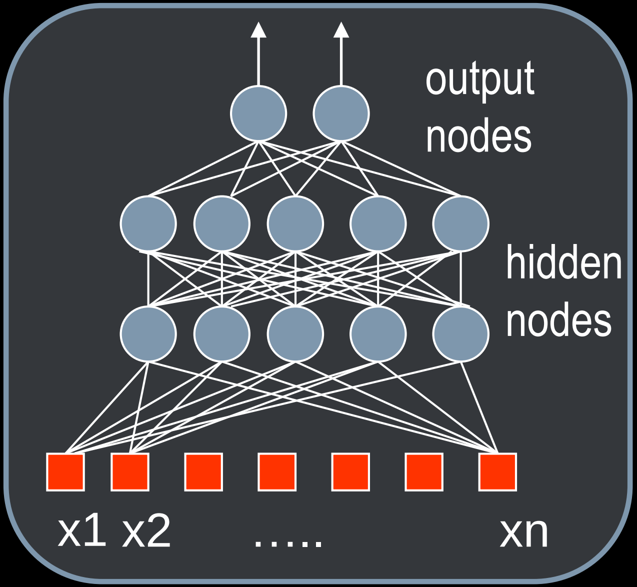
Learning Dynamics with Uncertainty

Last time…

- We talked about Neural Networks



- Can we always trust the output of a neural network?
  - What if the input we give it is not similar to the training data?
- How do we quantify the uncertainty of the network given a new input?
  - This is important if we want to use neural networks in safety-critical robots

Outline

- Case study: learning dynamics
- Types of uncertainty
  - Aleatoric
  - Epistemic
- Example Paper: Learning where to trust dynamics models for Deformable object manipulation

Motivation

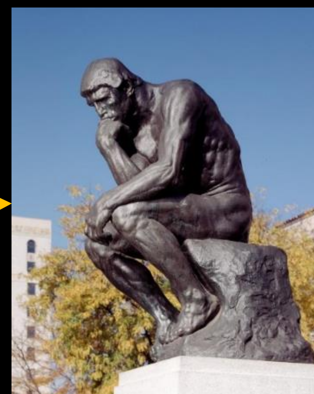
- Controllers, motion planners, Kalman/Particle filters, etc… need a dynamics function

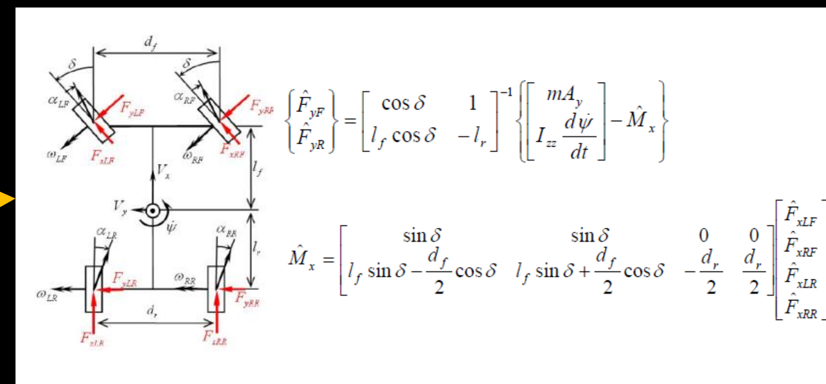
- In robotics, usually a discrete-time function:

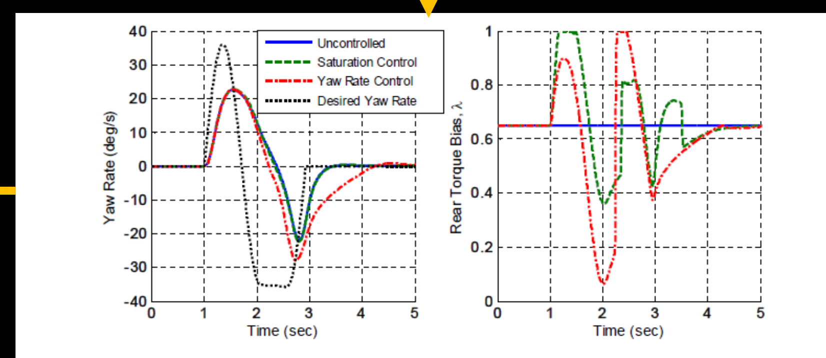
$$x_{t+1} = f(x_t, u_t)$$

- How do we get this function?

Traditionally: Write down a dynamics function by hand





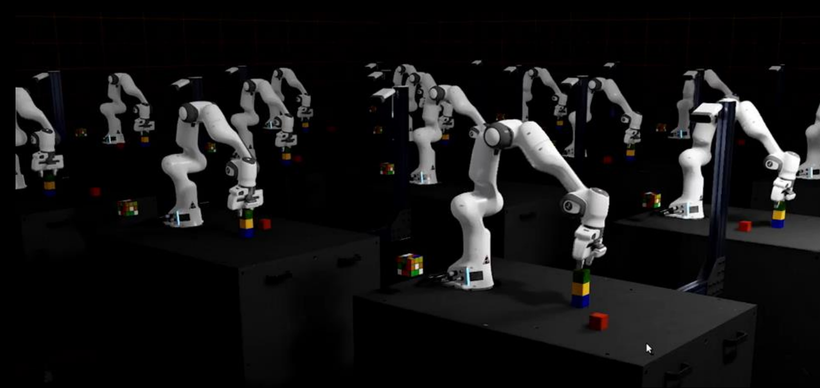


[Sill and Ayalew, International Journal of Vehicle Design, 2013]

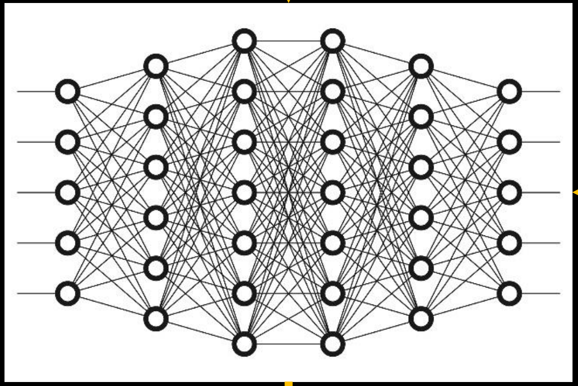
Very hard to model complex dynamics by hand

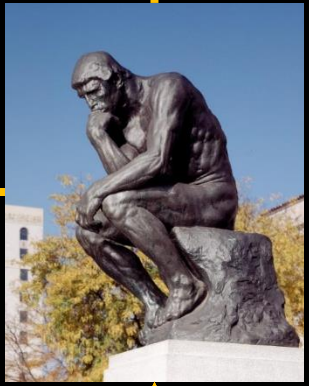
Recently: Machine Learning to the rescue!

Google Arm Farm



NVIDIA Isaac SIM

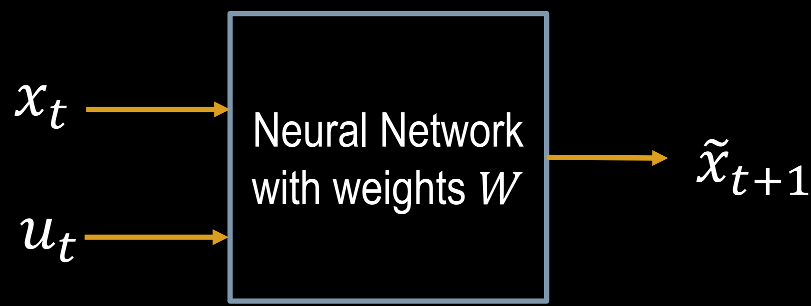




Dynamics function or Policy

Neural Networks for Learning Dynamics

1. Collect data of the form (x_t, u_t, x_{t+1}) from the system you want to model
2. Design a neural network (to do multi-dimensional regression):



3. Specify the Loss function. This quantifies the error between the output of the network and the desired output, e.g:

$$L(x_t, u_t, W) = ||x_{t+1} - NN(x_t, u_t, W)||$$

4. To find the weights try to solve this optimization problem:

$$\underset{W}{\operatorname{argmin}} \sum_t L(x_t, u_t, W)$$

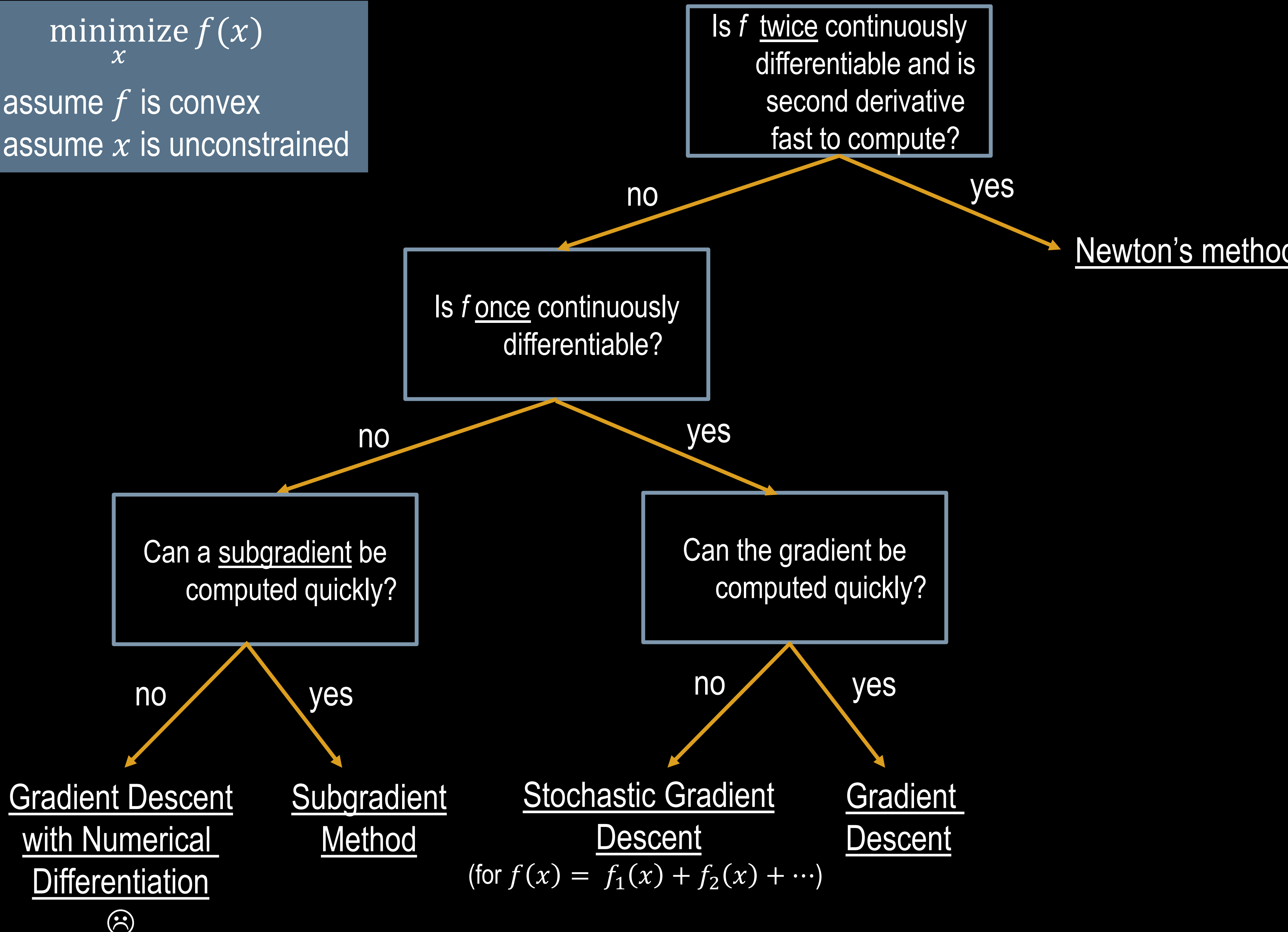
How do we solve this optimization problem?

$$\underset{W}{\operatorname{argmin}} \quad \sum_t L(x_t, u_t, W) \qquad\qquad L(x_t, u_t, W) = ||x_{t+1} - NN(x_t, u_t, W)||$$

- First, note that the *NN* function is NOT convex
  - We can only hope to find a local minimum ☹
- Computational Issues:
  - The data-set can be very large
  - The Neural Network could be very complex (many layers)
- What method should we use to find the local minimum?
  - Gradient Descent?
  - Newton's method?
- Computing the gradient of the NN function can be very expensive because it has to be done for all the data-points
- No hope of computing the Hessian (way too expensive), which is needed for Newton's method

$$\underset{x}{\text{minimize}}\ f(x)$$

- assume f is convex
- assume x is unconstrained



*Many more methods not covered!

How do we solve this optimization problem?

$$\underset{W}{\operatorname{argmin}} \quad \sum_t L(x_t, u_t, W) \qquad\qquad L(x_t, u_t, W) = ||x_{t+1} - NN(x_t, u_t, W)||$$

- If you can't optimize for the entire dataset of n datapoints at once, use Stochastic Gradient Descent (SGD)!
  - Recall the update rule for descent methods:

$$W^{(k+1)} = W^{(k)} + t^{(k)} \Delta W^{(k)}$$

  - Compute gradients for a batch (i.e. subset) of datapoints at each iteration:

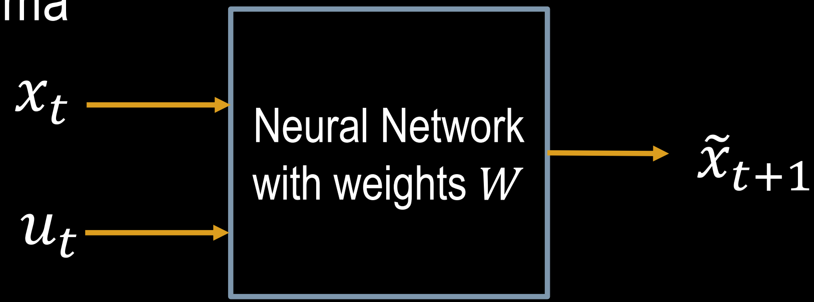
$$\Delta W^{(k)} = -\nabla L\left(x_i, u_i, W^{(k)}\right) - \nabla L\left(x_j, u_j, W^{(k)}\right) \ldots \qquad \text{for some } i, j, \ldots \leq T$$

  - Set the learning rate $t^{(k)}$ to a small constant, e.g. 0.01.
  - Iterate until the objective function value stops changing (or for a set number of iterations)

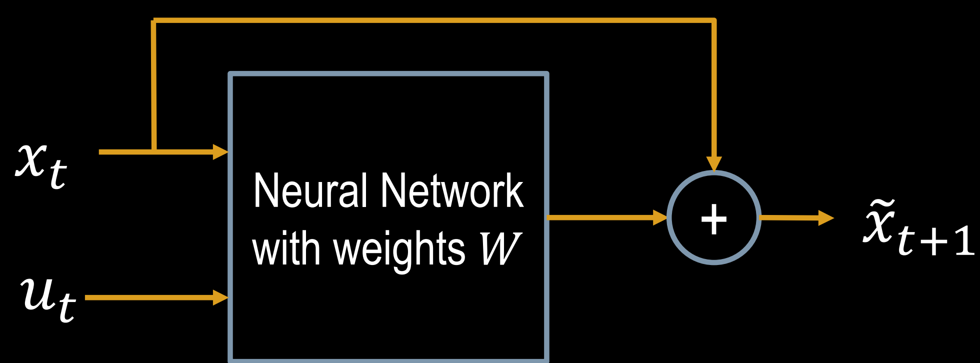
- Don't write the code yourself: Tensorflow or pyTorch (with Keras) make this very easy to do

Neural Networks for Learning Dynamics

- This network structure works in theory, but in practice often converges to bad local minima



- A more effective approach is to learn $\tilde{x}_{t+1} = x_t + NN(x_t, u_t, W)$



- This is called a residual model: learning to add a (small) change to an input

Why is a residual model better for learning dynamics?

- Consider a neural network with small random weights (usually how you initialize it)
- For any (x_t, u_t), $NN(x_t, u_t, W) \approx \epsilon$ (a vector of small numbers)
- For a *standard* model:

$$L(x_t, u_t, W) = \|x_{t+1} - NN(x_t, u_t, W)\| = \|x_{t+1} - \epsilon\| \approx \|x_{t+1}\|$$

- If $||x_{t+1}||$ is not near zero, you are starting with a big loss (i.e. you're far from a good local minimum)

- Instead, for a *residual* model:

$$L(x_t, u_t, W) = \|x_{t+1} - (x_t + NN(x_t, u_t, W))\|$$
$$= \|x_{t+1} - (x_t + \epsilon)\| \approx \|x_{t+1} - x_t\|$$

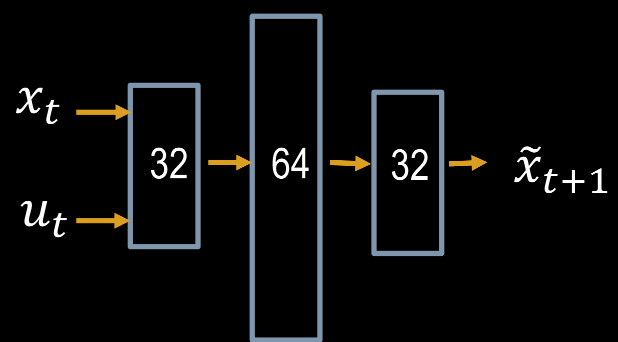
- If the state doesn't change too much (i.e. small time steps), you start with a lower loss (i.e. closer to a good local minimum)

Example dynamics learning

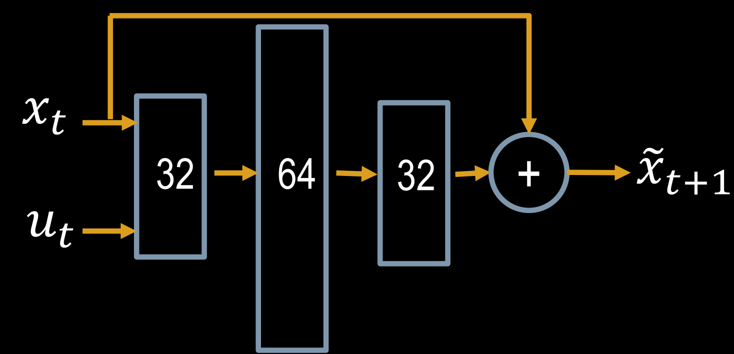
Ground truth dynamics: $x_{t+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} -0.2 & 0.1 \\ 0.15 & 0.15 \end{bmatrix} u_t$

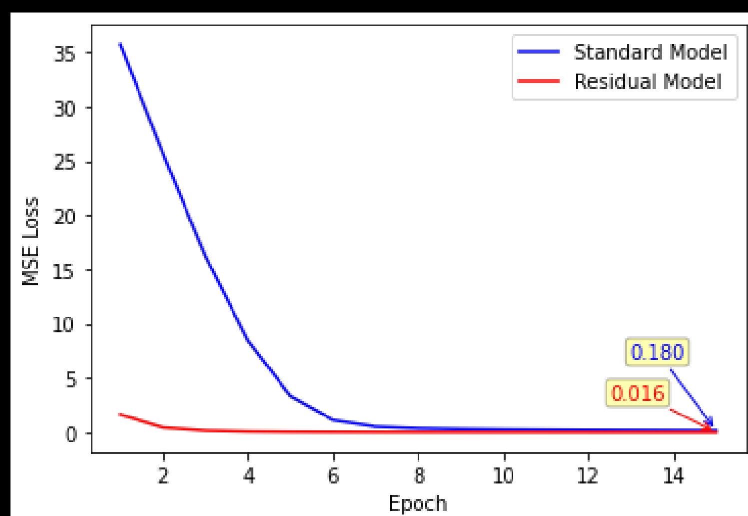
x, u range from
-1 to 1 in each dimension

Standard model



Residual model





trained on 100,000 random samples

Uncertainty in Learned Models

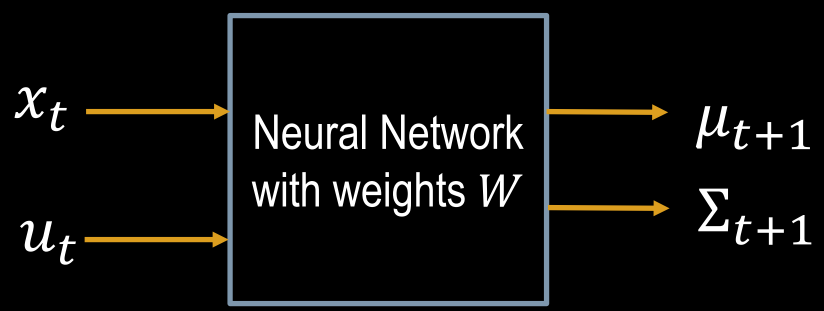
Two Types of Uncertainty

- Aleatoric Uncertainty
    - Uncertainty due to random chance, i.e. uncertainty inherent in the process (e.g. stochastic dynamics)
    - Getting more data will NOT reduce this kind of uncertainty

- Epistemic Uncertainty
    - Uncertainty due to a lack of information
    - Getting more data WILL reduce this kind of uncertainty

How do we account for *aleatoric* uncertainty?

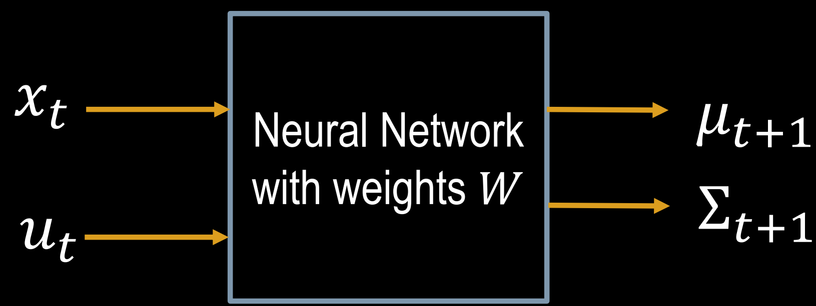
- Since we assume the function is inherently stochastic, we shouldn't output only a single prediction
- Instead, let's output a probability distribution!
  - E.g., for dynamics learning, output a Gaussian with mean μ and covariance matrix Σ:



Since covariance matrix is symmetric, only need to output half of it

- The output is called a predictive distribution
- Can also use a residual form here
- What should the loss function L be?

How do we account for *aleatoric* uncertainty?



- The Loss function should compute how likely it is that x_{t+1}(from training data) was sampled from the distribution $N(\mu_{t+1}, \Sigma_{t+1})$

- To get this, we just use the equation for Probability Density of a Gaussian:

$$L(x_t, u_t, W) = \frac{\exp(-0.5(x_{t+1} - \mu_{t+1})^T \Sigma_{t+1}^{-1} (x_{t+1} - \mu_{t+1}))}{\sqrt{(2\pi)^n \det(\Sigma_{t+1})}}$$

where n is the dimension of x

How do we account for *aleatoric* uncertainty?

- To find weights W, try to solve

$$\underset{W}{\operatorname{argmax}} \prod_t L(x_t, u_t, W)$$

- "argmax" because we want to *maximize* the likelihood x_{t+1} is sampled from the distribution $N(\mu_{t+1}, \Sigma_{t+1})$ for each datapoint.
- Using product because we want to maximize the probability that the *entire dataset* was generated by the learned function
- Products of small probabilities cause numerical problems
  - So use logs to make this numerically reliable:

$$\underset{W}{\operatorname{argmax}} \sum_t \log L(x_t, u_t, W)$$

- This is Maximum Likelihood Estimation (MLE)!
- Find a local minimum using SGD (like before)

Two Types of Uncertainty

- Aleatoric Uncertainty
  - Uncertainty due to random chance, i.e. uncertainty inherent in the process (e.g. stochastic dynamics)
  - Getting more data will NOT reduce this kind of uncertainty

- Epistemic Uncertainty
  - Uncertainty due to a lack of information
  - Getting more data WILL reduce this kind of uncertainty

Epistemic uncertainty in dynamics learning

- Let's say you learn the weights for the neural network from the dataset (x_t, u_t, x_{t+1}) for $t = 1 \dots T$
- Now you get a new input x_{new}, u_{new}

Should you trust the output, $\widetilde{\boldsymbol{x}}_{\boldsymbol{new+1}}$, is correct?

- Common assumption: The training data and the new data are i.i.d. So, *if you have enough data*, you can trust the output is correct.

- In reality: You never really have enough data in high-dimensional spaces.
  - Correctness of output often depends on which x_{new}, u_{new} you give the network.
  - Some x_{new}, u_{new} will be out-of-distribution (OOD) w.r.t the training data.

How do we account for *epistemic* uncertainty?

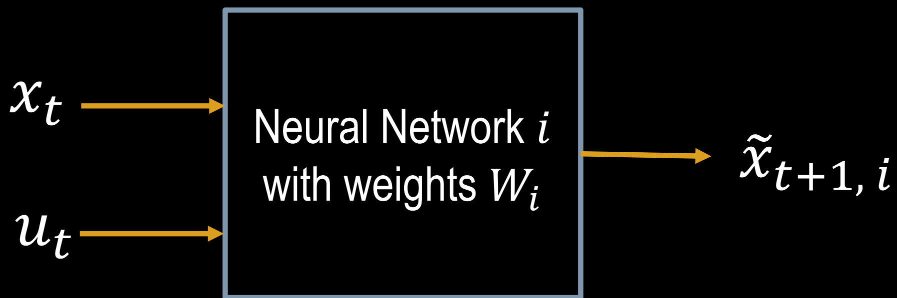
- Simple Idea: Measure distance between (x_{new}, u_{new}) and all (x_t, u_t) for $t = 1 \dots T$ in dataset.
  - The farther (x_{new}, u_{new}) is from the nearest (x_t, u_t), the less we should trust the network's prediction.

- This may work for simple systems, but
  - Need to come up with a distance metrics (hard to do for complex state representations)
  - Distance in input space may not be very informative (e.g. when many inputs map to the same output)
  - Need to keep the entire dataset

- Is there a better way?

How do we account for *epistemic* uncertainty?

- Many recent papers on estimating epistemic uncertainty
- One simple approach: use an ensemble of neural networks
  - Train K copies of the neural network with the same data, but *initialize each copy with different random weights*



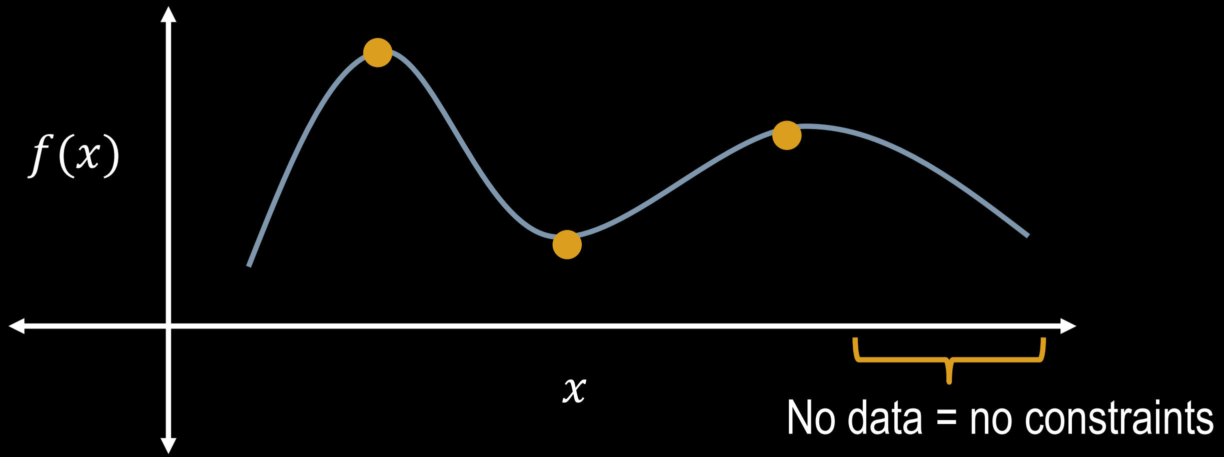
For $i = 1 \ldots K$

  - Train each network independently with SGD
- So for a given (x_t, u_t) input, we will have K predictions:

$$\tilde{X}_{t+1} = (\tilde{x}_{t+1,\,1}, \ldots, \tilde{x}_{t+1,\,K})$$

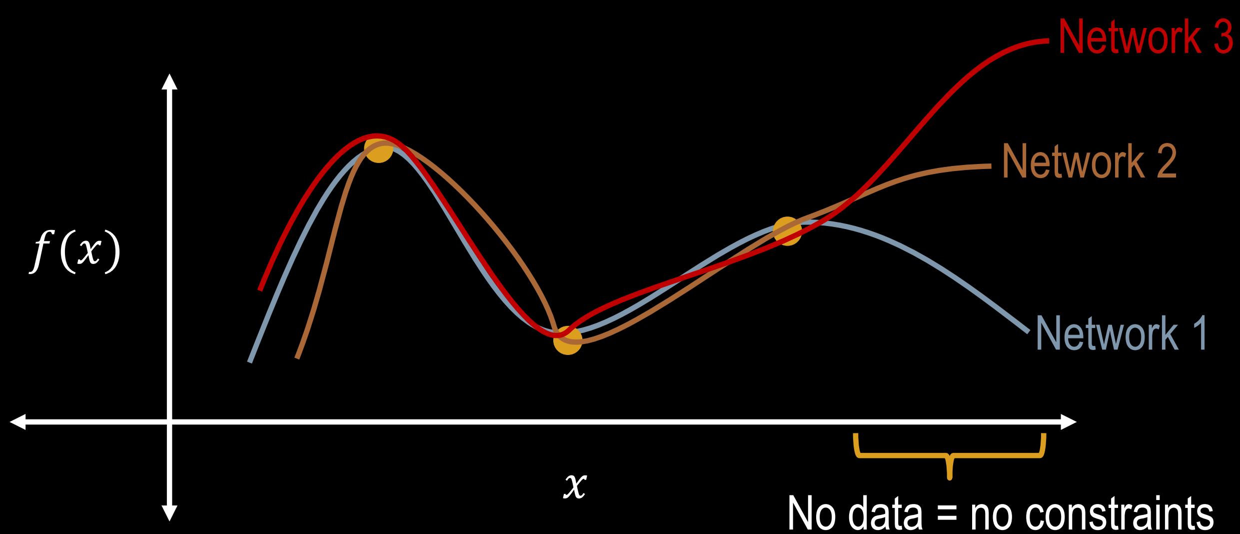
Understanding ensembles

- Example: Consider learning a 1D function from datapoints
  - The datapoints are basically constraints on the output of the function for certain inputs
- Assuming the function is smooth, it will be well-behaved when close to training datapoints
  - But far from the training data, the function is not constrained



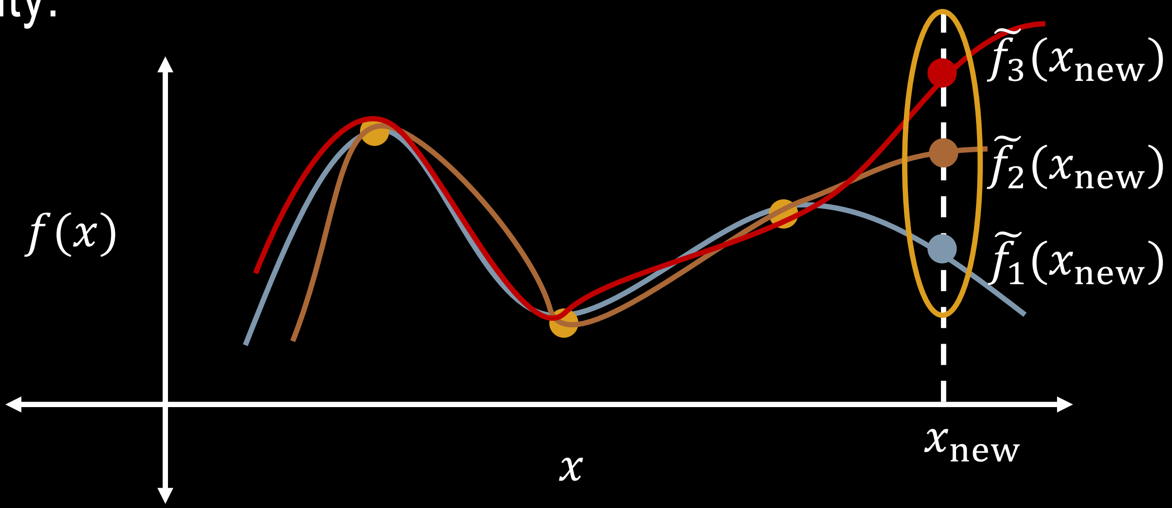
Understanding ensembles

- An ensemble is basically a set of functions that are consistent near the training data
- But since they were initialized with different random weights, *there is no reason for the functions to be consistent far from the training data*



How do we account for *epistemic* uncertainty?

- The *variance* in the predictions of the networks is an estimate of epistemic uncertainty:



- For dynamics learning, have K predictions $\tilde{X}_{t+1} = [\tilde{x}_{t+1,\,1}, \dots, \tilde{x}_{t+1,\,K}]$:

1. Compute mean: $\bar{x}_{t+1} = \sum_{i=1}^{K} \frac{\tilde{x}_{t+1,i}}{K}$ ← The mean of the predictions is the overall prediction of the ensemble
2. Subtract $\bar{x}_{t+1}$ from every column of $\tilde{X}_{t+1}$
3. Compute variance $V = \frac{\tilde{X}_{t+1}\tilde{X}_{t+1}^{\mathrm{T}}}{(K-1)}$
4. $\mathrm{trace}(V)$ is an estimate of epistemic uncertainty (higher = more uncertainty)

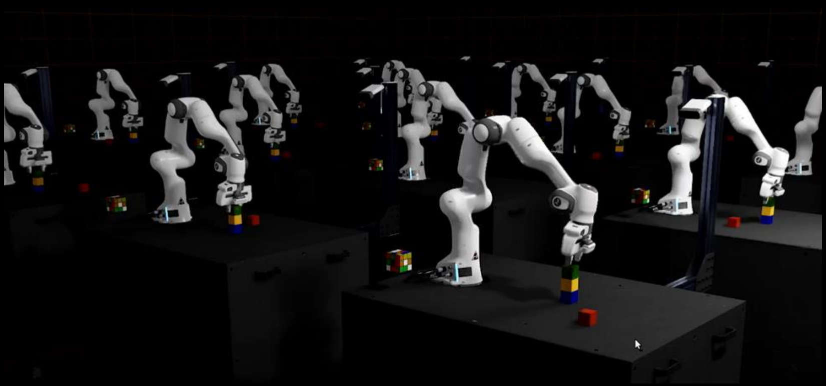
Two Types of Uncertainty

- Aleatoric Uncertainty
  - Uncertainty due to random chance, i.e. uncertainty inherent in the process (e.g. stochastic dynamics)
  - Getting more data will NOT reduce this kind of uncertainty
- Epistemic Uncertainty
  - Uncertainty due to a lack of information
  - Getting more data WILL reduce this kind of uncertainty
- What if you have both at the same time?
  - Create an ensemble of models that output predictive distributions

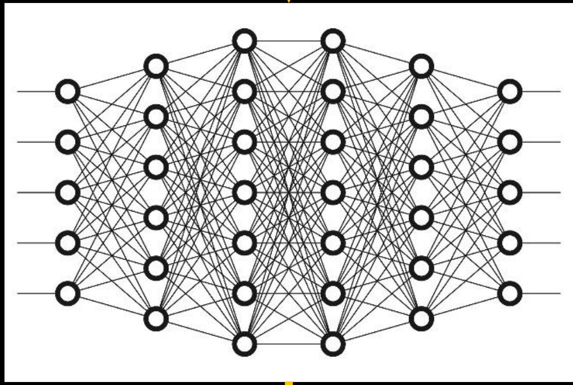
Break

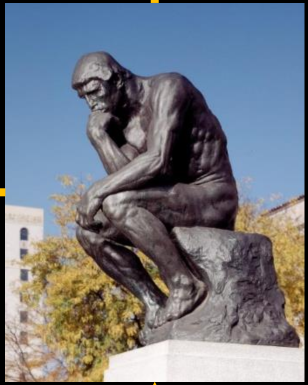
Recently: Machine Learning to the rescue!

Google Arm Farm



NVIDIA Isaac SIM





Never enough data to cover high-dim. state/action space

Dynamics function or Policy

VB

The Machine

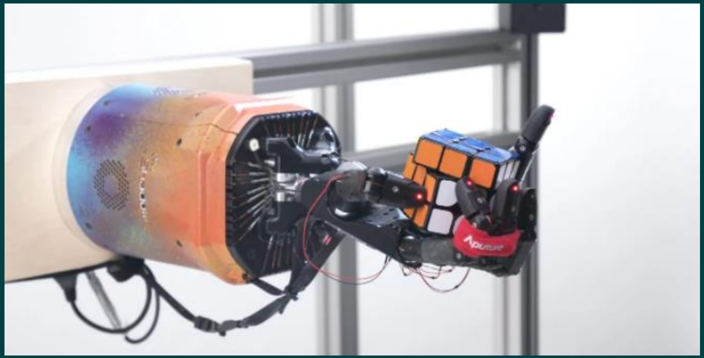
Making sense of AI

OpenAI disbands its robotics research team

Kyle Wiggers

@Kyle_L_Wiggers

July 16, 2021 11:24 AM



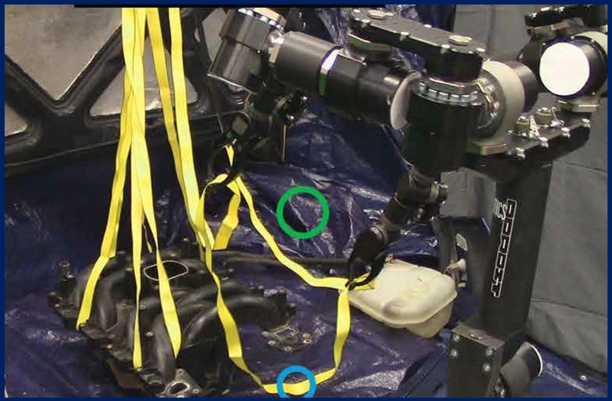
"Company cofounder Wojciech Zaremba quietly revealed…that OpenAI has shifted its focus to other domains, <u>where data is more readily available</u>."

Proposition: Our dynamics models will never be reliable for all states/actions of a real-world robot

Reasoning about Dynamics Uncertainty for Deformable Object Manipulation

What do we do when our dynamics models are unreliable?

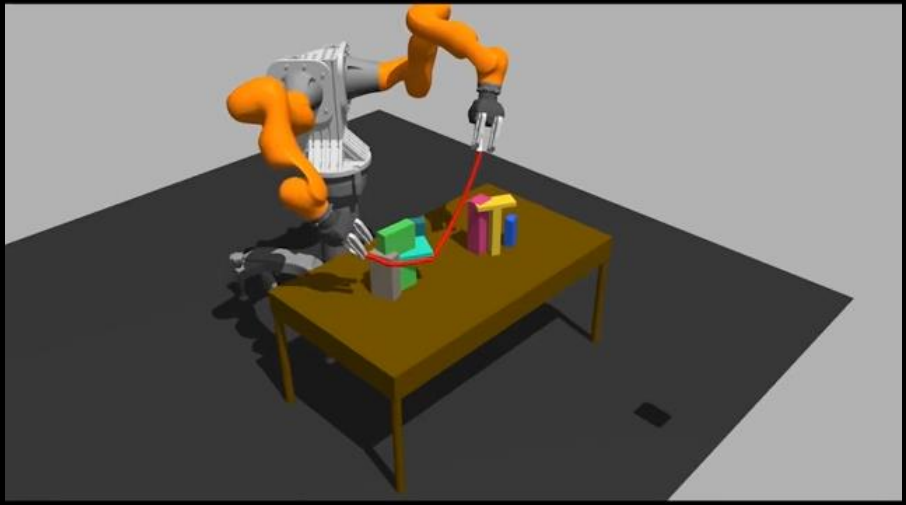
Learn to avoid a model's mistakes



[Mitrano, McConachie, and Berenson, Science Robotics 2021]

Learning dynamics for deformable object manipulation

- Goal: Learn a dynamics model to use for motion planning for a rope amongst obstacles:



- Problem: We haven't found any method that will actually learn a good estimate of these dynamics from data (even if we have a lot of data)

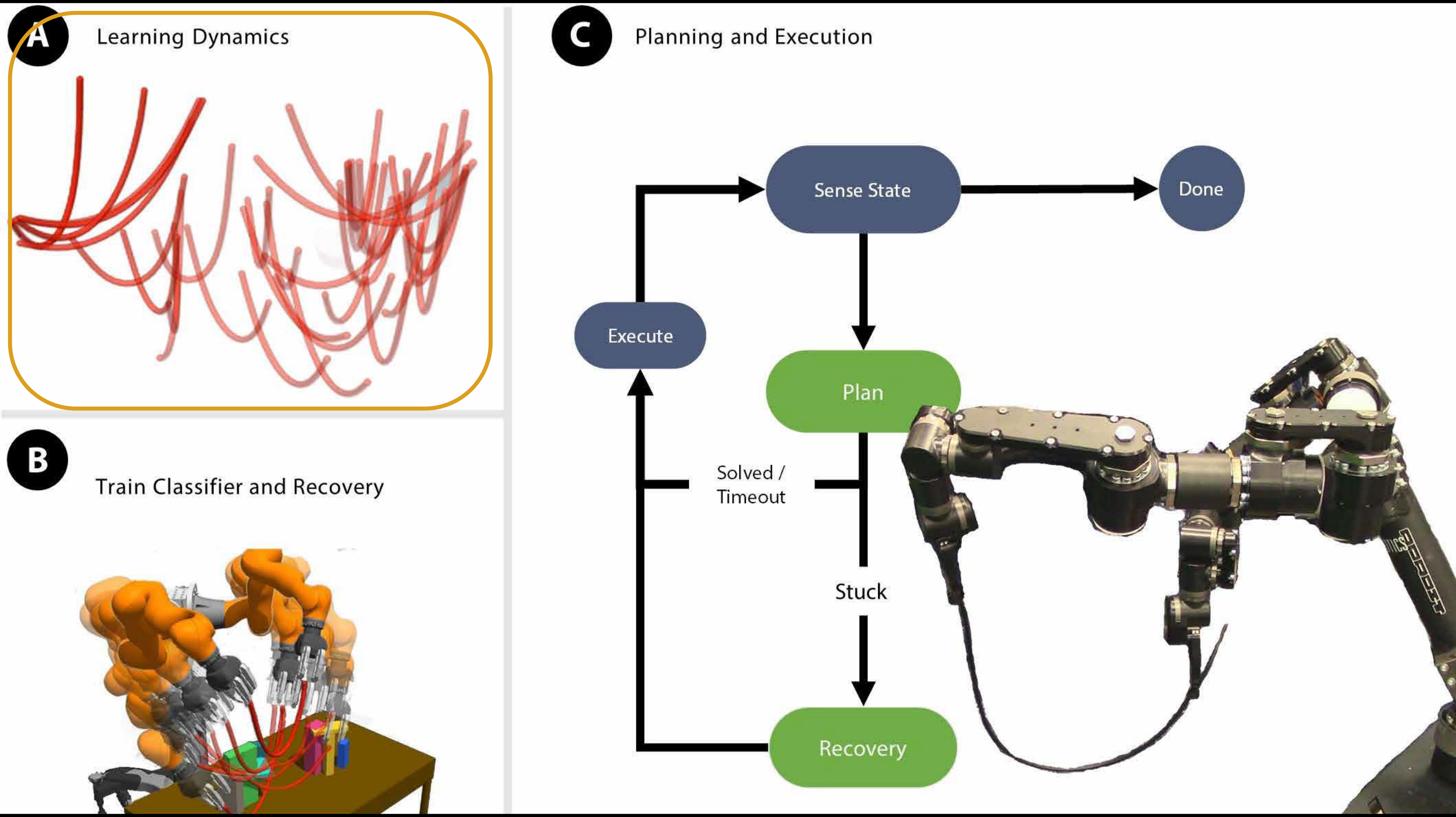
Peter Mitrano
PhD Student

Dale McConachie
PhD Student
Now at TRI

Key Questions

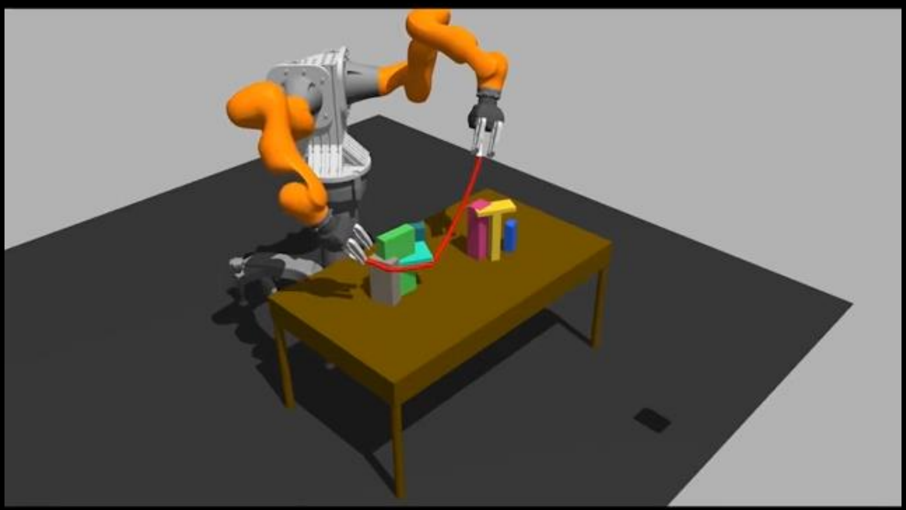
- How do we learn a dynamics model for complex deformable manipulation tasks?
  - Learn dynamics model in unconstrained environment
- When should we trust the model?
  - Learn classifier to predict when that model is reliable in constrained environments
- What do we do if we are in a state where our model is unreliable?
  - Use Recovery to move toward a state where the model *is* reliable

[Mitrano et al., Science Robotics, May 2021]

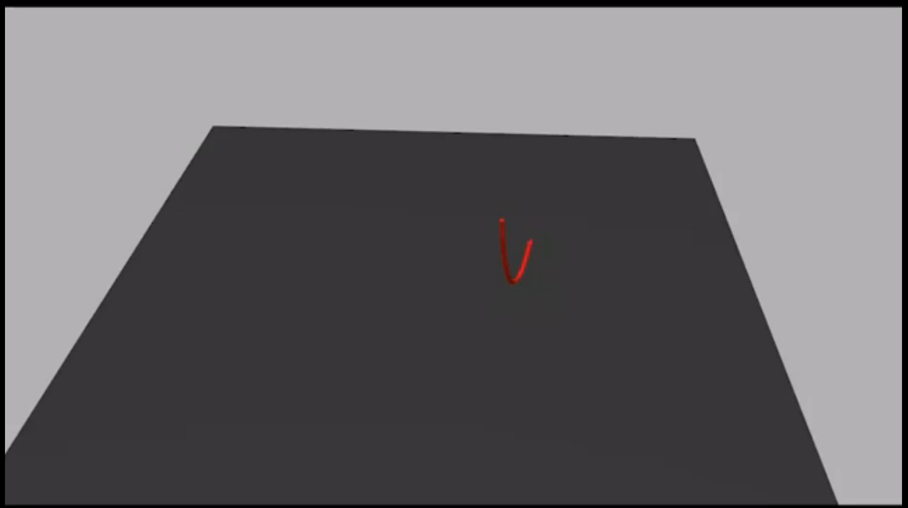


A
Learning Dynamics
B
Train Classifier and Recovery
C
Planning and Execution
Sense State
Done
Execute
Plan
Solved /
Timeout
Stuck
Recovery

Learning Dynamics: Example



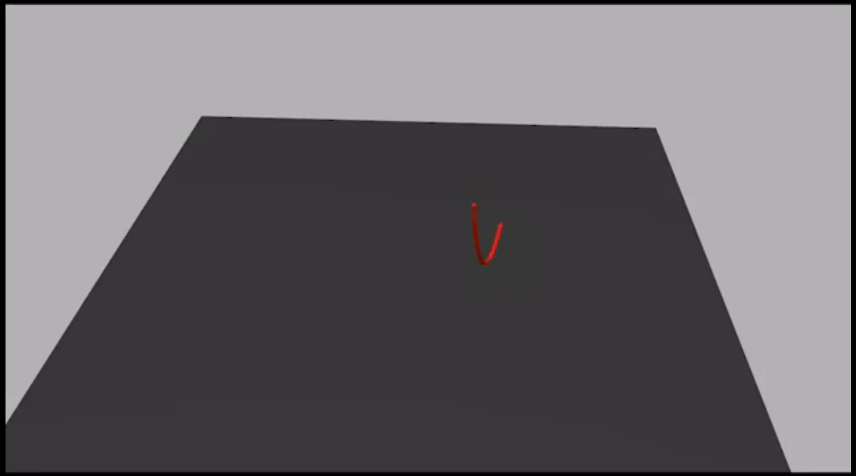
Learning these dynamics is very difficult



Learning the dynamics is tractable

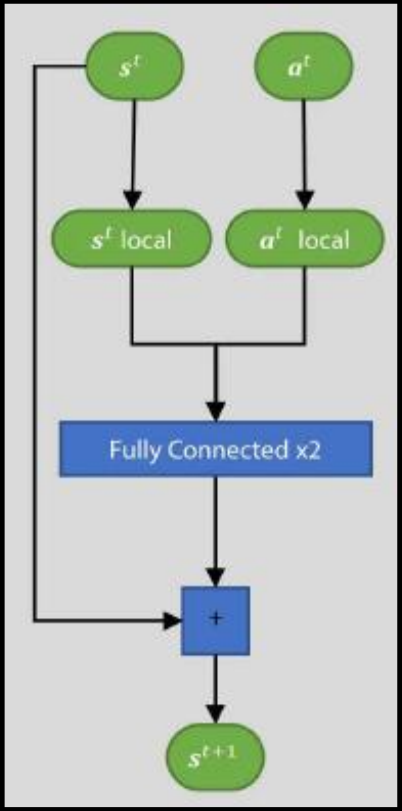
Learning Dynamics

1. Collect data by rolling out random trajectories in *unconstrained* environment

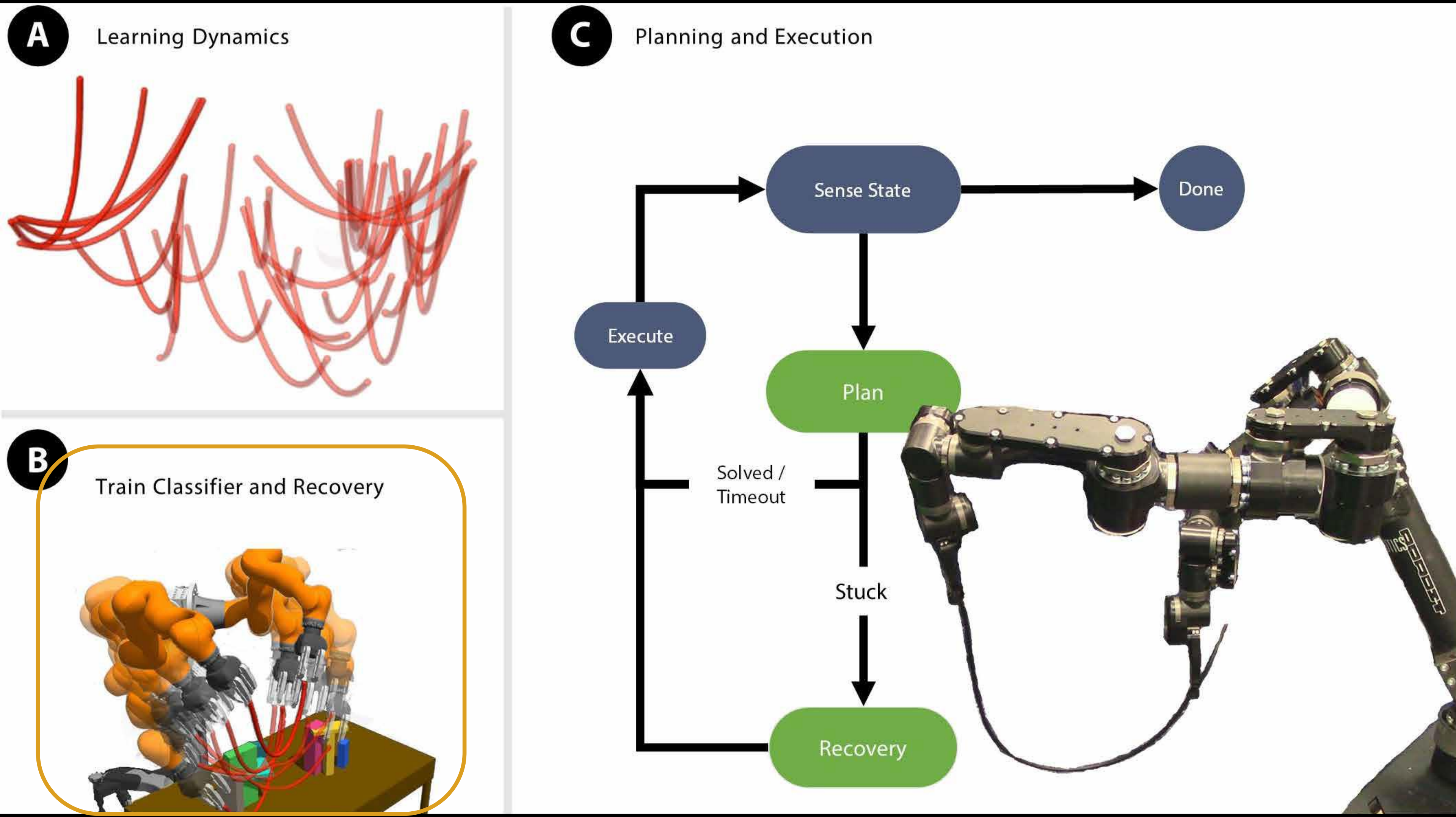


2. Learn $s^{t+1} = s^t + f(s^t, a^t)$

state action



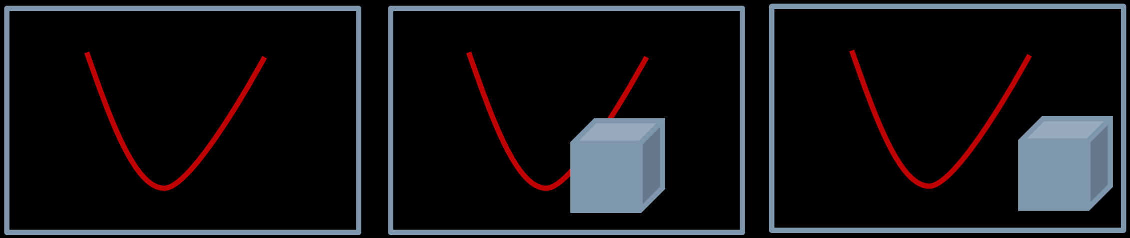
next state


A
Learning Dynamics
B
Train Classifier and Recovery
C
Planning and Execution
Sense State
Done
Execute
Plan
Solved /
Timeout
Stuck
Recovery

Why not use an ensemble to predict when the learned model is valid in the constrained scenario?

- Neural networks can have high epistemic uncertainty even when there are *irrelevant* differences between training and new data

Example:

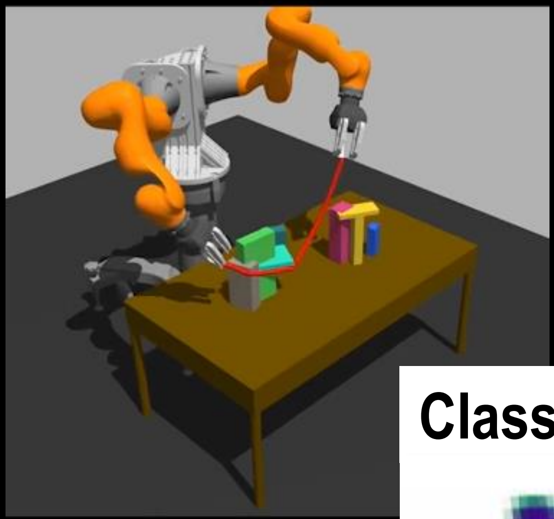


Ensemble would say:	Model is **reliable**	Model is **unreliable**	Model is **unreliable**
Correct?	✓	✓	✗

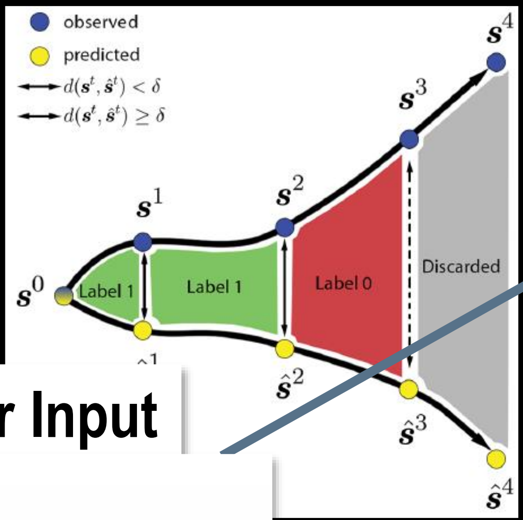
- I.e. the ensemble would say the dynamics are *always* unreliable for non-trivial problems ☹
- Ensembles are a general way to estimate epistemic uncertainty, but too conservative
  - We can do better with methods that use knowledge of the structure of the problem

Estimating Dynamics Reliability by Learning a Classifier

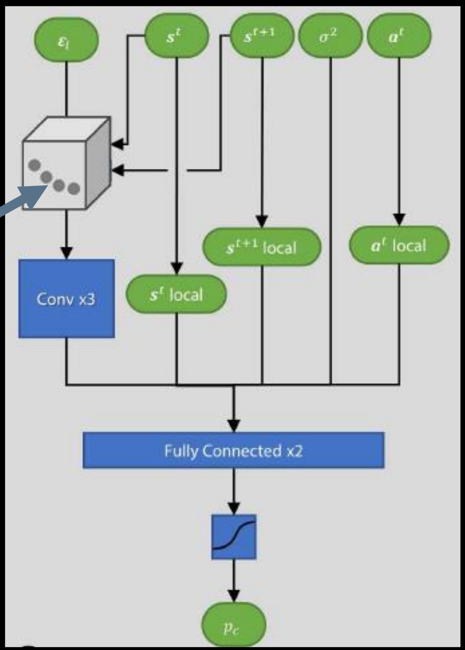
1. Collect another dataset in environments with constraints



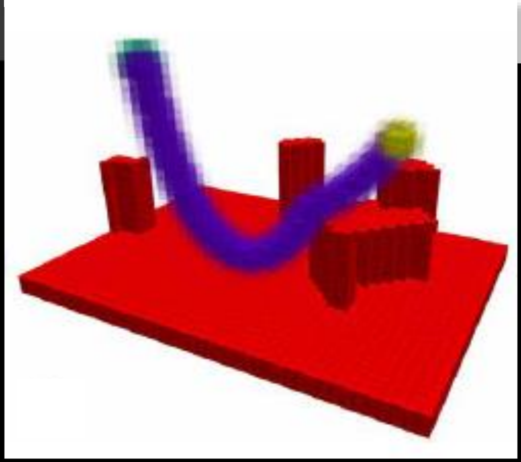
2. Label the data w.r.t to dynamics accuracy



3. Learn classifier which predicts model reliability for a given s^t, a^t

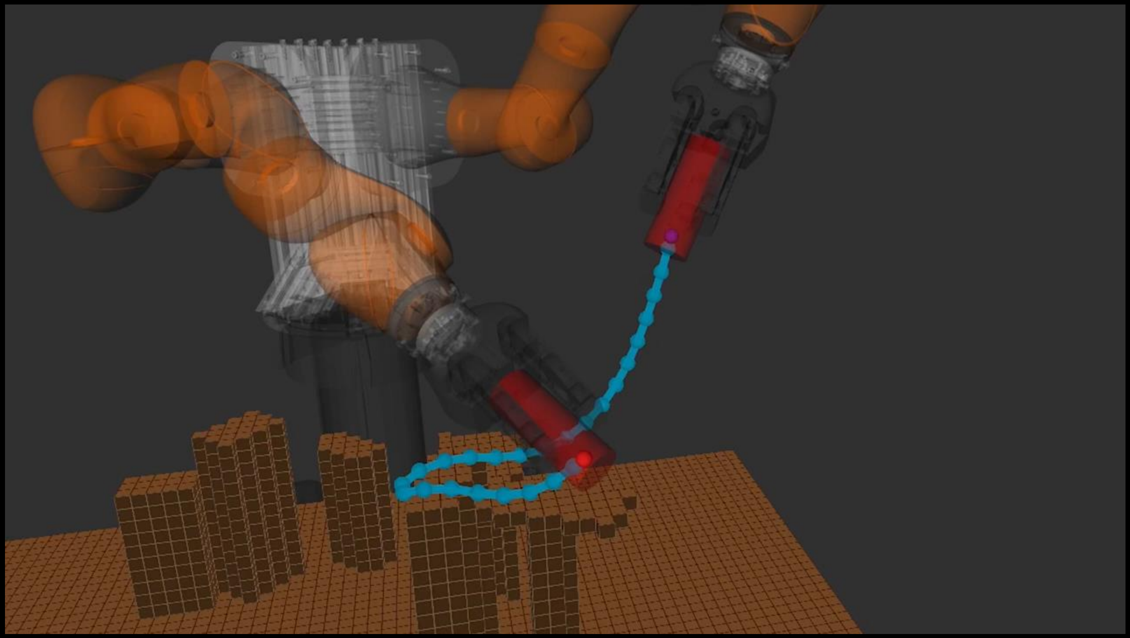


Classifier Input



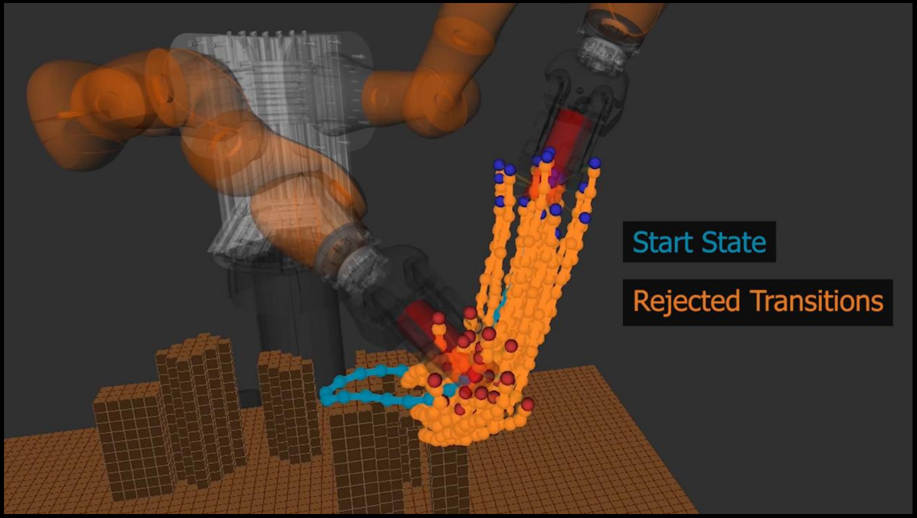
Recovery

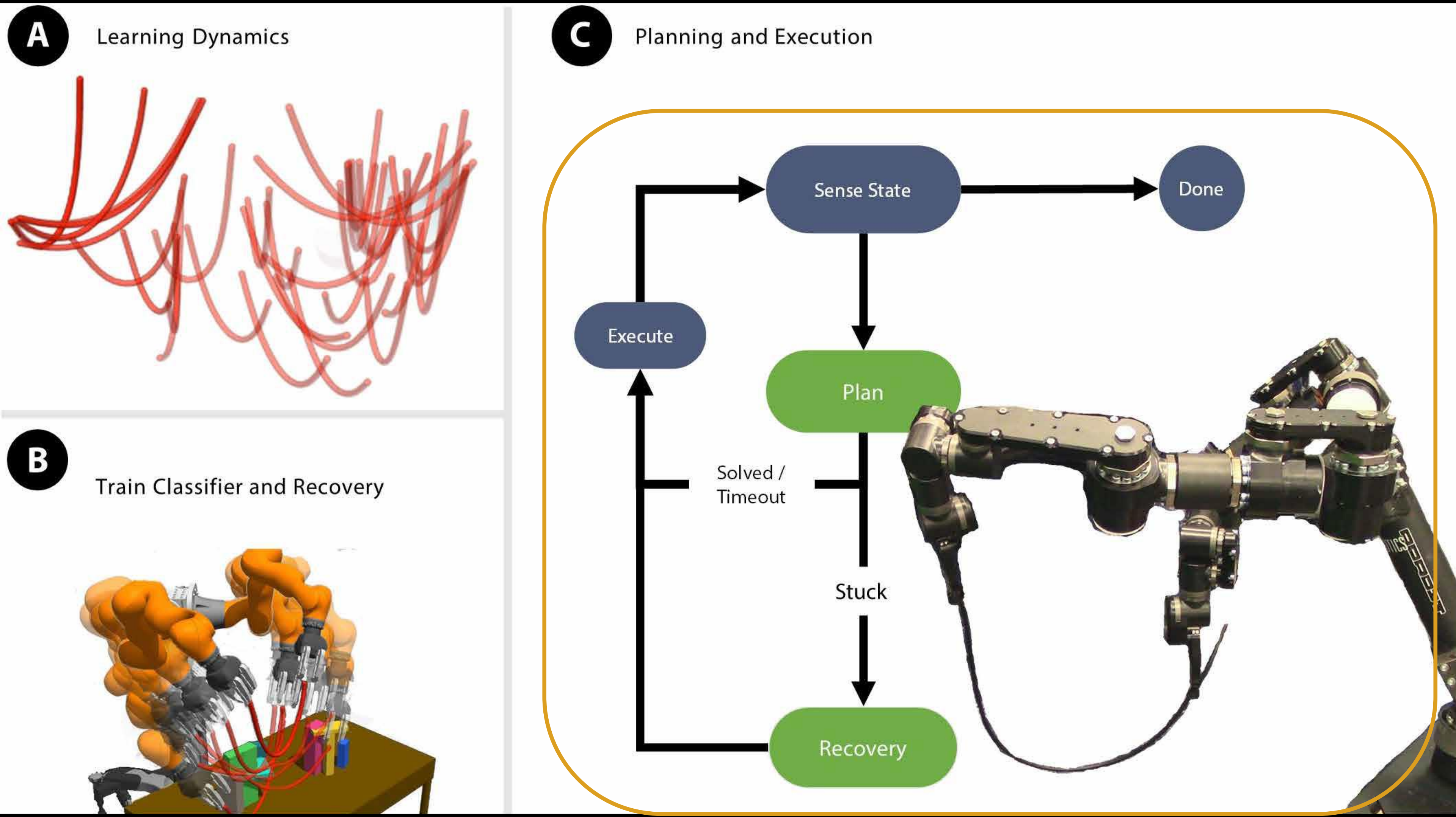
- How to detect when we are stuck (e.g. the object starts in a state where the model is not reliable)?
  - Sample over possible actions and check classifier output
  - If classifier rejects all actions, need to recover



Learning to Recover

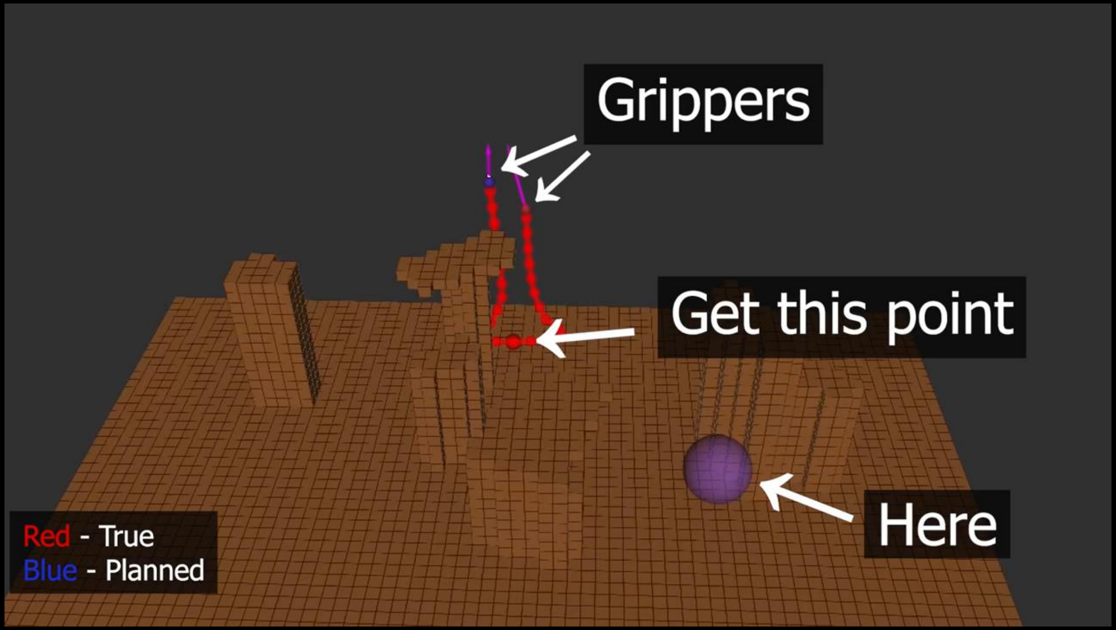
- How do we move toward a state where the model *can be trusted*?
  - Use dataset from environments with constraints to find states where *classifier predicts 0* for many actions
  - Then look for transitions from these states that yield states where the *classifier predicts 1*
- Learn to predict probability of recovery for a given s^t, a^t (another neural net)
- Use recovery network to select the best action online when stuck




A
Learning Dynamics
B
Train Classifier and Recovery
C
Planning and Execution
Sense State
Done
Execute
Plan
Solved /
Timeout
Stuck
Recovery

Planning and execution

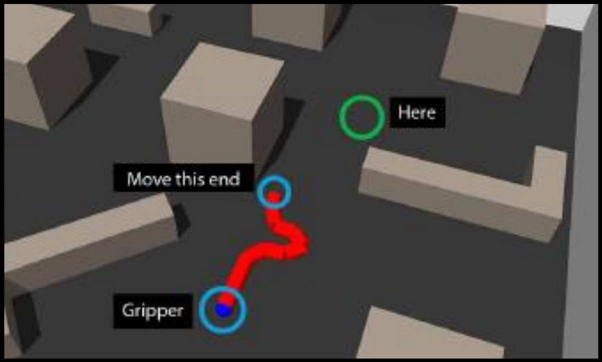
- Use Kinodynamic RRT planner with learned dynamics
- Use classifier to reject transitions where the dynamics are unreliable

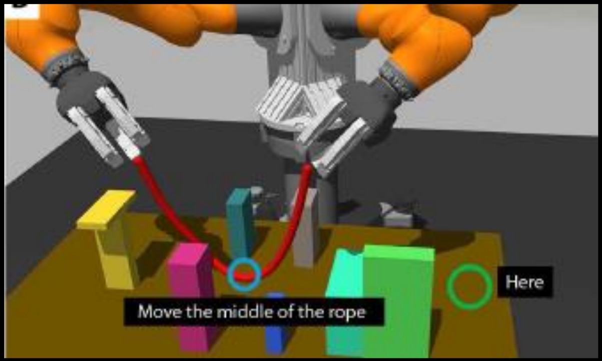


- If stuck, recover and replan until time limit

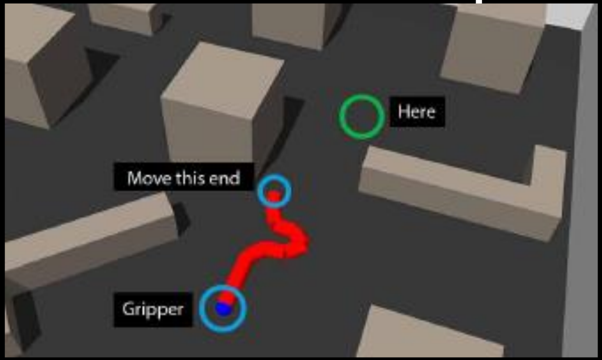
Experiments

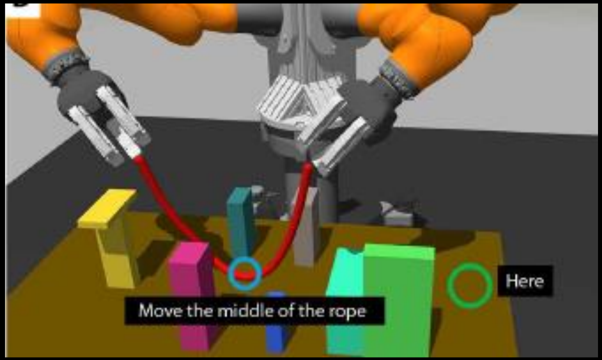
- Tested on random environments for
  - Rope dragging tasks
  - Dual-arm rope manipulation
- Compared in simulation to baselines
  - Learning Full Dynamics (same data)
  - Ablations of our method
- Metric
  - Distance to goal after 3min of planning+execution maximum
    - Being within 0.05m of the goal counts as "success"

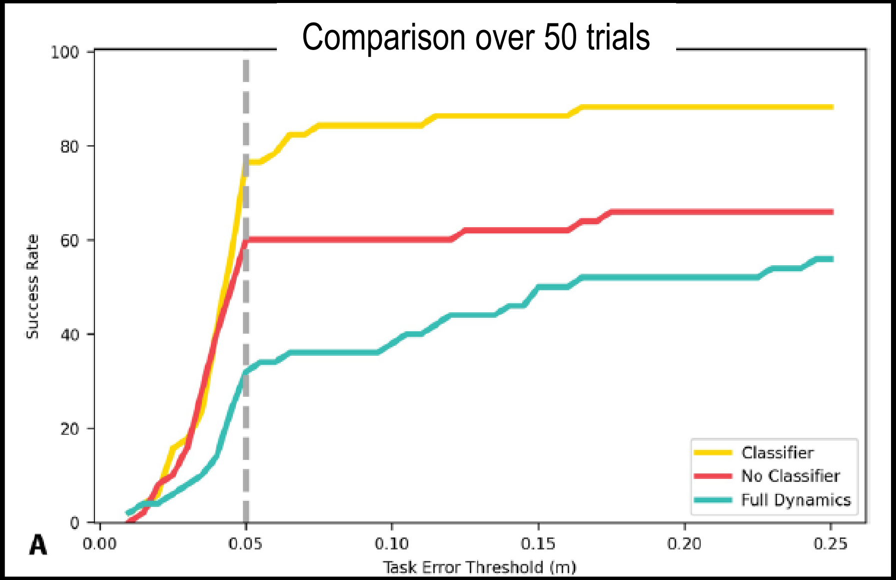


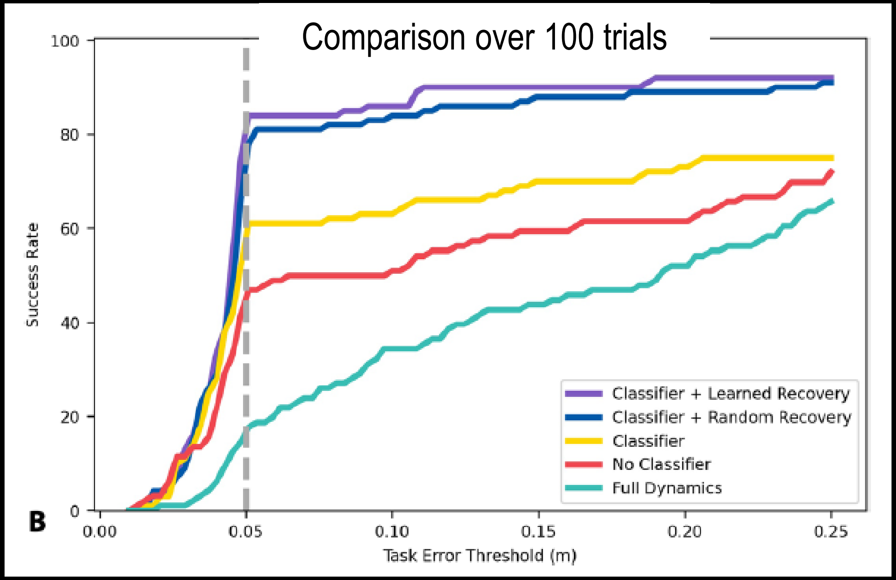


Results of Comparison









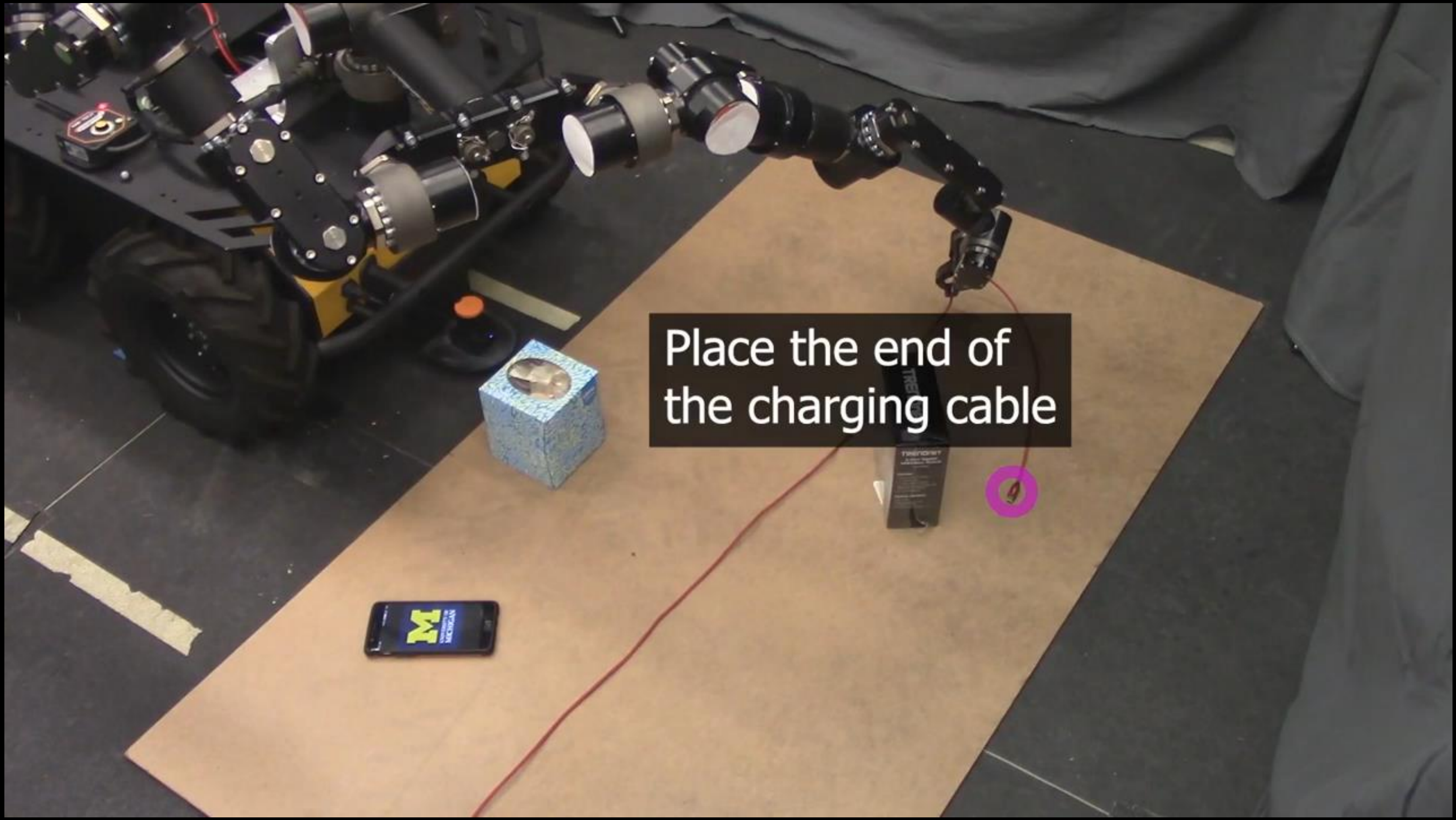
Using the classifier improved success rate by about 20% for rope dragging

Classifier+learned recovery was best for dual arm rope, though randomized recovery did nearly as well

Our approach vastly outperforms learning Full Dynamics (using same data)

Examples

- Planned in simulation, executed in the real world



Summary

- We can learn a function that approximates the dynamics of system using a neural network
- Need to consider two kinds of uncertainty:
  - Aleatoric: Inherent randomness (e.g. a stochastic system)
    - Use networks that output predictive distributions
  - Epistemic: Uncertainty due to lack of data
    - In general, use ensembles
- Can use knowledge of structure of the problem to reason about dynamics uncertainty for deformable object manipulation
  - Learn dynamics in a simplified setting
  - Learn classifier to predict where the learned dynamics can be trusted
  - Use the dynamics and classifier in motion planning

Homework

- Final project!