our system to degrade. Then we can define:

$$\mathcal{X}^{\text{anomaly}} := \{x : \text{Cost}\left[\xi^{NN(x)}\right] \geq \delta\}$$

as the set of all "system-level anomalies"

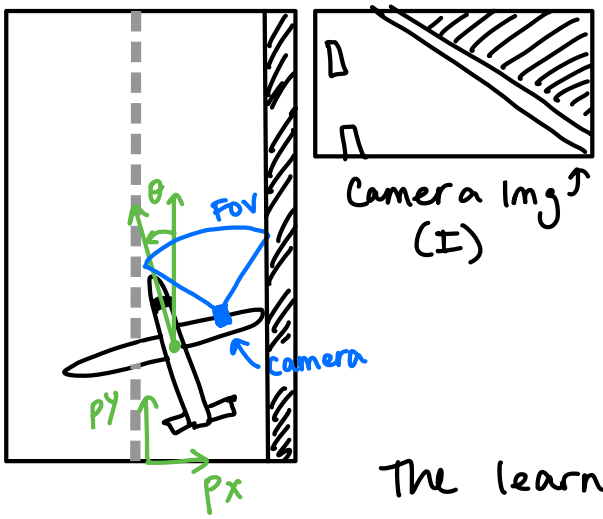
IDENTIFY

Ok, but how do we practically instantiate this? Seems really hard (and it is!) but lets look @ two case studies to understand:

Case Study #1: Perception Module.

↳ From: Chakraborty & Bansal. "Discovering Closed-loop Failures of Vision-Based Controllers ..." RA-L, 2023.

The setup: You have an aircraft w/ camera on right wing.



The true state + dyns. of system are modelled as:

$$\dot{p}_x = v\cos\theta$$

$$\dot{p}_y = v\sin\theta$$

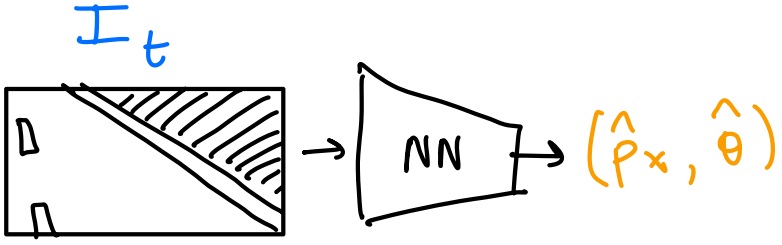
$$\dot{\theta} = u \leftarrow \text{control ang. vel.}$$

p_x = "cross track error"

p_y = "downtrack error"

θ = "heading error"

The learned perception model (is a CNN) takes as input the image I_t and returns an estimated (p_x, θ)



Then, a proportional controller (P-ctrl) steers the aircraft towards the centerline:

$$\pi(I_t) := \tan(-0.74\,\hat{p}_x - 0.44\,\hat{\theta}) = u$$

The failure states of the overall system are the aircraft leaving the runway: $F = \{x : |p_x| \geq 10\}$

Thus, our goal is: find $I_{0:T}$ s.t. $\exists t \in [0,T]$, $\xi^{\pi(I_t)}(t) \in F$

Q What type of problem does this remind you of?

[A] Key IDEA: we can pose this as a reachability problem! But, searching over space of all $\mathcal{I}_{0:T}$ is too high-D.... We can exploit a trick here to reduce dimensionality:

In simulation, we know the sensor mapping from any true state x_t to the image $\mathcal{I}_t$! Let $\mathcal{I}_t = S(x_t; E)$ be this sensor map from true state x_t in environment E.

So, we can rewrite:

$$u = \pi(\mathcal{I}_t) = \pi(S(x_t, E)) \Rightarrow \hat{\pi}(x_t)$$

This is the closed-loop, state feedback policy

Now, compute the BRT for all true states:

$$BRT_x := \{ x : \exists \tau \in [0,T] \ \zeta_x^{\hat{\pi}}(\tau) \in \mathcal{F} \}$$

And then transform it into the image space to get the set of all failure inputs:

$$BRT_{\mathcal{I}} = \{ \mathcal{I} : \mathcal{I} = S(x; E), x \in BRT_x \}$$

Recipe: ① get closed-loop dyns. $\hat{\pi}$ via uniform sampling over state space $\mathcal{X}$ & then rendering $\mathcal{I}$ + getting u
② computing $BRT_{\mathcal{I}}$ to get set of images that will lead to failures.

⚠ Compared to fwd-simulation methods:
→ reachability: 6.5 hours (most time on rendering)
→ fwd sim: ~67.5 DAYS

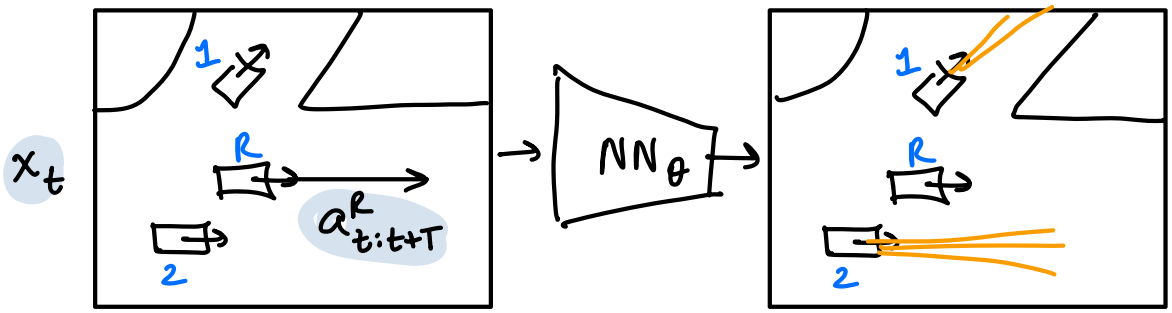
CASE STUDY #2: Trajectory Forecasting

↳ From: Nakamura, Tian, Bajcsy. "Not All Errors Are Made Equal." CORL, 2024.

The setup: We have a learned trajectory forecasting model:

$$P_\theta(a^1_{t:t+T}, \dots a^M_{t:t+T} \mid a^R_{t:t+T}, x_t)$$

- $a^1_{t:t+T}, \dots a^M_{t:t+T}$: future behavior of M agents
- $a^R_{t:t+T}$: Robot plan
- x_t: joint state: $x_t = [x^1_t, x^2_t \dots x^M_t, x^R_t]$



We also have a robot policy:

$$\pi_\phi(x, P_\theta) = a^R_{t:t+T}$$

- P_θ: preds.
- $a^R_{t:t+T}$: ex. sequence of linear + angular ctrls
- x: current state of world

Deploy the robot for T timesteps in real world in N scenarios (ex. diff init conds, environments, cities, etc) and collect

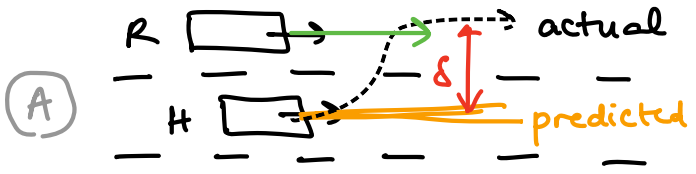
$$\mathcal{D} = \{(x_{0:T}, a^{1:M}_{0:T}, a^R_{0:T})_n\}_{n=1}^N$$

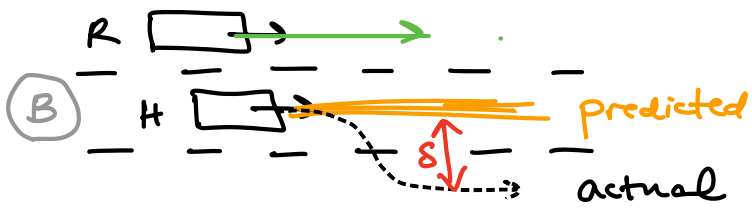
- $x_{0:T}$: observed joint state traj.
- $a^{1:M}_{0:T}$: observed agent actions
- $a^R_{0:T}$: executed robot actions

This data contains real interactions btwn. the agents + robot.

We want to find $\boxed{\mathcal{D}_{fail} \subset \mathcal{D}}$ which have system-level prediction failures: errors were poor predictions degraded perform.

Naive IDEA 1: Why not construct $\mathcal{D}_{fail}$ as all scenarios where the predicted vs. real actions of $a^{1:M}_{0:T}$ deviated?





$$\delta = \| a^H_{0:T} - \hat{a}^H_{0:T} \|_2 \quad \equiv \quad \delta = \| a^H_{0:T} - \hat{a}^H_{0:T} \|_2$$

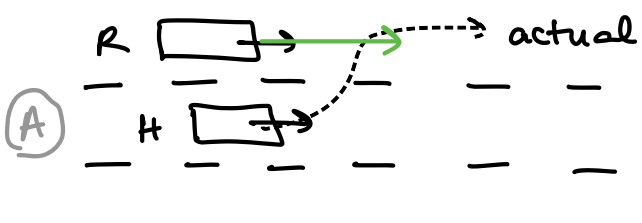
↑ SAME L2 prediction error ↑
BUT (A) IS MUCH MORE THAN (B)

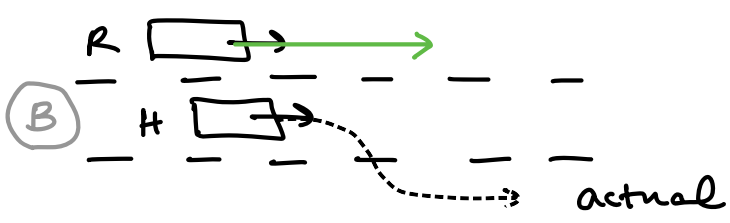
KEY IDEA: system-level prediction errors are high regret:

$$\text{Regret}_t := \max_{\bar{a}^R_{t:t+T}} R_\phi(\bar{a}^R_{t:t+T}, a^{1:M}_{t:t+T}, x_t) - R_\phi(a^R_{t:t+T}, a^{1:M}_{t:t+T}, x_t)$$

$\bar{a}^R_{t:t+T}$: reward of the best robot decision in hindsight

reward of executed robot decision AND observed agent behavior



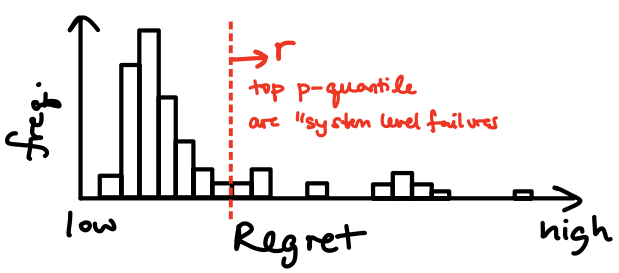


$\max_{a^R} [R(a^R, \cdots)] - R(\rightarrow, \cdots)$

In hindsight, should have slowed down! ⇒ HIGH REGRET!

$\max_{a^R} [R(a^R, \cdots)] - R(\rightarrow, \cdots)$

In hindsight, would have still done → ⇒ LOW REGRET!



$$\mathcal{D}_{fail} = \{ d = (a^{1:M}, a^R, x) : \text{Regret}(d) \text{ is in top } p\text{-quantile} \}$$

NOTE: high regret scenarios are more than collisions! ex. inefficiency is also high regret.

REPAIR

Identification is hard ... repair is even harder. But, there are two main techniques:

① re-training the faulty predictive model / component on the system-level failure data.

② monitor + fallback planning

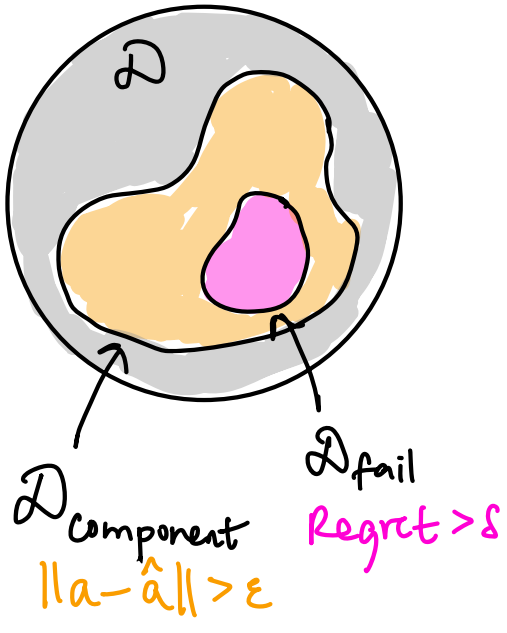
① In the perception case-study, we can take all these:

$$BRT_I = \{ I : I = S(x; f), x \in BRT_X \}$$

and re-train the prediction model $I_t \rightarrow \square \rightarrow (\hat{x}, \hat{\theta})$

↳ in paper from above, re-training results in a 20% decrease in the BRT_I after re-training and 10% error reduction in the prediction error.

In the trajectory forecasting case-study, we can



If you fine-tune model on only $\mathcal{D}_{fail}$ (which is 77% less data than $\mathcal{D}$!) reduces regret and collision cost during re-deployment by 53% & 65% respectively. BUT if you only fine-tune on LOW-REGRET DATA, you don't see statistically significant improvement.