

Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

# Imitation Learning

Spring 2020, CMU 10-403

Katerina Fragkiadaki

# Limitations of Learning by Interaction

- The agent should have the chance to try (and fail) MANY times
- This is hard when safety is a concern: we cannot afford to fail
- This is also quite hard in general in real life where each interaction takes time (in contrast to simulation)



Crusher robot

*Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain, Silver et al. 2010*

# Imitation Learning (a.k.a. Learning from Demonstrations)

## visual imitation



The actions of the teacher need to be inferred from visual sensory input and mapped to the action space of the agent.

Two challenges:

- 1) visual understanding
- 2) action mapping, especially when the agent and the teacher do not have the same action space

(later lecture)

## kinesthetic imitation



- The teacher takes over the end-effectors of the agent.
- Demonstrated actions are in the action space of the imitator and can be imitated directly)

this lecture

# Notation



Richard Bellman



Lev Pontryagin

actions  $a_t$

states  $s_t$

rewards  $r_t$

dynamics  $p(s_{t+1} | s_t, a_t)$

observations  $o_t$

actions  $u_t$

states  $x_t$

costs  $c(x_t, u_t)$

dynamics  $p(x_{t+1} | x_t, u_t)$

# Imitation learning VS Sequence labelling

## Imitation learning



Training data:

$$o_1^1, u_1^1, o_2^1, u_2^1, o_3^1, u_3^1, \dots$$

$$o_1^2, u_1^2, o_2^2, u_2^2, o_3^2, u_3^2, \dots$$

$$o_1^3, u_1^3, o_2^3, u_2^3, o_3^3, u_3^3, \dots$$

$\mathbf{u}_t$  :the action at time t

$\mathbf{o}_t$  :the observation at time t

$\mathbf{x}_t$  :the state at time t

## Sequence labelling



Training data:

$$o_1^1, y_1^1, o_2^1, y_2^1, o_3^1, y_3^1, \dots$$

$$o_1^2, y_1^2, o_2^2, y_2^2, o_3^2, y_3^2, \dots$$

$$o_1^3, y_1^3, o_2^3, y_2^3, o_3^3, y_3^3, \dots$$

$y_1$

$y_2$

$y_3$

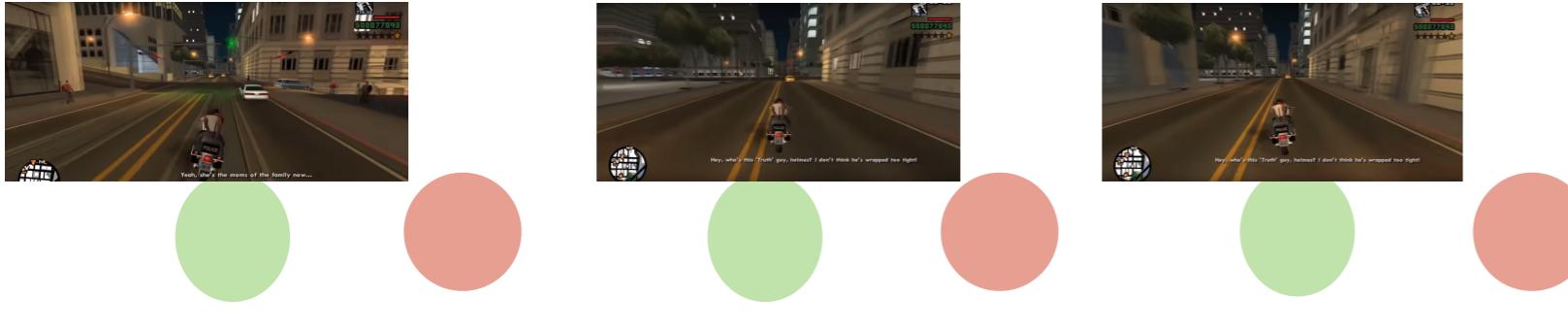
$y_t$  : which product was purchased at frame t (if any)

$\mathbf{o}_t$  :the observation at time t

$\mathbf{x}_t$  :the state at time t

# Imitation learning VS Sequence labelling

## Imitation learning



Training data:

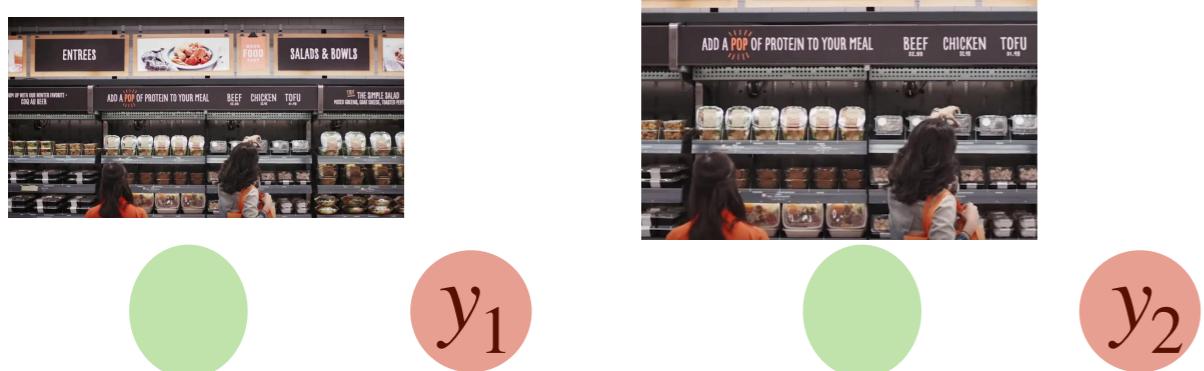
$$\begin{aligned} o_1^1, u_1^1, o_2^1, u_2^1, o_3^1, u_3^1, \dots \\ o_1^2, u_1^2, o_2^2, u_2^2, o_3^2, u_3^2, \dots \\ o_1^3, u_1^3, o_2^3, u_2^3, o_3^3, u_3^3, \dots \end{aligned}$$

$\mathbf{u}_t$  :the action at time t

$\mathbf{o}_t$  :the observation at time t

$\mathbf{x}_t$  :the state at time t

## Sequence labelling



Training data:

$$\begin{aligned} o_1^1, y_1^1, o_2^1, y_2^1, o_3^1, y_3^1, \dots \\ o_1^2, y_1^2, o_2^2, y_2^2, o_3^2, y_3^2, \dots \\ o_1^3, y_1^3, o_2^3, y_2^3, o_3^3, y_3^3, \dots \end{aligned}$$

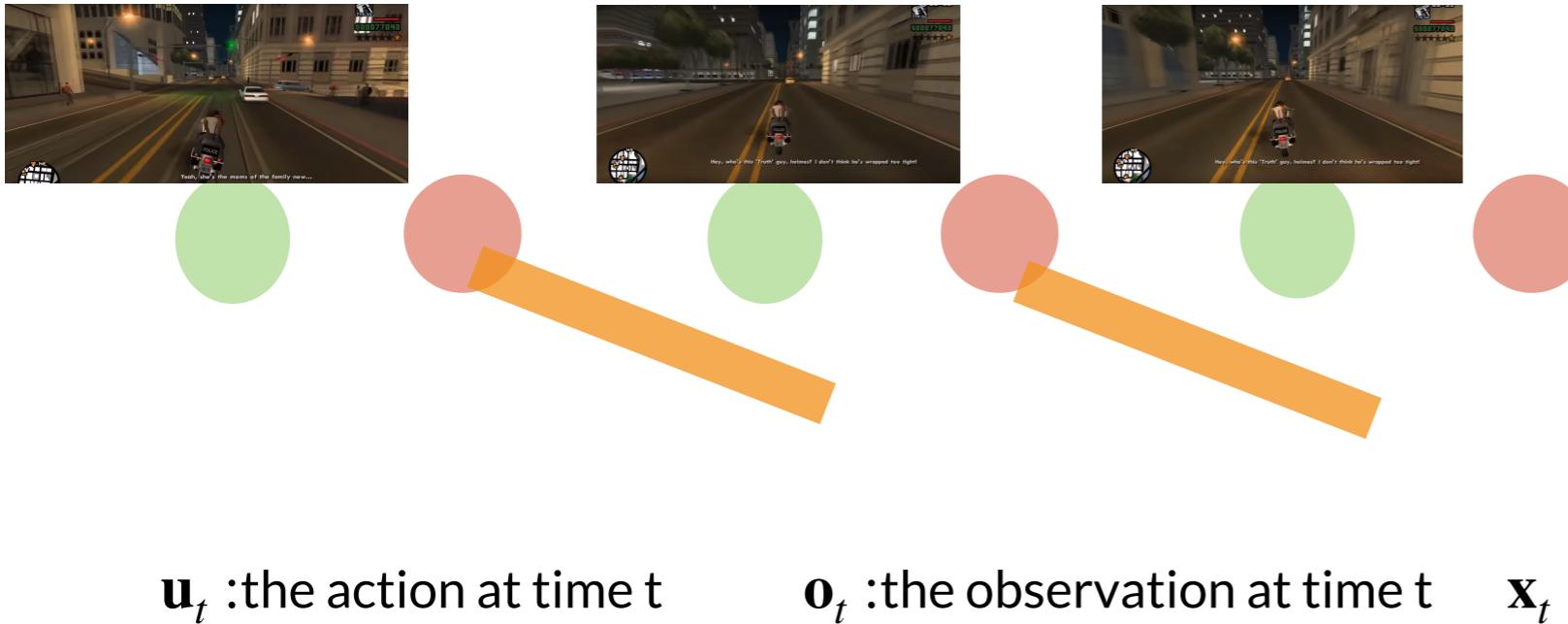
$y_t$  : which product was purchased at frame t (if any)

$\mathbf{o}_t$  :the observation at time t

$\mathbf{x}_t$  :the state at time t

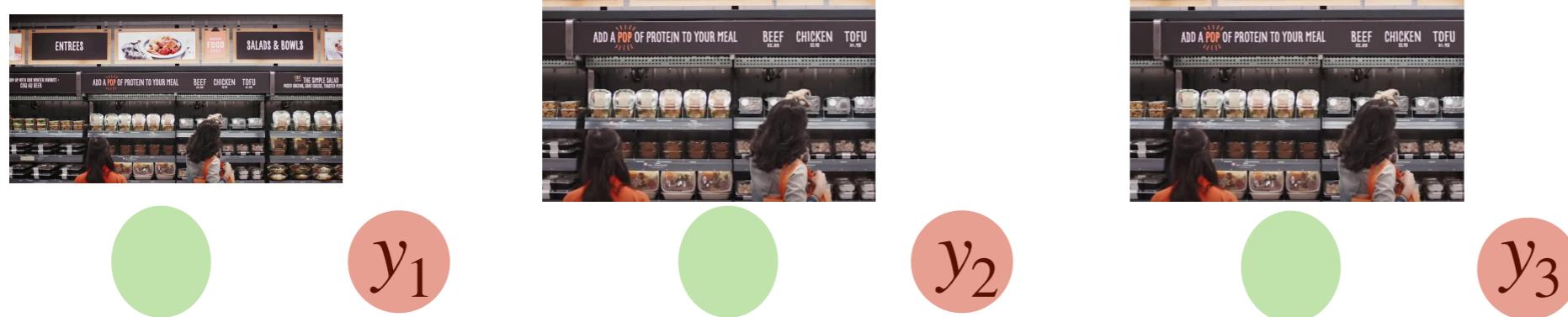
# Imitation learning VS Sequence labelling

## Imitation learning



- In RL, our actions will influence our future state, and thus our future data.
- In sequence labelling, our labels won't influence the future frames.

## Sequence labelling



Training data:

$$\begin{aligned} & o_1^1, y_1^1, o_2^1, y_2^1, o_3^1, y_3^1, \dots \\ & o_1^2, y_1^2, o_2^2, y_2^2, o_3^2, y_3^2, \dots \\ & o_1^3, y_1^3, o_2^3, y_2^3, o_3^3, y_3^3, \dots \end{aligned}$$

# Video sequence labelling

Action labelling: a mapping from states/observations to action labels



***“picking up chicken A”***

$$f_{\theta}(y_t | \mathbf{o}_t)$$

- Assume action labels in an annotated video are i.i.d. (independent and identically distributed).
- Train a classifier to map observations to labels at each time step of the trajectory



$\mathbf{o}_t$

$y_t$

$$f_{\theta}(y_t | \mathbf{o}_t)$$

# Imitation Learning

Policy: a mapping from observations to actions

- Assume actions in the expert trajectories are i.i.d. (independent and identically distributed)
- Train a function to map observations/states to actions at each time step of the trajectory

# What can go wrong?

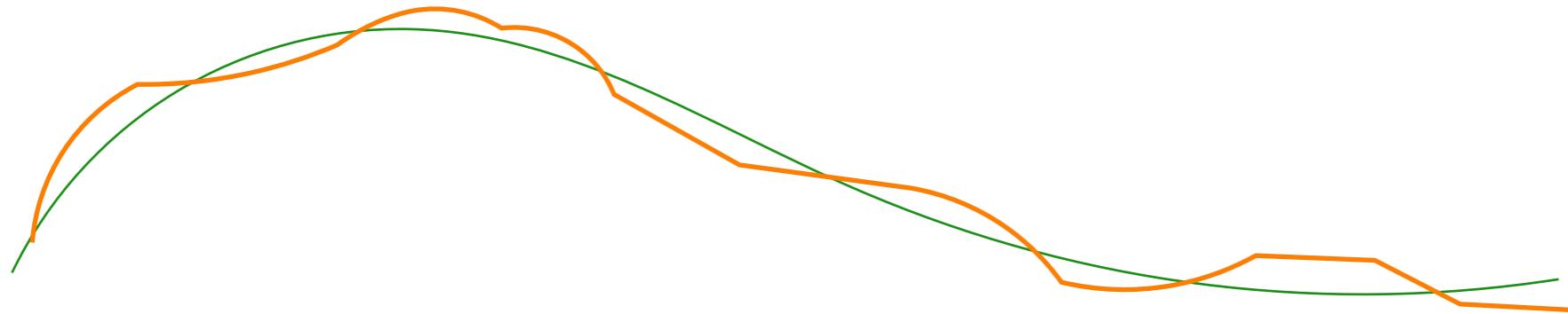
- Compounding errors  
Fix: data augmentation
- Non-markovian observations  
Fix: observation concatenation or recurrent models
- Stochastic expert actions  
Fix: generative modelling (GANs, stochastic latent variable models, action discretisation, gaussian mixture networks)

# What can go wrong?

- Compounding errors  
Fix: data augmentation
- Non-markovian observations  
Fix: observation concatenation or recurrent models
- Stochastic expert actions  
Fix: generative modelling (GANs, stochastic latent variable models, action discretisation, gaussian mixture networks)

# Independent in time errors

This means that at each time step  $t$ , the agent wakes up on a state drawn from the state distribution of the expert trajectories, and executes an action.

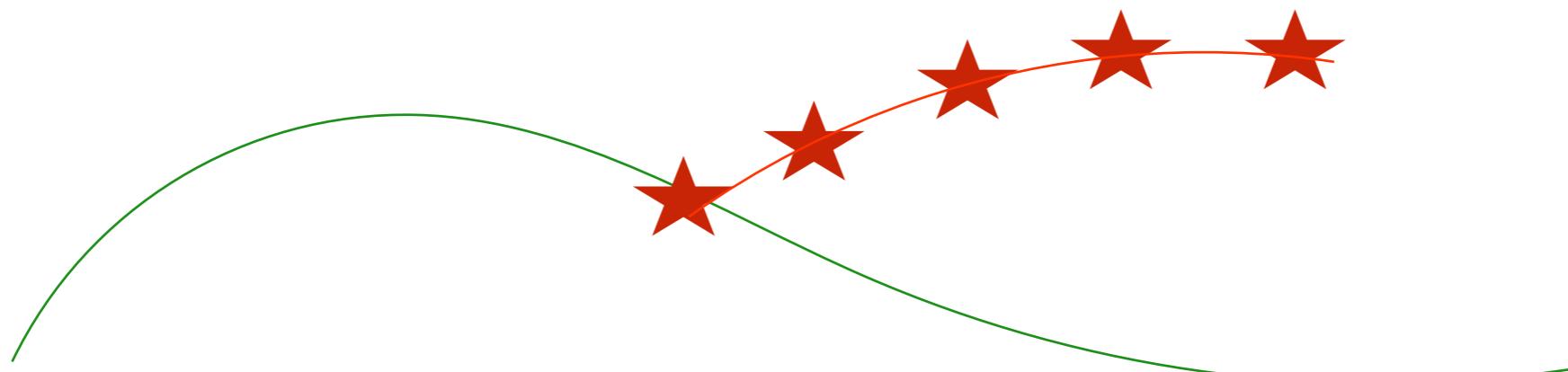


error at time  $t$  with probability  $\varepsilon$

$E[\text{Total errors}] \leq \varepsilon T$ ,  $T$  the length of the trajectory

# Compounding Errors

This means that at each time step  $t$ , the agent wakes up on a state drawn from the state distribution resulting from executing the action the learned policy suggested in the previous time step.

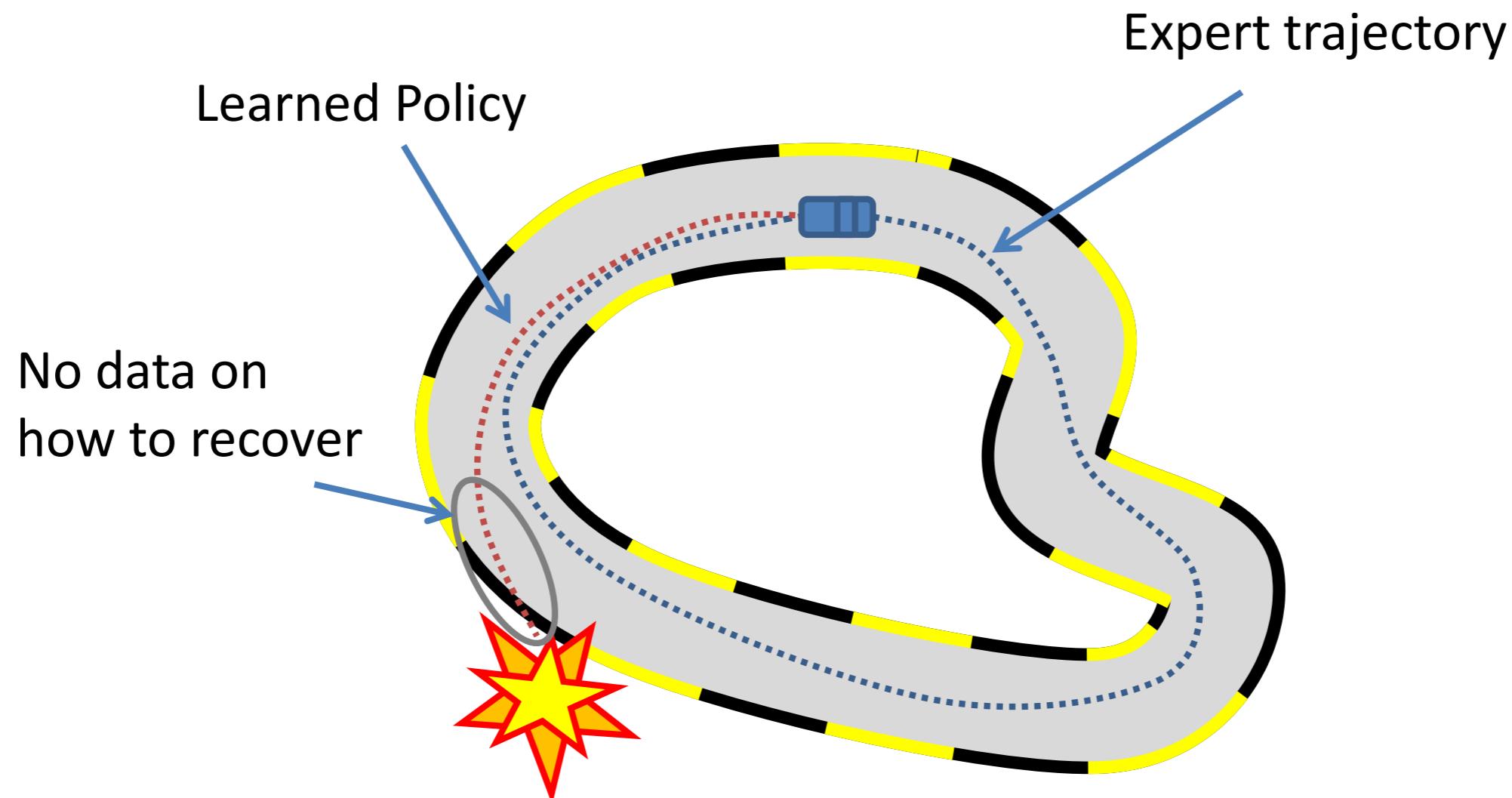


error at time  $t$  with probability  $\varepsilon$

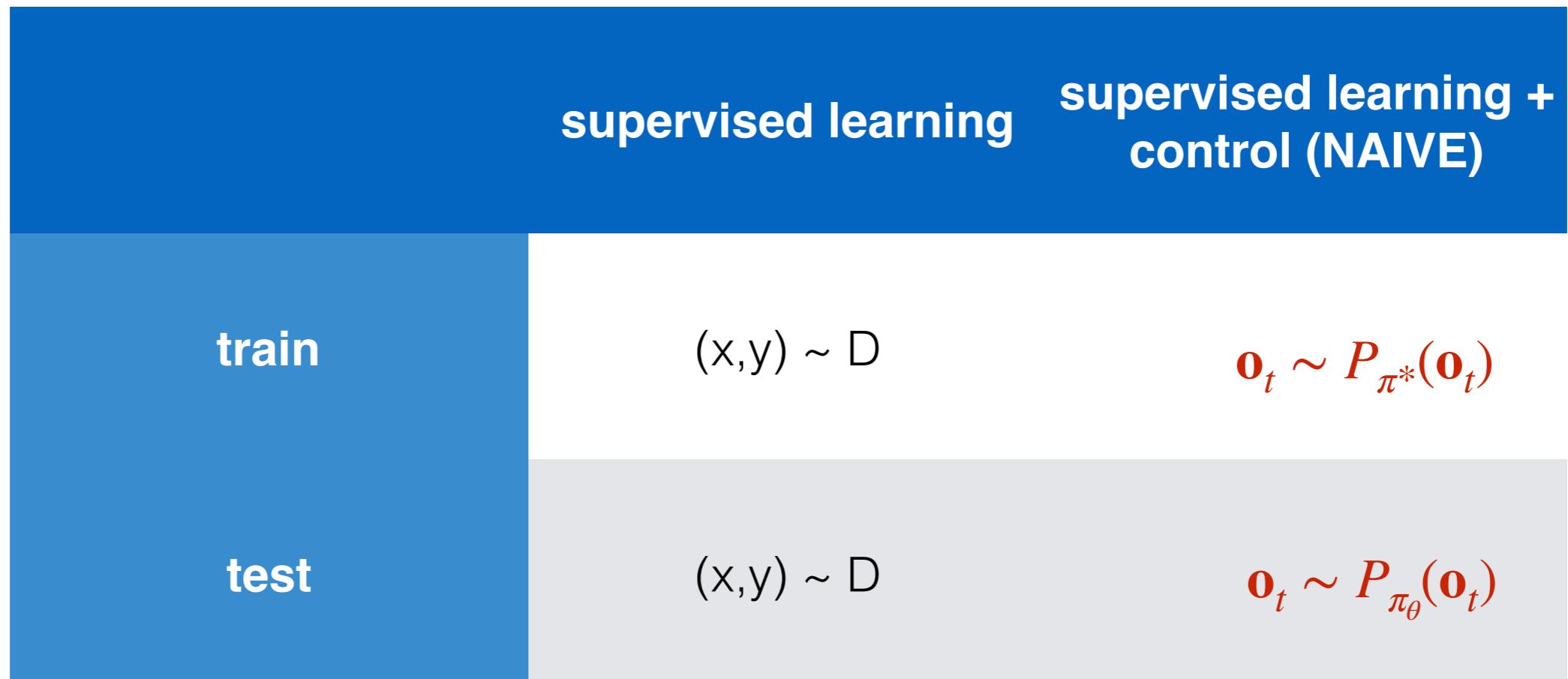
$$\mathbb{E}[\text{Total errors}] \lesssim \varepsilon(T + (T-1) + (T-2) + \dots + 1) \propto \varepsilon T^2$$

# Distribution mismatch (distribution shift)

$$P_{\pi^*}(\mathbf{o}_t) \neq P_{\pi_\theta}(\mathbf{o}_t)$$



# Distribution mismatch (distribution shift)



Supervised learning succeeds when training and test data distributions match, that is a fundamental assumption.

# Solution: data augmentations

Change  $P_{\pi^*}(\mathbf{o}_t)$  by **augmenting** the expert demonstration trajectories.

This means: add examples in expert demonstration trajectories to cover the states/observations points where the agent will land when trying out its own policy. How?

1. By generating synthetic data in simulation
2. By collecting additional data via clever hardware
3. By interactively querying the experts in additional datapoints

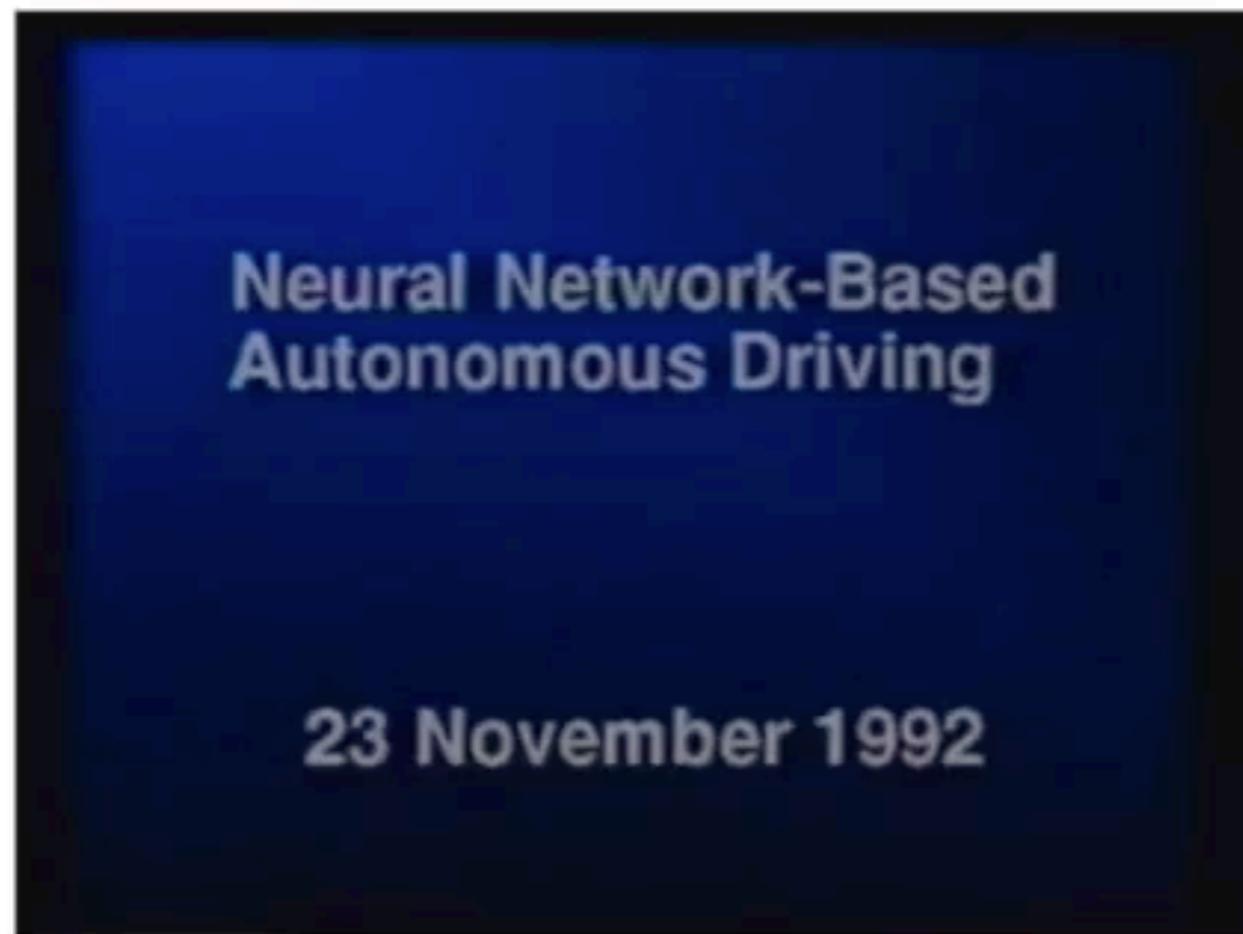
# Solution: data augmentations

Change  $P_{\pi^*}(\mathbf{o}_t)$  by **augmenting** the expert demonstration trajectories.

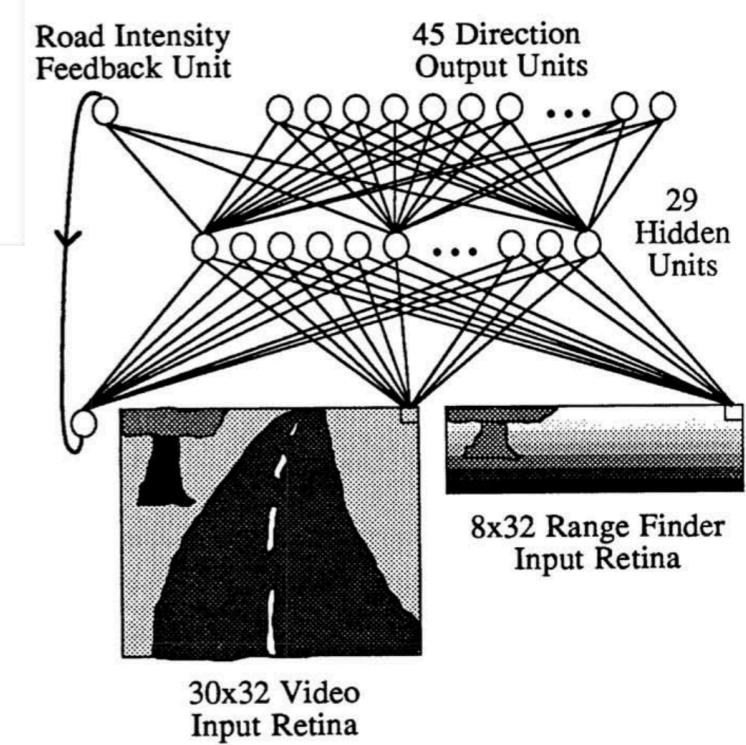
Add examples in expert demonstration trajectories to cover the states/observations points where the agent will land when trying out its own policy. How?

1. By generating synthetic data in simulation
2. By collecting additional data via clever hardware
3. By interactively querying the experts in additional datapoints

# Demonstration Augmentation: ALVINN 1989



[Courtesy of Dean Pomerleau]

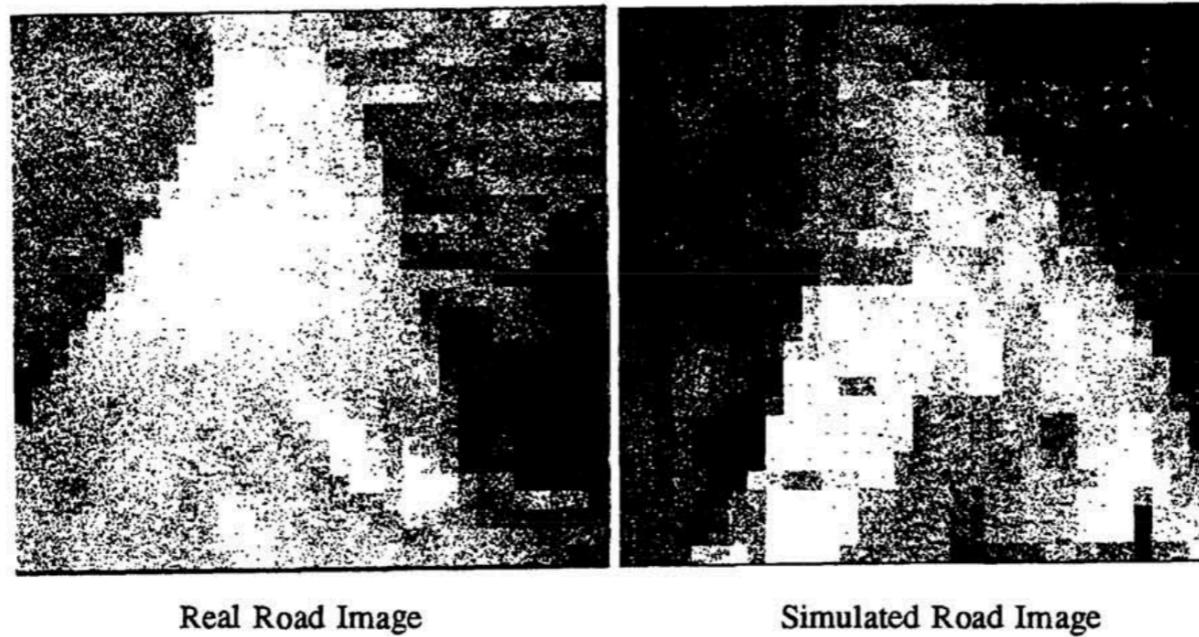


# Demonstration Augmentation: ALVINN 1989

“In addition, the network must not solely be shown examples of accurate driving, but also how to recover (i.e. return to the road center) once a mistake has been made. Partial initial training on a variety of simulated road images should help eliminate these difficulties and facilitate better performance.”

ALVINN: An autonomous Land vehicle in a neural Network”,  
Pomerleau 1989

# Demonstration Augmentation: ALVINN 1989



- Use of image simulator to generate images of how the road looks like when the vehicle deviates slightly from its trajectory.
- Simulating the images too longer than training the network

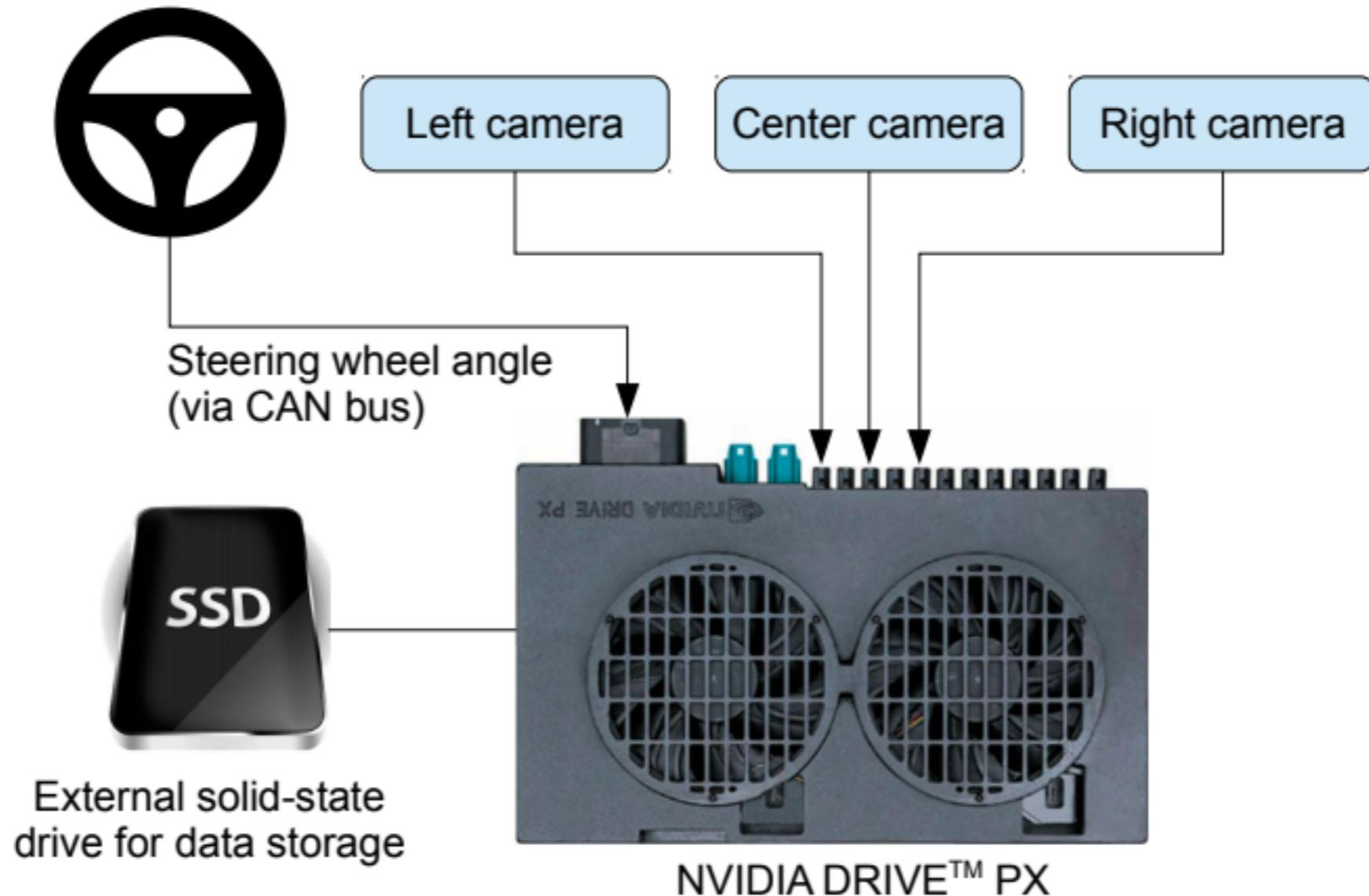
# Solution: data augmentations

Change  $P_{\pi^*}(\mathbf{o}_t)$  by **augmenting** the expert demonstration trajectories.

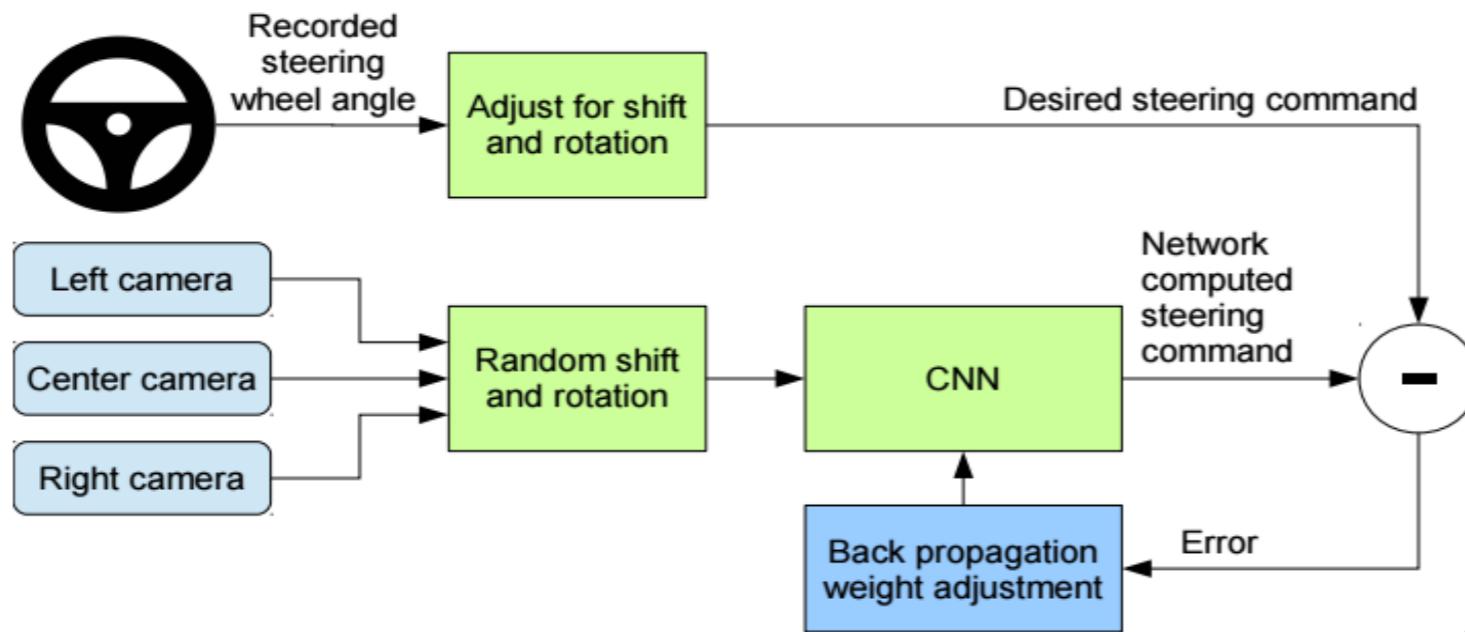
Add examples in expert demonstration trajectories to cover the states/observations points where the agent will land when trying out its own policy. How?

1. By generating synthetic data in simulation
2. By collecting additional data via clever hardware
3. By interactively querying the experts in additional datapoints

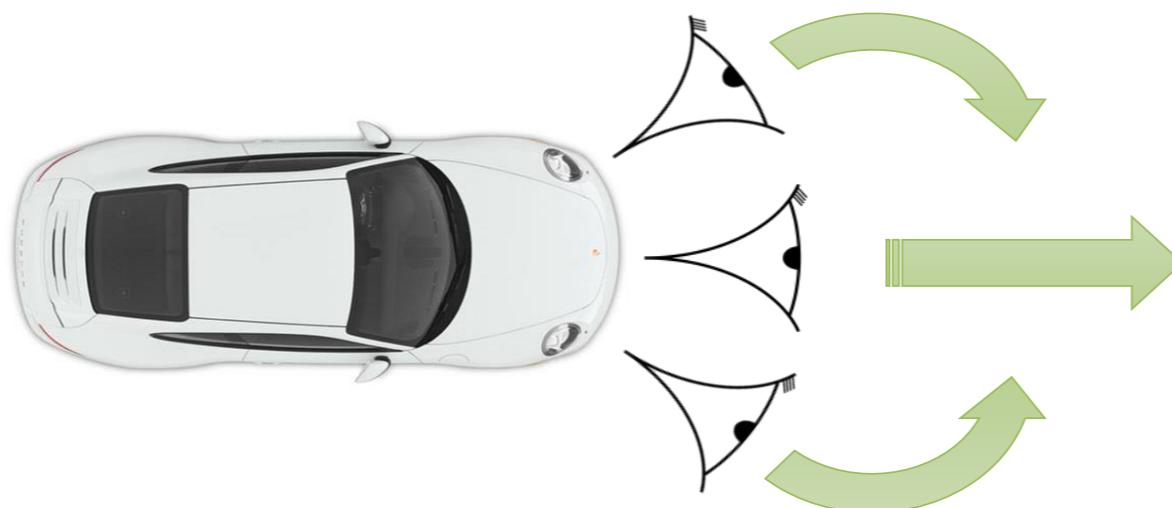
# Learning to Drive a Car: Supervised Learning



# Demonstration Augmentation: NVIDIA 2016



Additional, left and right cameras with automatic grant-truth labels to recover from mistakes



“DAVE-2 was inspired by the pioneering work of Pomerleau [6] who in 1989 built the Autonomous Land Vehicle in a Neural Network (ALVINN) system. Training with data from only the human driver is not sufficient. The network must learn how to recover from mistakes. ....”

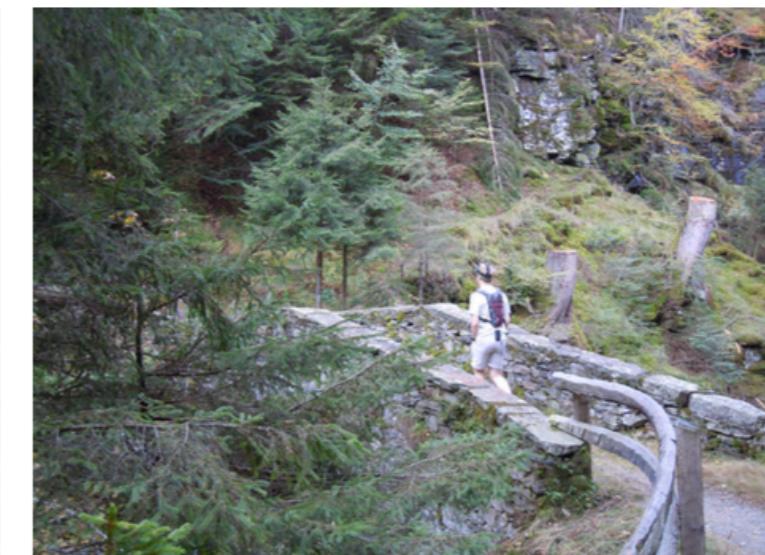
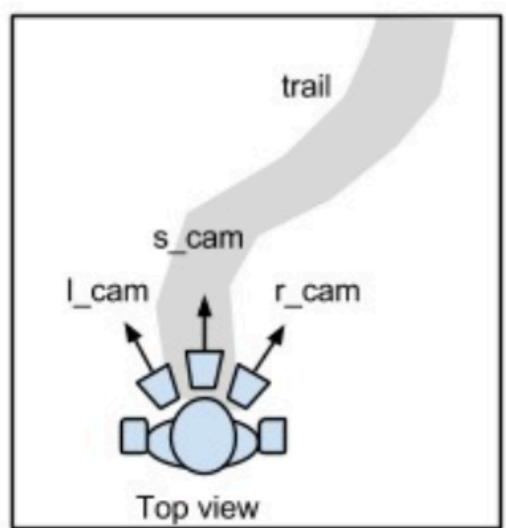
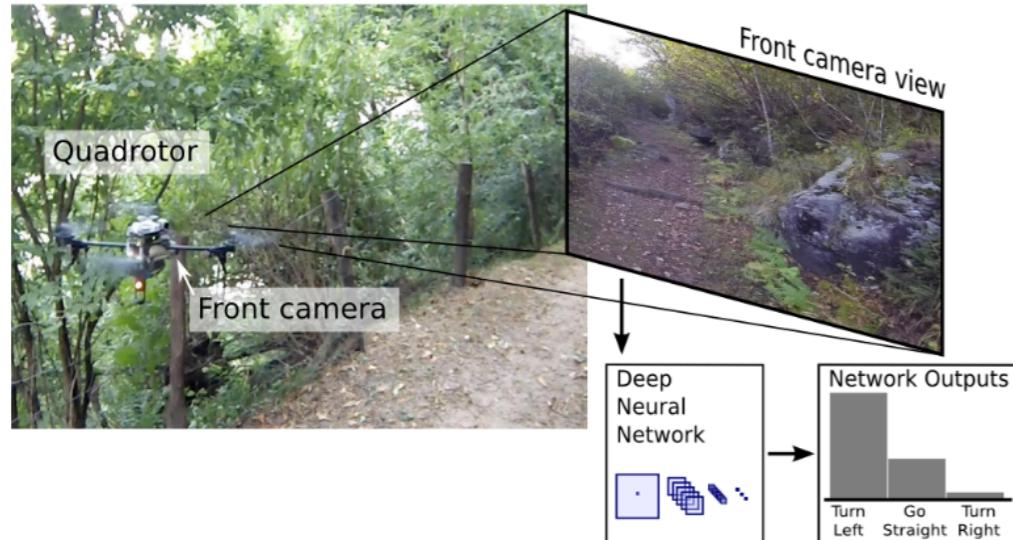
# Data Augmentation (2): NVIDIA 2016

## DAVE 2 Driving a Lincoln

- A convolutional neural network
- Trained by human drivers
- Learns perception, path planning, and control  
"pixel in, action out"
- Front-facing camera is the only sensor

"DAVE-2 was inspired by the pioneering work of Pomerleau [6] who in 1989 built the Autonomous Land Vehicle in a Neural Network (ALVINN) system. Training with data from only the human driver is not sufficient. The network must learn how to recover from mistakes. . . .", End to End Learning for Self-Driving Cars , Bojarski et al. 2016

# Data Augmentation (3): Trails 2015



# Data Augmentation (3): Trails 2015

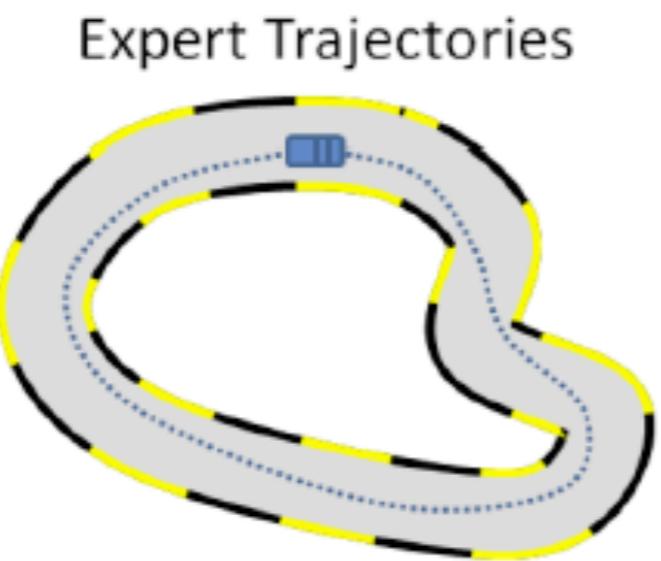
# Solution: data augmentations

Change  $P_{\pi^*}(\mathbf{o}_t)$  by **augmenting** the expert demonstration trajectories.

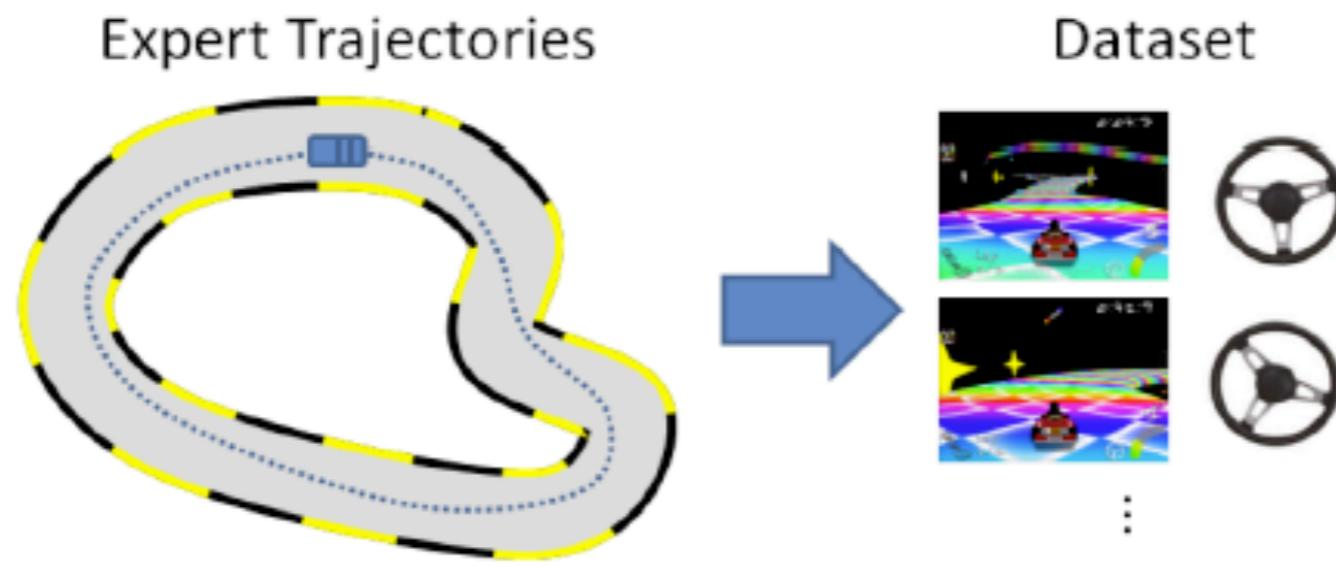
Add examples in expert demonstration trajectories to cover the states/observations points where the agent will land when trying out its own policy. How?

1. By generating synthetic data in simulation
2. By collecting additional data via clever hardware
3. By interactively querying the experts in additional datapoints

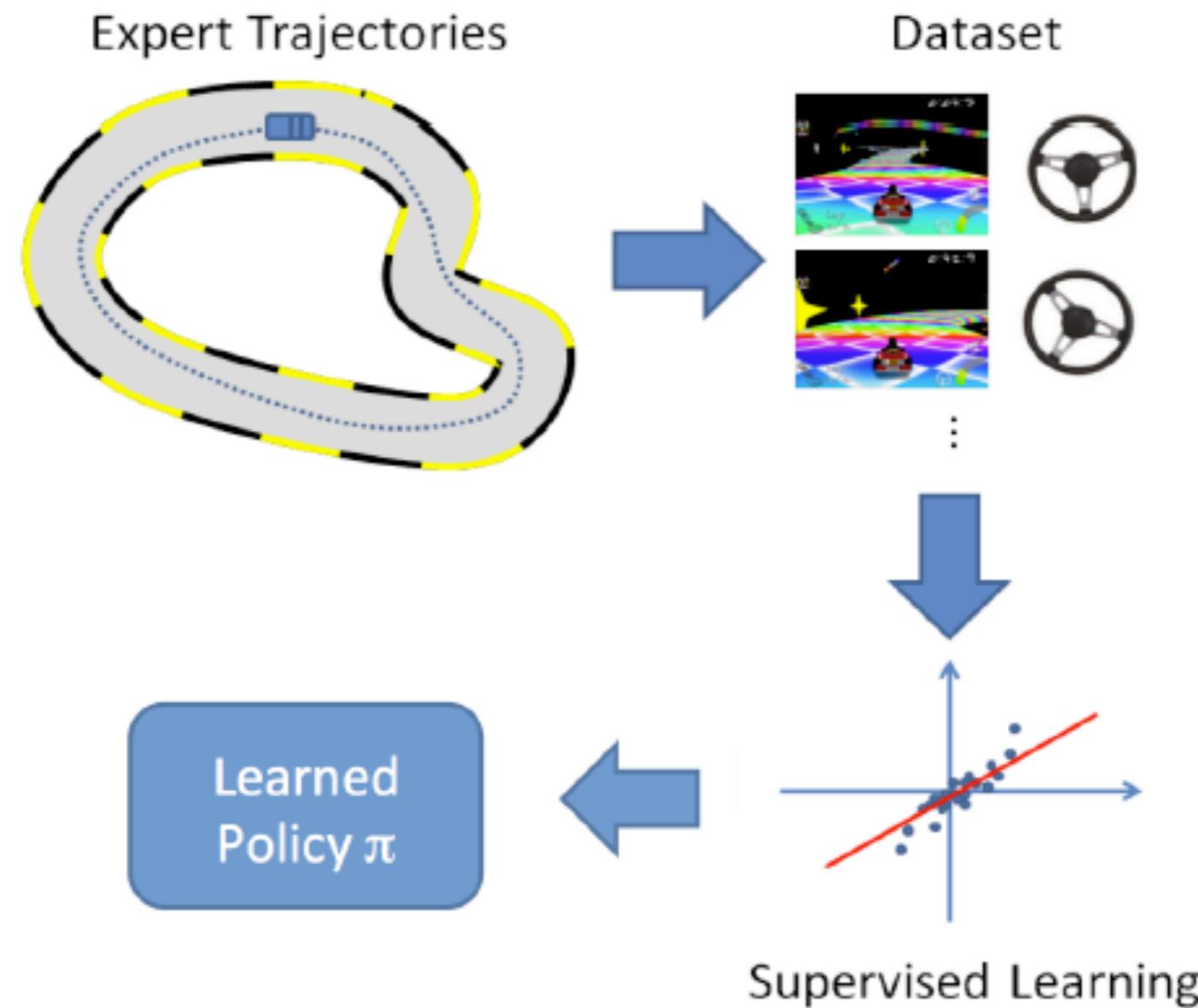
# Learning to Drive a Car: Supervised Learning



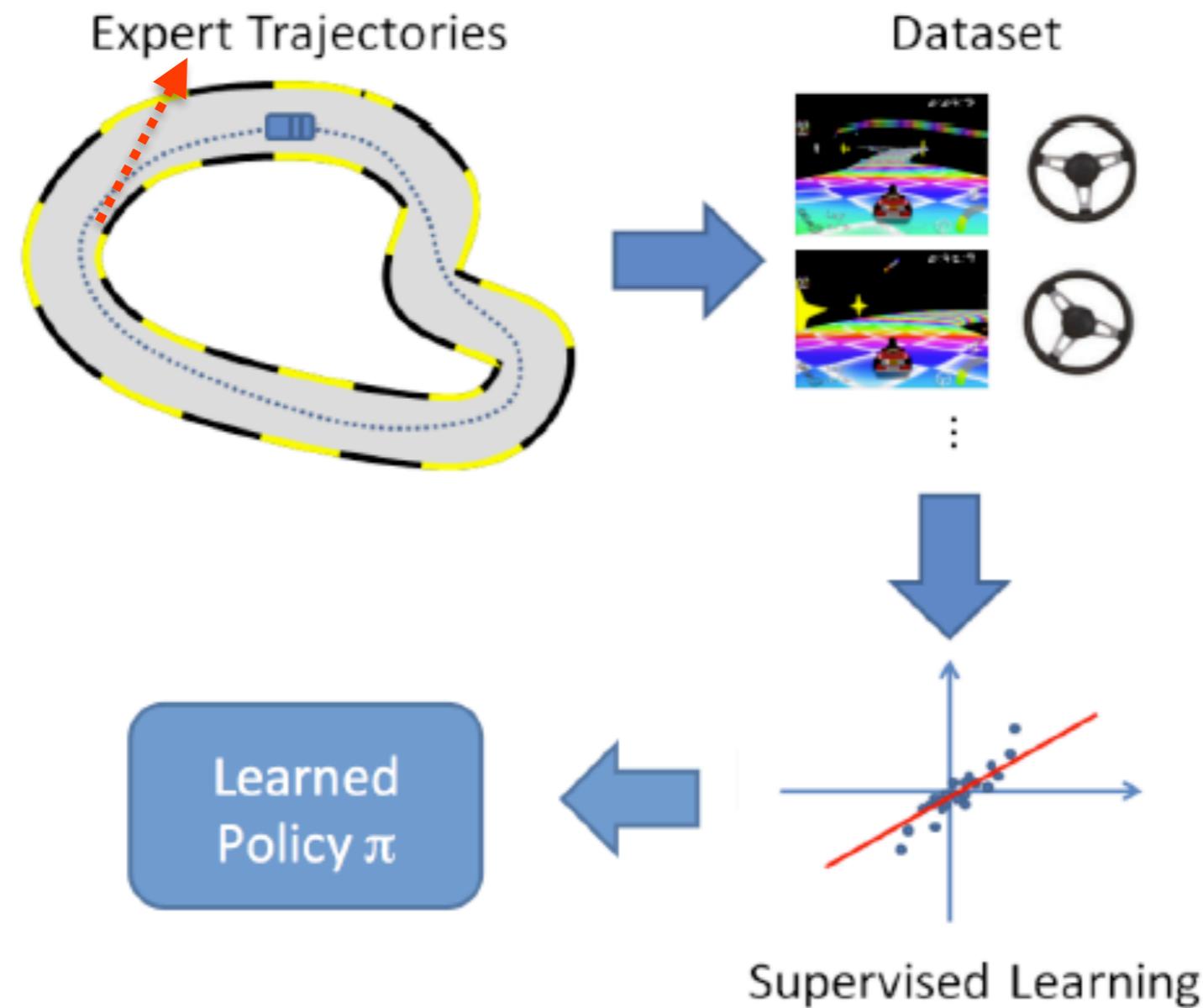
# Learning to Drive a Car: Supervised Learning



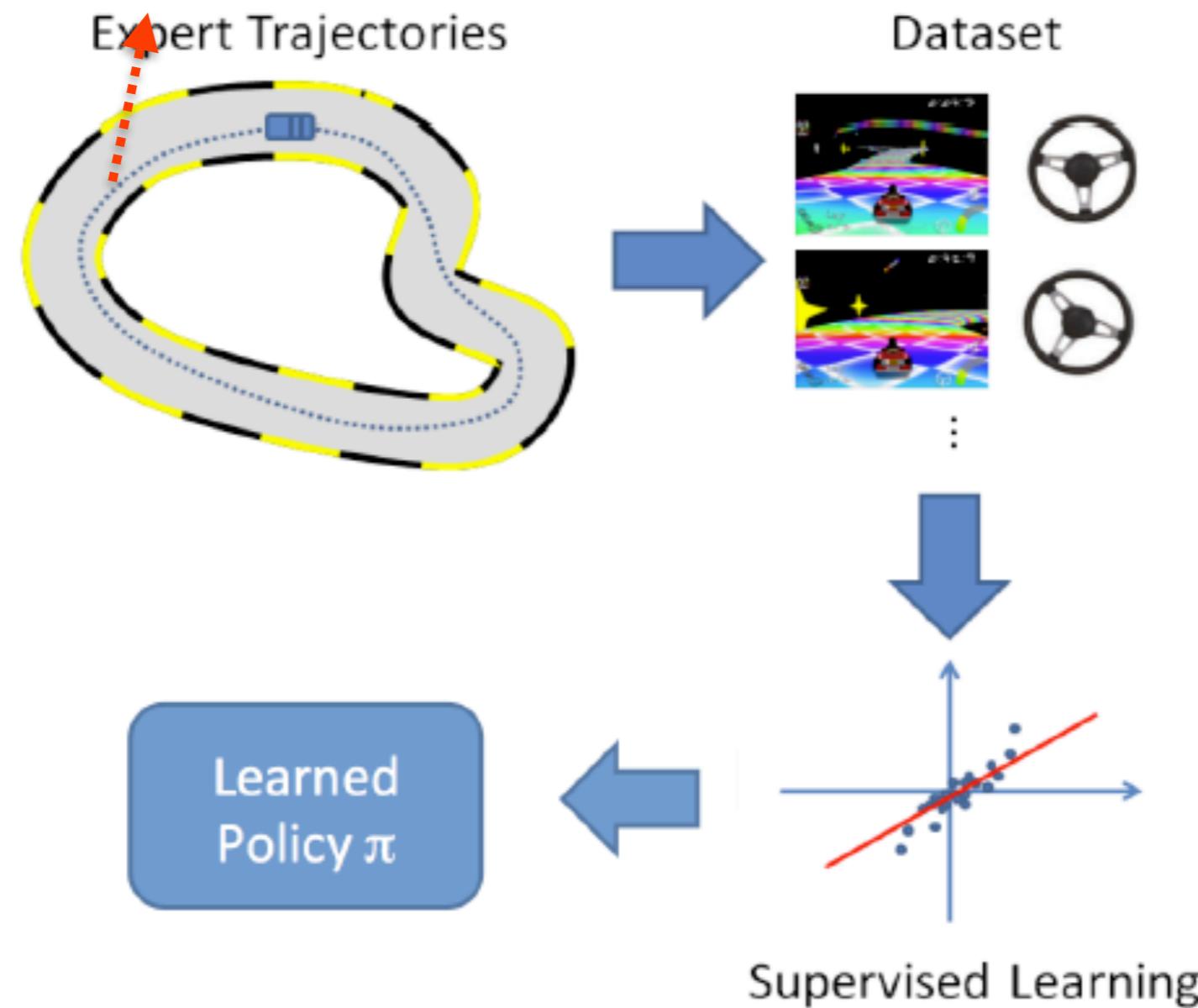
# Learning to Drive a Car: Supervised Learning



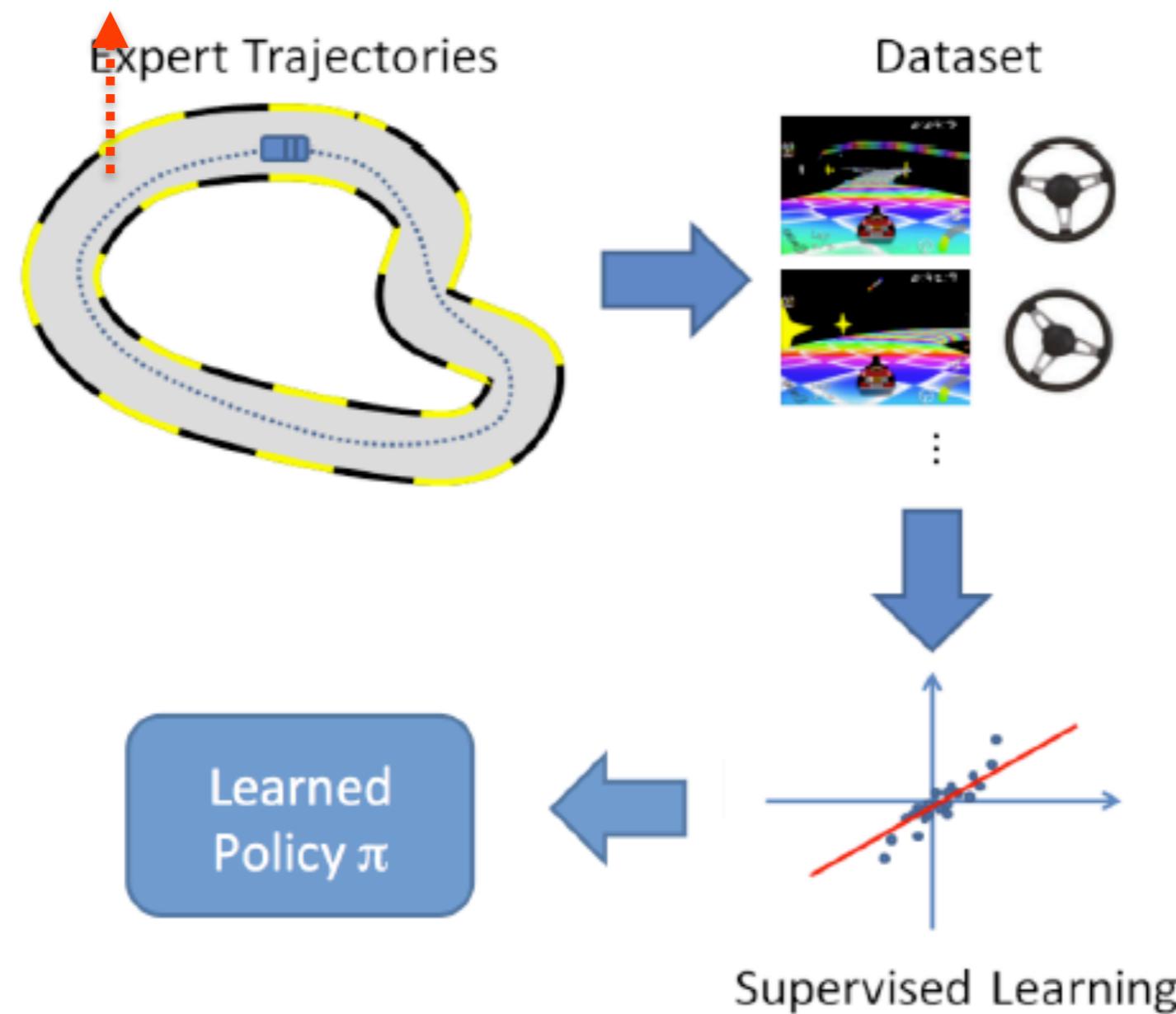
# Learning to Drive a Car: Supervised Learning



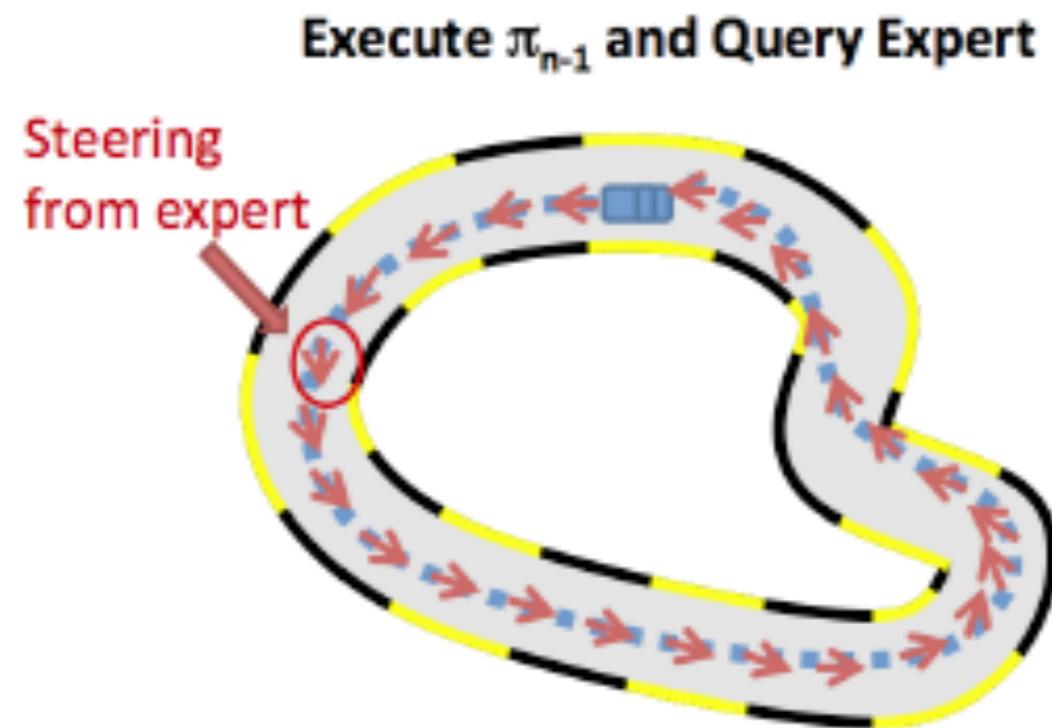
# Learning to Drive a Car: Supervised Learning



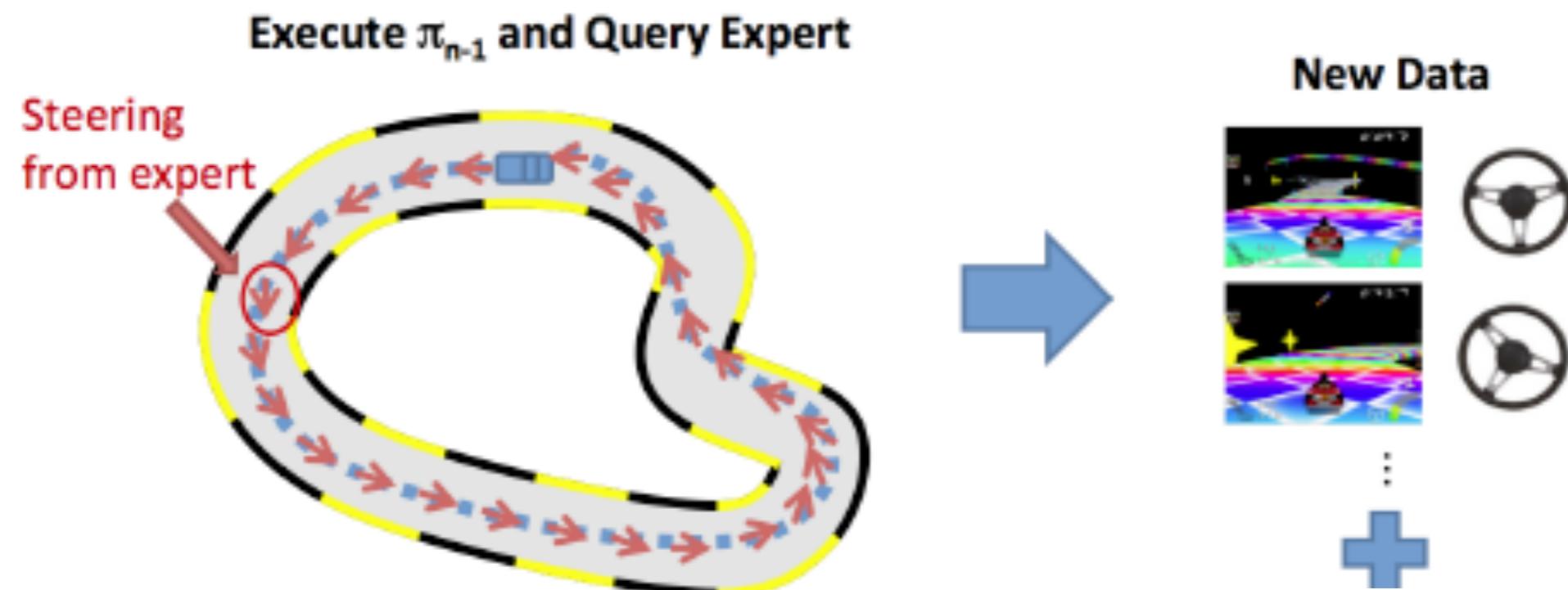
# Learning to Drive a Car: Supervised Learning



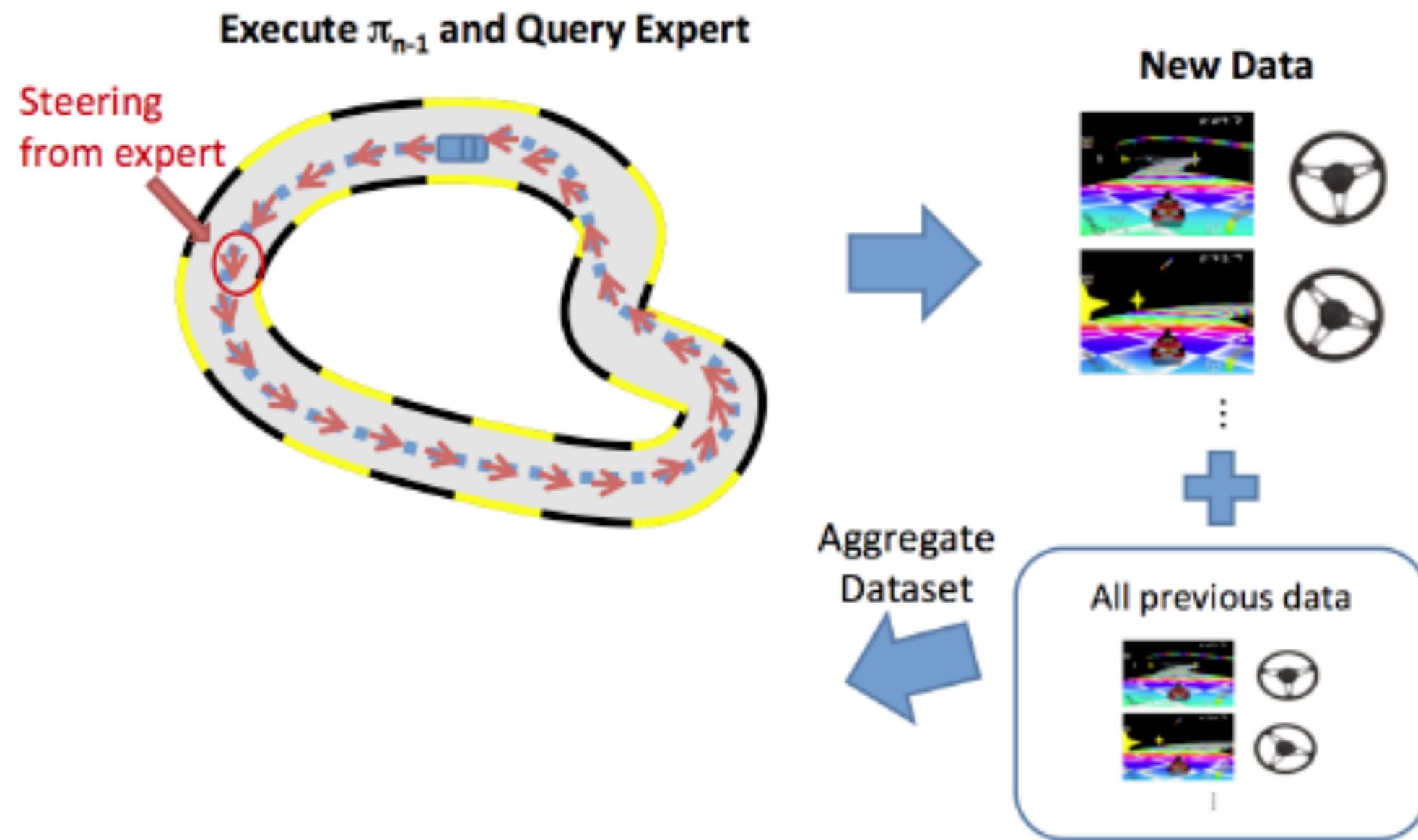
# Learning to Race a Car : Interactive learning-DAGGer



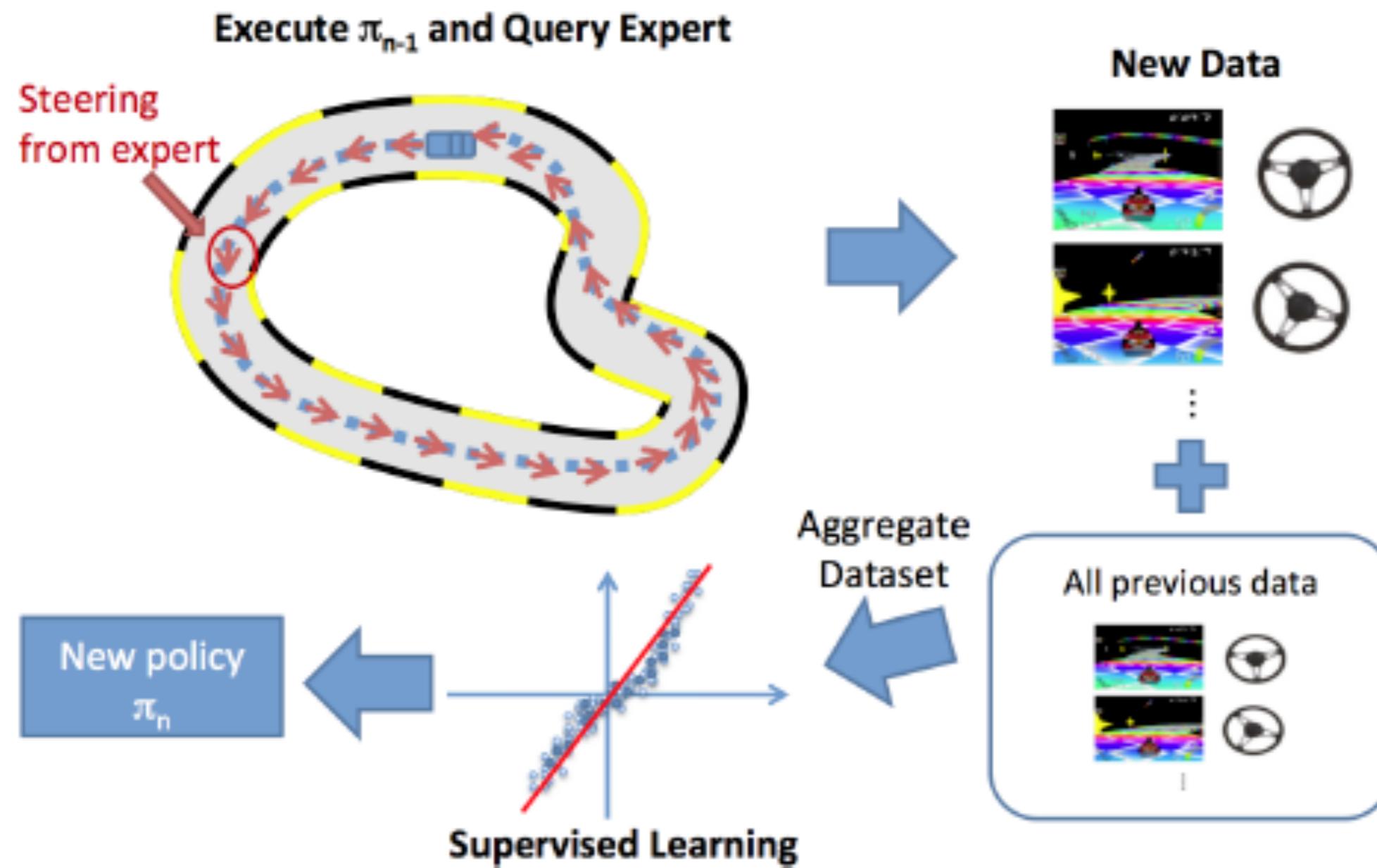
# Learning to Race a Car : Interactive learning-DAGGer



# Learning to Race a Car : Interactive learning-DAGGer

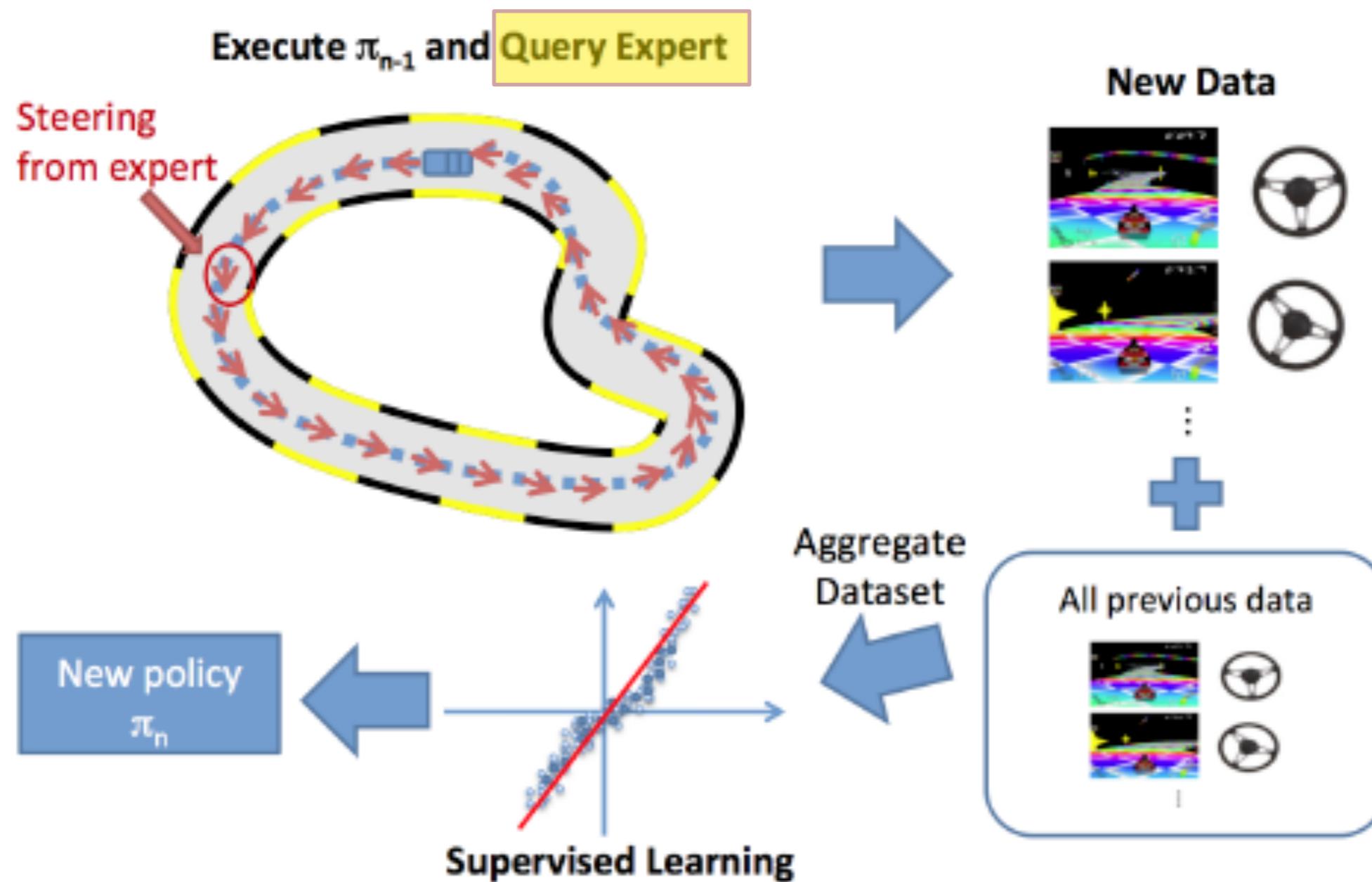


# Learning to Race a Car : Interactive learning-DAGGer



# Learning to Race a Car : Interactive learning-DAGGer

This assumes you can actively access an expert during training!



A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning  
Stephane Ross, Geoffrey J. Gordon, J. Andrew Bagnell

# DAGGER (in simulation)

Dataset AGGregation: bring learner's and expert's trajectory distributions closer by labelling additional data points resulting from applying the current policy.

# DAGGER (in simulation)

Dataset AGGregation: bring learner's and expert's trajectory distributions closer by (asking human experts to provide) labelling additional data points resulting from applying the current policy

1. Train  $\pi_\theta(u_t | o_t)$  from human data  $\mathcal{D}_{\pi^*} = \{o_1, u_1, \dots, o_N, u_N\}$
2. Run  $\pi_\theta(u_t | o_t)$  to get dataset  $\mathcal{D}_\pi = \{o_1, \dots, o_M\}$
3. Ask human to label  $\mathcal{D}_\pi$  with actions  $u_t$
4. Aggregate:  $\mathcal{D}_{\pi^*} \leftarrow \mathcal{D}_{\pi^*} \cup \mathcal{D}_\pi$
5. GOTO step 1.

## Problems:

- execute an unsafe/partially trained policy
- repeatedly query the expert

# DAGGER (on a real platform)

Application on drones: given RGB from the drone camera predict steering angles



# DAGGER (on a real platform)

Application on drones : given RGB from the drone camera predict steering angle

## Caveats:

1. It is hard for the expert to provide the right magnitude for the turn without feedback of his own actions! Solution: provide him with visual feedback



# DAGGER (on a real platform)

## Caveats:

1. Is hard for the expert to provide the right magnitude for the turn **without feedback of his own actions!** Solution: provide him with his visual feedback
  2. The expert's **reaction time** to the drone's behavior is **large**, this causes imperfect actions to be commanded. Solution: play-back in slow motion offline and record their actions.
  3. Executing an **imperfect policy causes accidents**, crashes into obstacles. Solution: safety measures which make again the data distribution matching imperfect between train and test, but good enough.
- **Experts do not need to be humans.**
  - Machinery that we develop in this lecture can be used for imitating expert policies found through (easier) optimization in a constrained smaller part of the state space.
  - Imitation then means distilling knowledge of expert constrained policies into a general policy that can do well in all scenarios the simpler policies do well.

# What can go wrong?

- Compounding errors  
Fix: data augmentation
- Non-markovian observations  
Fix: observation concatenation or recurrent models
- Stochastic expert actions  
Fix: generative modelling (GANs, stochastic latent variable models, action discretisation, gaussian mixture networks)

# Markov property

A stochastic process has the Markov property if the **conditional probability distribution** of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it

$$\mathbb{P}[R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_1, \dots, S_{t-1}, A_{t-1}, R_t, S_t, A_t] = \mathbb{P}[R_{t+1} = r, S_{t+1} = s' | S_t, A_t]$$

for all  $s' \in \mathcal{S}$ ,  $r \in \mathcal{R}$  and all histories

# Non-markovian observations

$$\pi_\theta(\mathbf{u}_t | \mathbf{o}_t)$$

behavior depends only  
on current observation

$$\pi_\theta(\mathbf{u}_t | \mathbf{o}_1, \dots, \mathbf{o}_t)$$

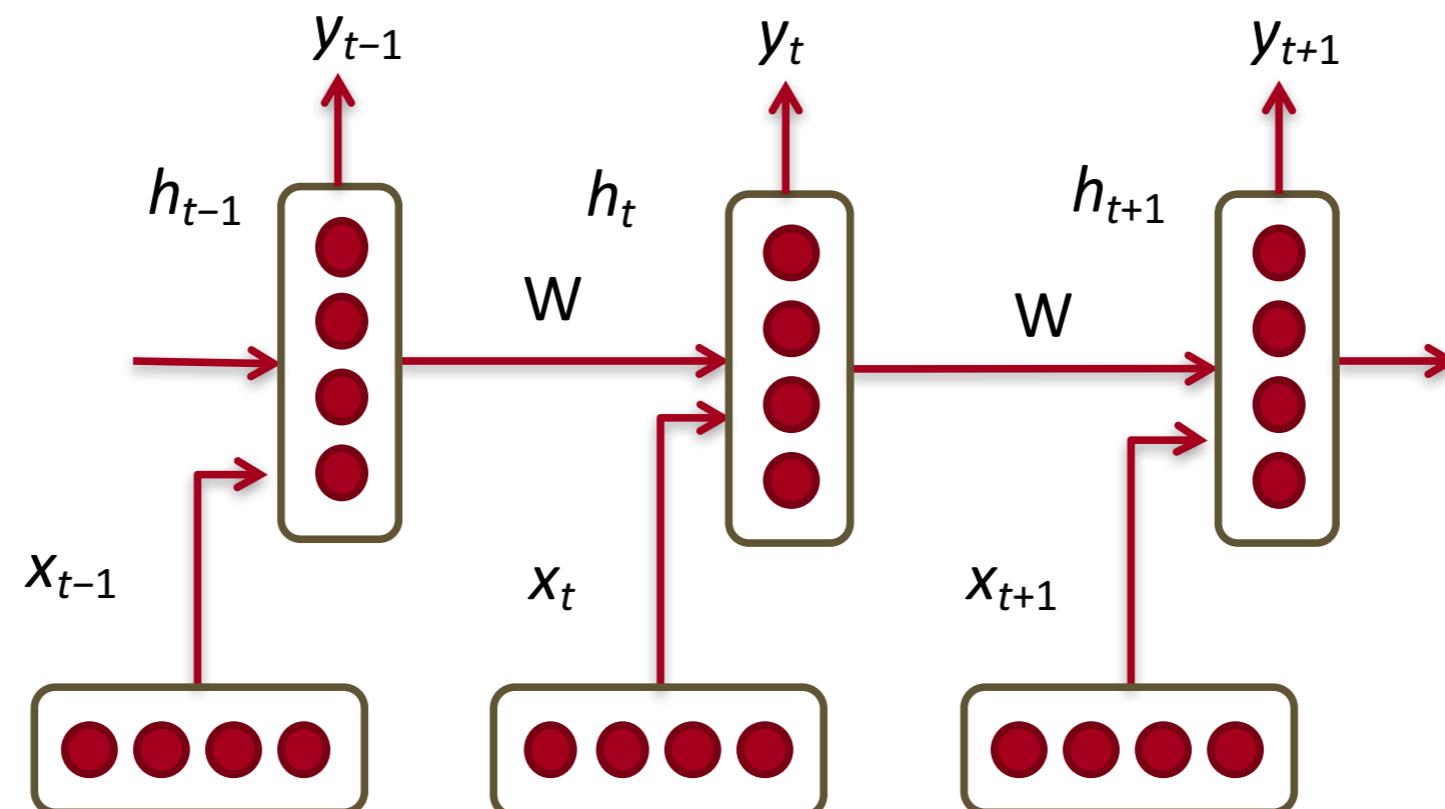
behavior depends on  
all past observations

# Fix 1: concatenate observations

# Fix 2: use (vanilla) recurrent networks

# Recurrent Neural Networks (RNNs)

- RNNs tie the weights at each time step
- Condition the neural network on all previous inputs
- In principle, any interdependencies can be modeled across time steps.
- In practice, limitations from SGD training, capacity, initialization etc.
- Later lecture we will explain why vanilla LSTMs



*Diagram from Richard Socher*

# Recurrent Neural Network (single hidden layer)

Given list of vectors:

$$x_1, \dots, x_{t-1}, x_t, x_{t+1}, \dots, x_T$$

At a single time step:

$$h_t = \sigma(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]})$$

$$\hat{y}_t = \text{softmax}(W^{(S)} h_t)$$

(in case of discrete labels)

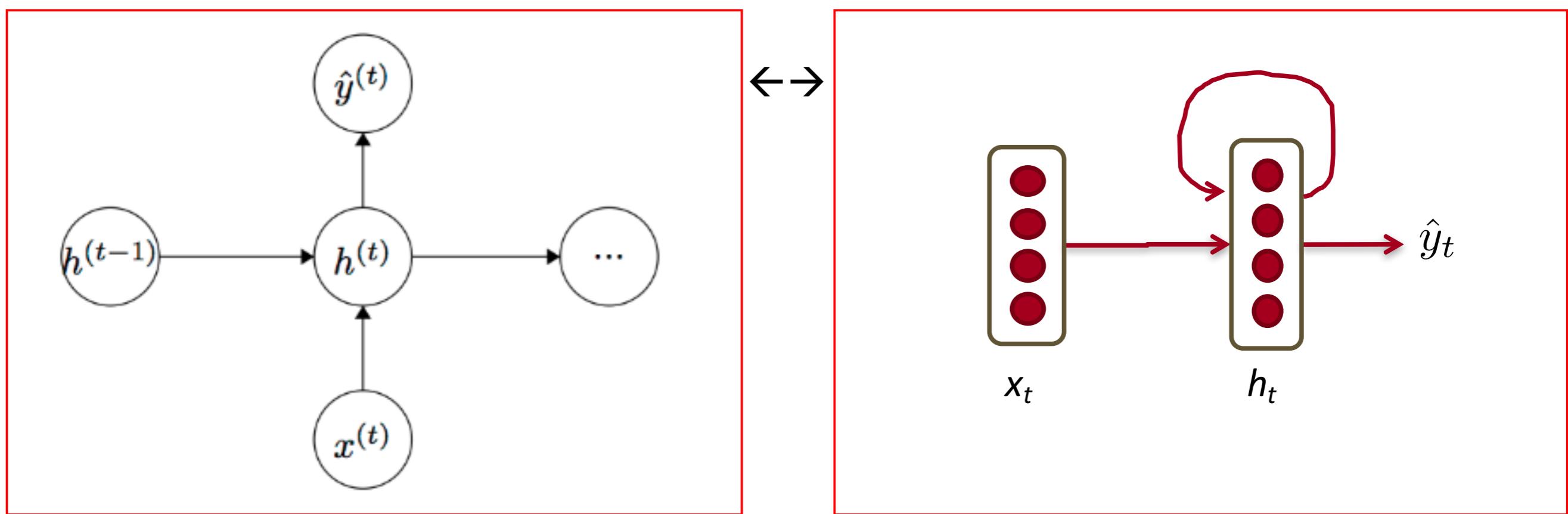


Diagram from Richard Socher

# Fix 2: use recurrent networks

Diagram from Sergey Levine

# Fix 2: use recurrent networks



- Usually much more structure is needed in the latent state than a vanilla LSTM can provide, e.g., detections and trackless of objects.
- We will discuss later in the course structured recurrent neural networks for video perception.