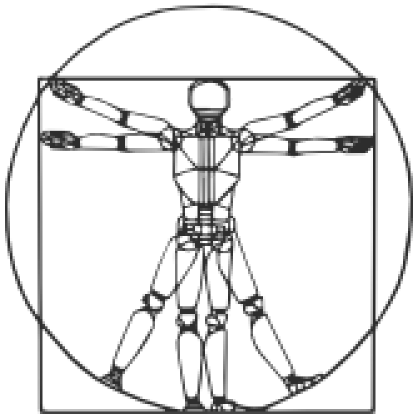


CSC2626
Imitation Learning for Robotics

Florian Shkurti

Week 1: Behavioral Cloning vs. Imitation

Today's agenda

- Administrivia
- Topics covered by the course
- Behavioral cloning
- Imitation learning
- Quiz about background and interests
- (Time permitting) Query the expert only when policy is uncertain

Administrivia

Administrivia

This is a graduate level course

Course website: http://www.cs.toronto.edu/~florian/courses/csc2626w22

Discussion forum + announcements: https://q.utoronto.ca (Quercus)

Request improvements anonymously: https://www.surveymonkey.com/r/LJJV5LY

Course-related emails should have CSC2626 in the subject

Prerequisites

Mandatory:

- Introductory machine learning (e.g. CSC411/ECE521 or equivalent)
- Basic linear algebra + multivariable calculus
- Intro to probability
- Programming skills in Python or C++ (enough to validate your ideas)

Recommended:

- Experience training neural networks or other function approximators
- Introductory concepts from reinforcement learning or control (e.g. value function/cost-to-go)

Prerequisites

Mandatory:

- Introductory machine learning (e.g. CSC411/ECE521 or equivalent)
- Basic linear algebra + multivariable calculus
- Intro to probability
- Programming skills in Python or C++ (enough to validate your ideas)

If you're missing any of these this is not the course for you.

You're welcome to audit.

Recommended:

- Experience training neural networks or other function approximators
- Introductory concepts from reinforcement learning or control (e.g. value function/cost-to-go)

If you're missing this we can organize tutorials to help you.

Grading

Two assignments: 50%

Course project: 50%

- Project proposal: 10%
- Midterm progress report: 5%
- Project presentation: 5%
- Final project report (6-8 pages) + code: 30%

Project guidelines

http://www.cs.toronto.edu/~florian/courses/csc2626w22/CSC2626_Project_Guidelines.pdf

Grading

Two assignments: 50%

Individual submissions

Course project: 50%

- Project proposal: 10%
- Midterm progress report: 5%
- Project presentation: 5%
- Final project report (6-8 pages) + code: 30%

Project guidelines

http://www.cs.toronto.edu/~florian/courses/csc2626w22/CSC2626_Project_Guidelines.pdf

Grading

Two assignments: 50% — Individual submissions

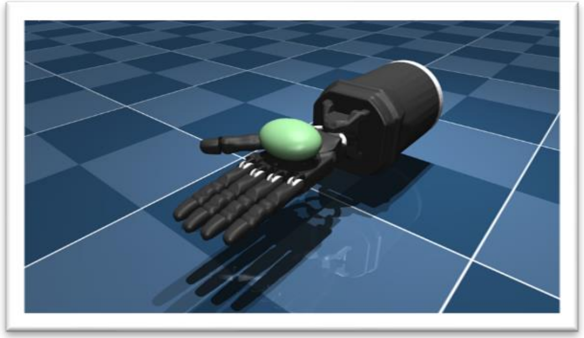
Course project: 50% — Groups of 2-3

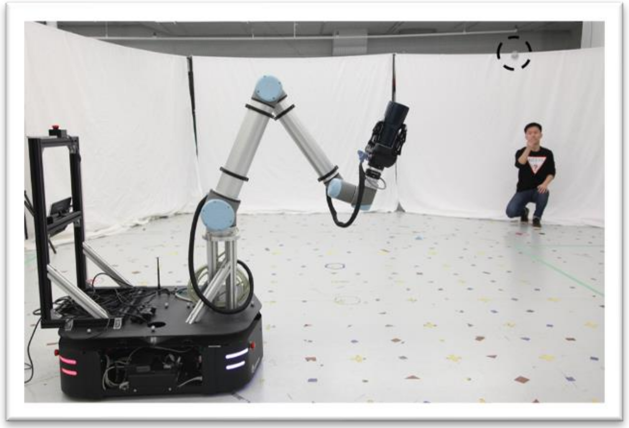
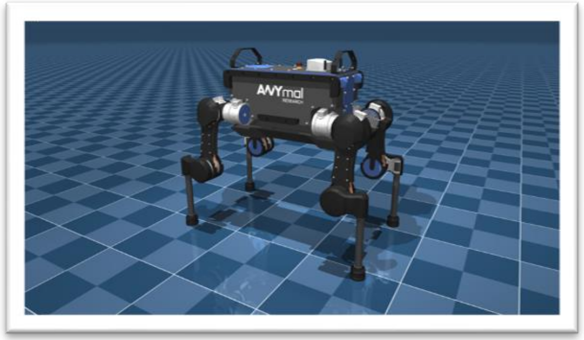
- Project proposal: 10%
- Midterm progress report: 5%
- Project presentation: 5%
- Final project report (6-8 pages) + code: 30%

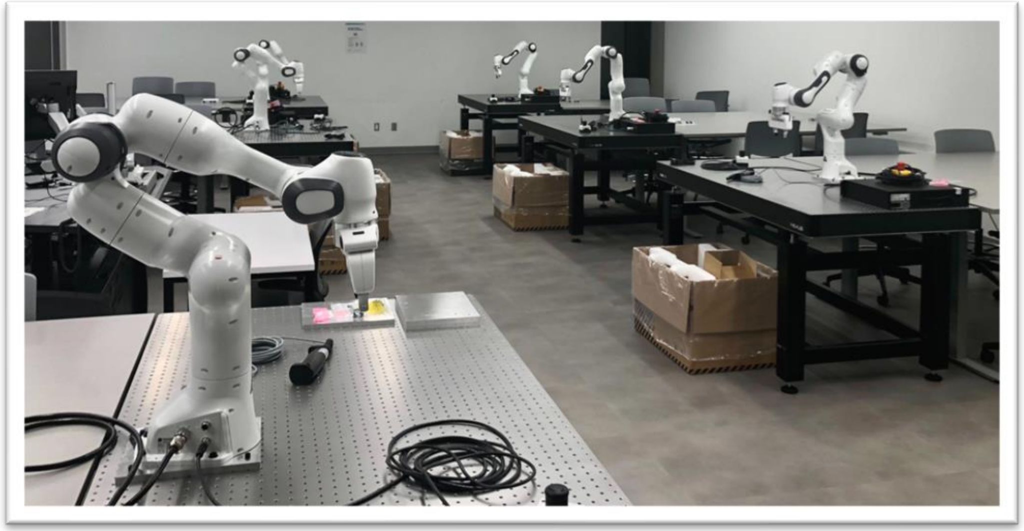
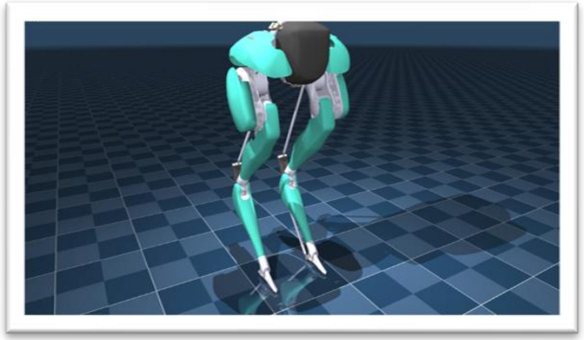
Project guidelines

http://www.cs.toronto.edu/~florian/courses/csc2626w22/CSC2626_Project_Guidelines.pdf

Evaluation environments: simulators & real robots

Guiding principles for this course

Robots do not operate in a vacuum. They do not need to learn everything from scratch.

Guiding principles for this course

Robots do not operate in a vacuum. They do not need to learn everything from scratch.

Humans need to easily interact with robots and share our expertise with them.

Guiding principles for this course

Robots do not operate in a vacuum. They do not need to learn everything from scratch.

Humans need to easily interact with robots and share our expertise with them.

Robots need to learn from the behavior and experience of others, not just their own.

Main questions

How can robots incorporate others' decisions into their own?

How can robots easily understand our objectives from demonstrations?

How do we balance autonomous control and human control in the same system?

Main questions

How can robots incorporate others' decisions into their own?

Learning from demonstrations
Apprenticeship learning
Imitation learning

How can robots easily understand our objectives from demonstrations?

Reward/cost learning
Task specification
Inverse reinforcement learning
Inverse optimal control
Inverse optimization

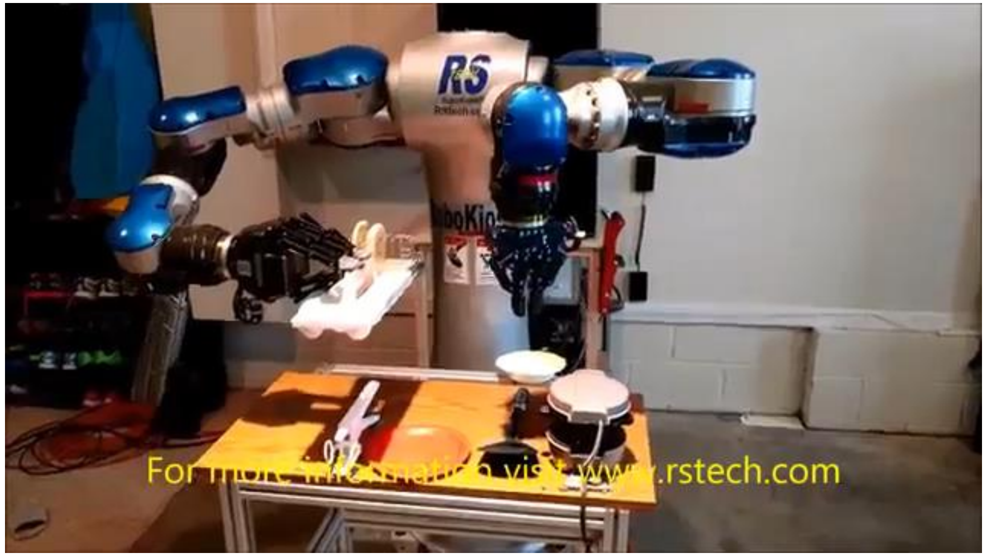
How do we balance autonomous control and human control in the same system?

Shared or sliding autonomy

Applications

Any control problem where:

- **writing down a dense cost function is difficult**
- there is a hierarchy of decision-making processes
- our engineered solutions might not cover all cases
- unrestricted exploration during learning is slow or dangerous



https://www.youtube.com/watch?v=M8r0gmQXm1Y

Applications

Any control problem where:

- **writing down a dense cost function is difficult**
- there is a hierarchy of interacting decision-making processes
- our engineered solutions might not cover all cases
- unrestricted exploration during learning is slow or dangerous

https://www.youtube.com/watch?v=Q3LXJGha7Ws

Applications

Any control problem where:

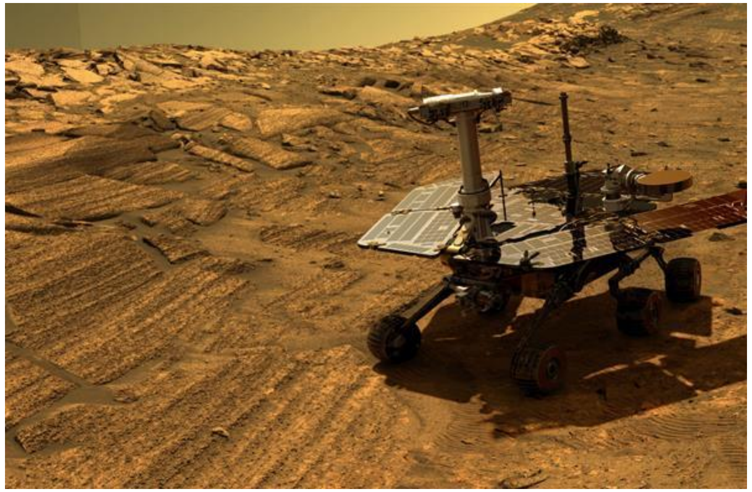
- **writing down a dense cost function is difficult**
- there is a hierarchy of interacting decision-making processes
- our engineered solutions might not cover all cases
- unrestricted exploration during learning is slow or dangerous

https://www.youtube.com/watch?v=RjGe0GiiFzw

Applications

Any control problem where:

- **writing down a dense cost function is difficult**
- there is a hierarchy of interacting decision-making processes
- our engineered solutions might not cover all cases
- unrestricted exploration during learning is slow or dangerous



Robot explorer

Applications

Any control problem where:

- **writing down a dense cost function is difficult**
- there is a hierarchy of interacting decision-making processes
- our engineered solutions might not cover all cases
- unrestricted exploration during learning is slow or dangerous

https://www.youtube.com/watch?v=0XdC1HUp-rU

Back to the future

https://www.youtube.com/watch?v=I39sxwYKIEE

Ernst Dickmans + Mercedes
(1986-2003)

https://www.youtube.com/watch?v=2KMAAmkz9go

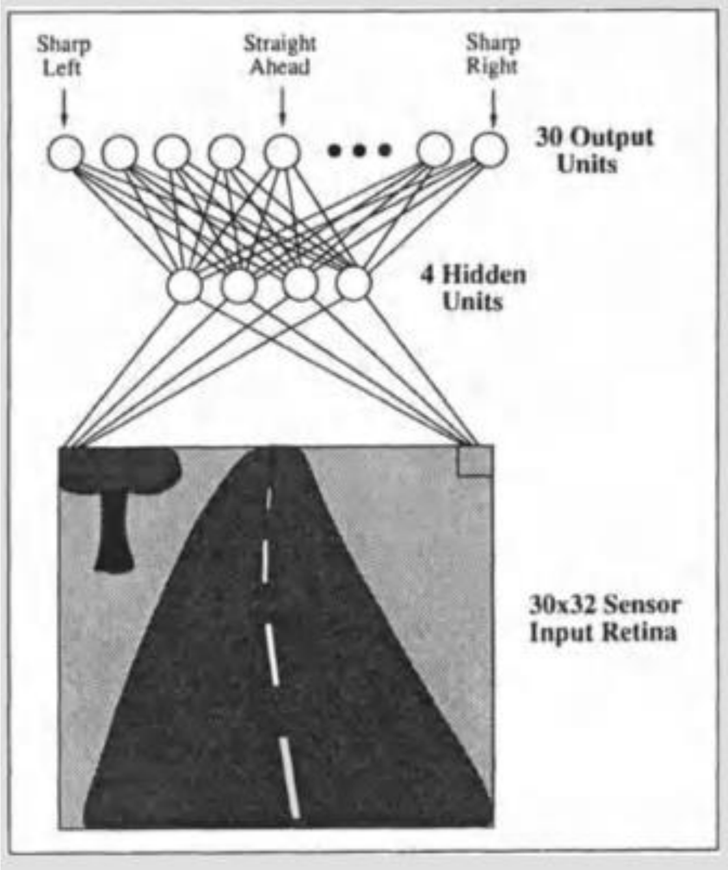
Navlab 1
(1986-1989)

https://www.youtube.com/watch?v=iIP4aPDTBPE

Navlab 2 + ALVINN, Dean Pomerleau's PhD thesis
(1989-1993)

30 x 32 pixels, 3-layer network, outputs steering command, ~5 minutes of training per road type

ALVINN: architecture



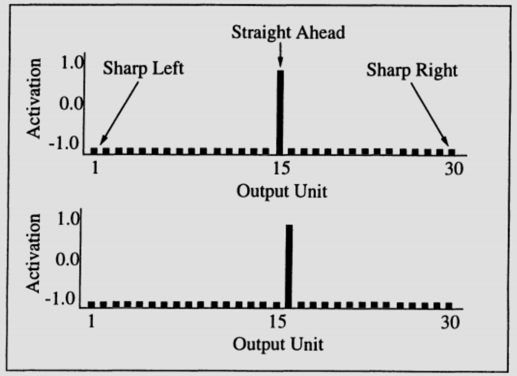


Figure 2.7: The representation of two steering directions using a "one-of-N" encoding. The top graph represents a straight ahead steering direction, since the middle output unit is activated. The bottom graph represents a slight right turn, since an output unit slightly right of center is activated.

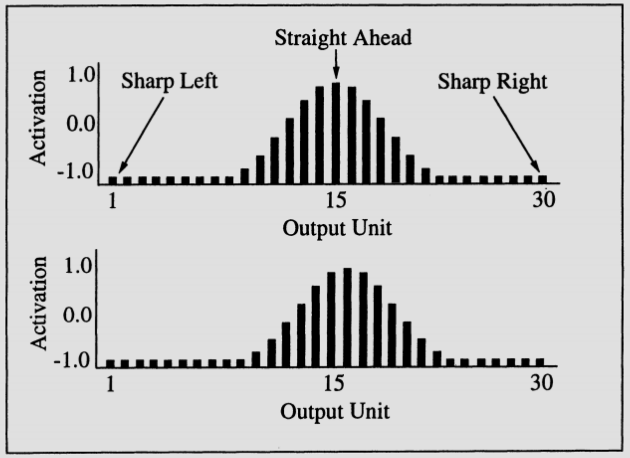


Figure 2.10: The representation of two steering directions using a gaussian output encoding. The top graph represents a straight ahead steering direction, since the gaussian "hill" of activation is centered on the middle output unit. The bottom graph represents a slight right turn, since the "hill" of activation is centered slightly right of the middle unit.

https://drive.google.com/file/d/0Bz9namoRIUKMa0pJYzRGSFVwbm8/view

Dean Pomerleau's PhD thesis

ALVINN: training set

To generate synthetic training data for the task of autonomous road following, I developed a program that generated aerial views of simulated stretches of roads and then used a model of the camera to back-project the aerial map into a 2D image of the road ahead. The simulated road image generator used nearly 200 parameters in order to generate a variety of realistic road images. Some of the most important parameters are listed in Figure 3.1.

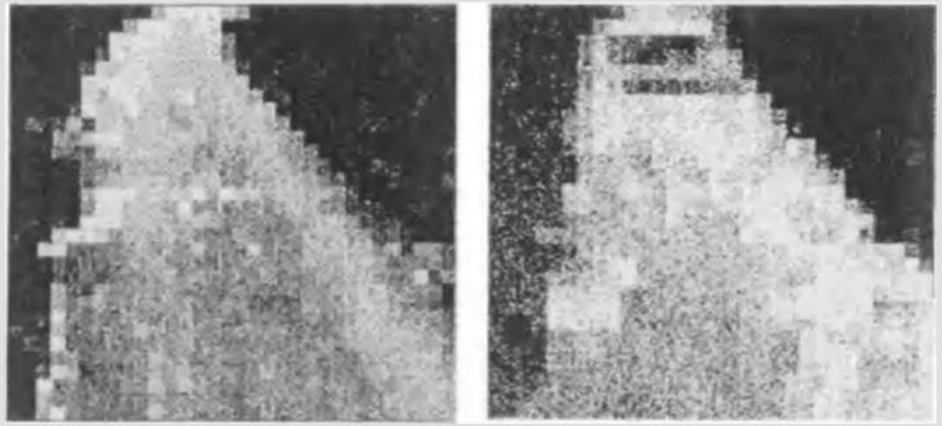
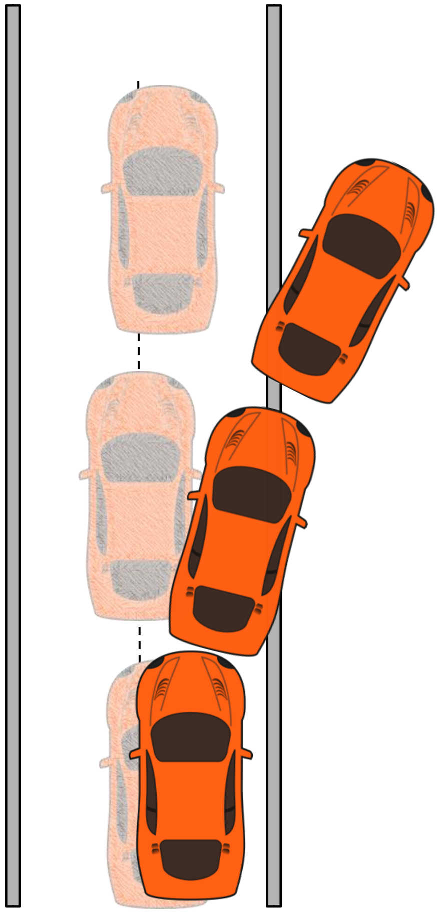


Figure 3.2: A low resolution video image of a single lane road (left) and an artificial single lane road image created by the road image generator (right).

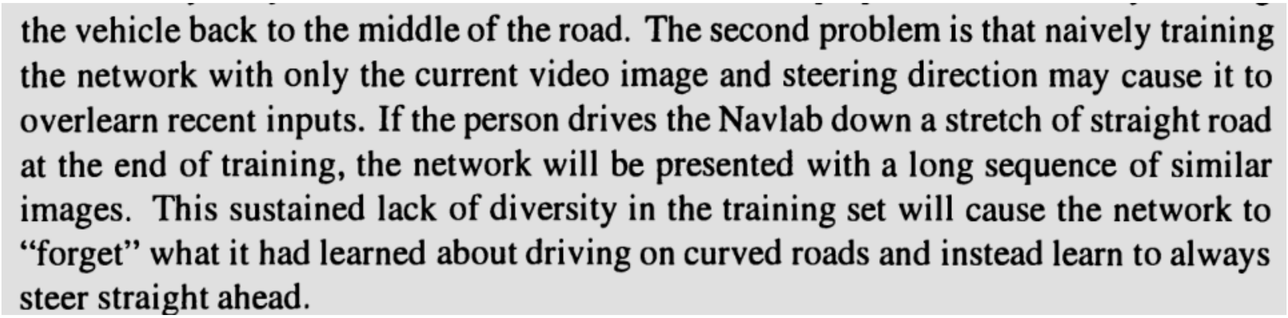
3.2 Training "on-the-fly" with Real Data

Online updates via backpropagation

Problems Identified by Pomerleau



1. **Test distribution is different from training distribution (covariate shift)**


the vehicle back to the middle of the road. The second problem is that naively training the network with only the current video image and steering direction may cause it to overlearn recent inputs. If the person drives the Navlab down a stretch of straight road at the end of training, the network will be presented with a long sequence of similar images. This sustained lack of diversity in the training set will cause the network to "forget" what it had learned about driving on curved roads and instead learn to always steer straight ahead.

2. **Catastrophic forgetting**

(Partially) Addressing Covariate Shift

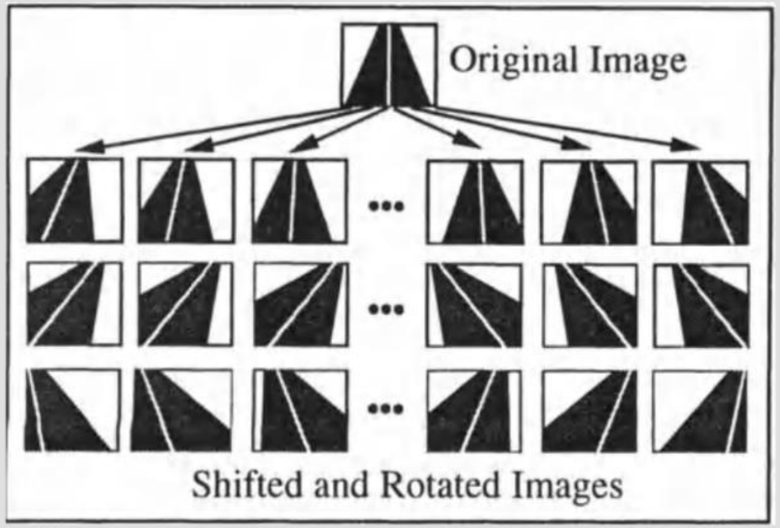
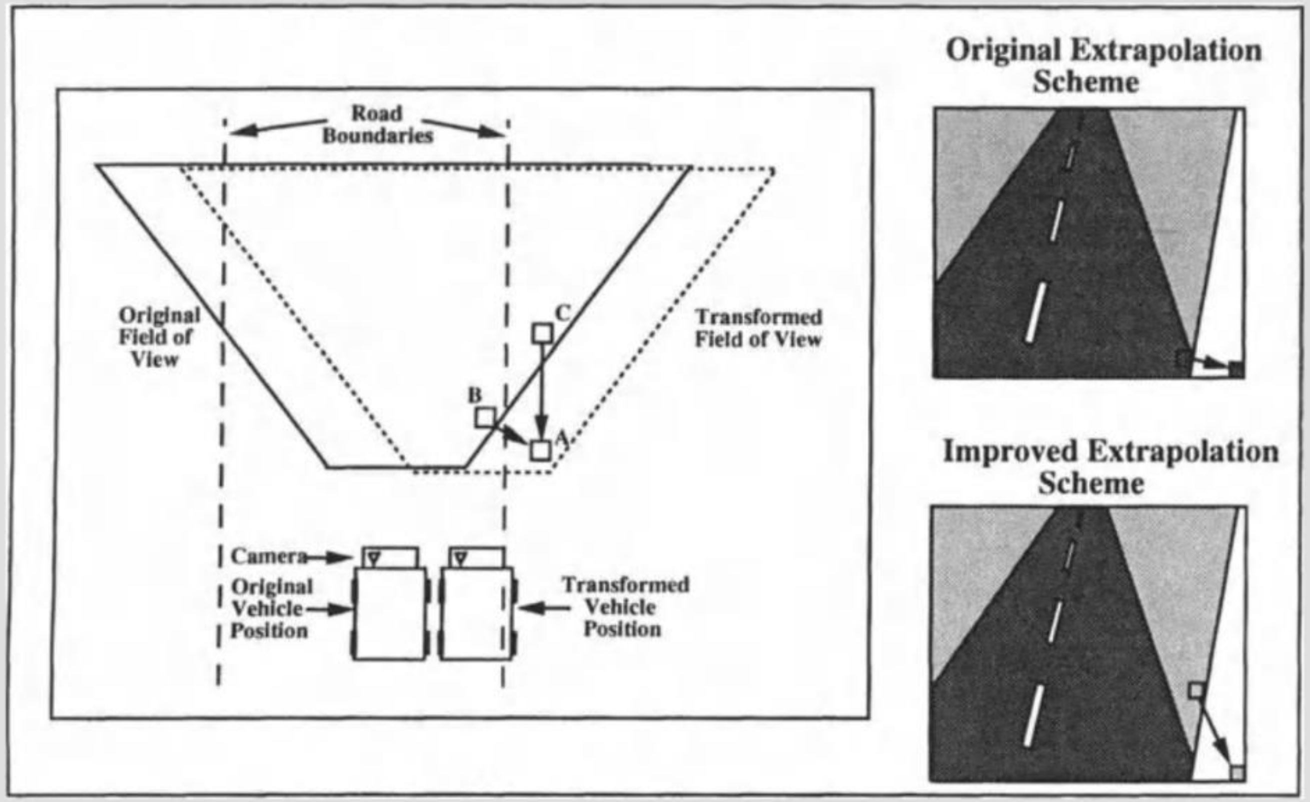


Figure 3.4: The single original video image is shifted and rotated to create multiple training exemplars in which the vehicle appears to be at different locations relative to the road.

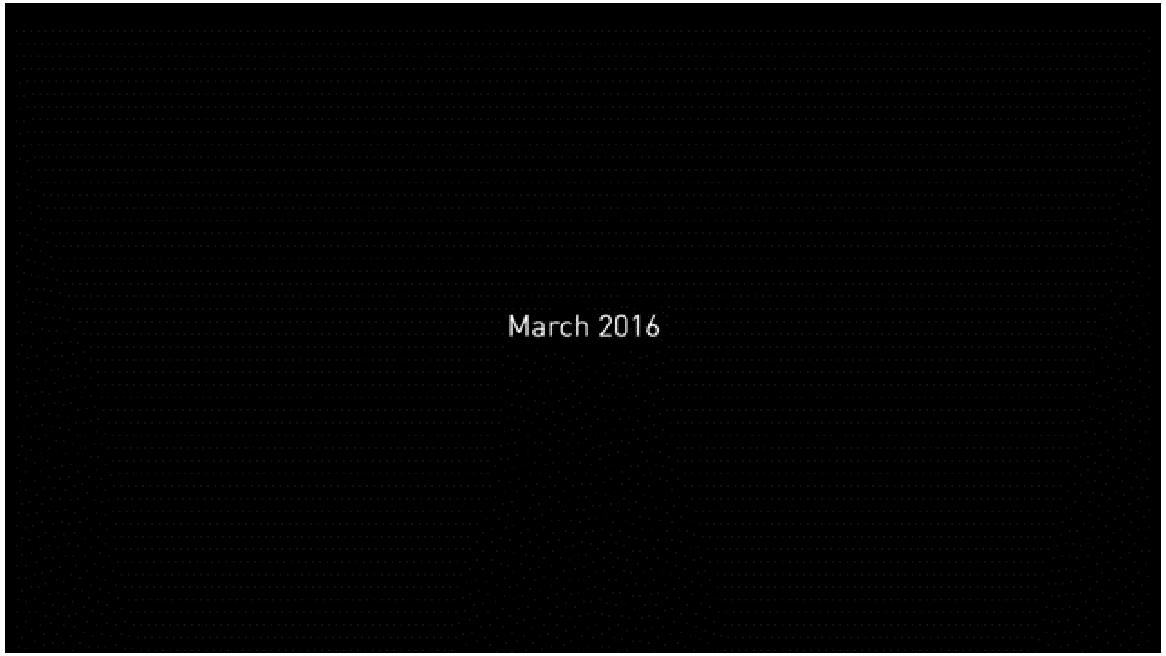


(Partially) Addressing Catastrophic Forgetting

1. Maintains a buffer of old (image, action) pairs
2. Experiments with different techniques to ensure diversity and avoid outliers

Behavioral Cloning = Supervised Learning

25 years later: what has changed?



https://www.youtube.com/watch?v=qhUvQiKec2U

What has changed?

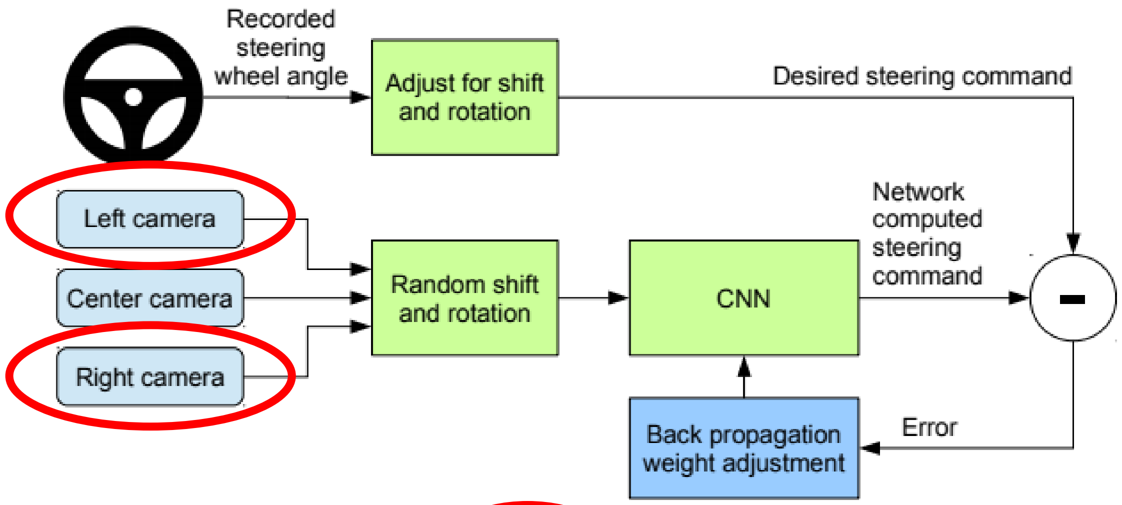


Figure 2: Training the neural network.

offline

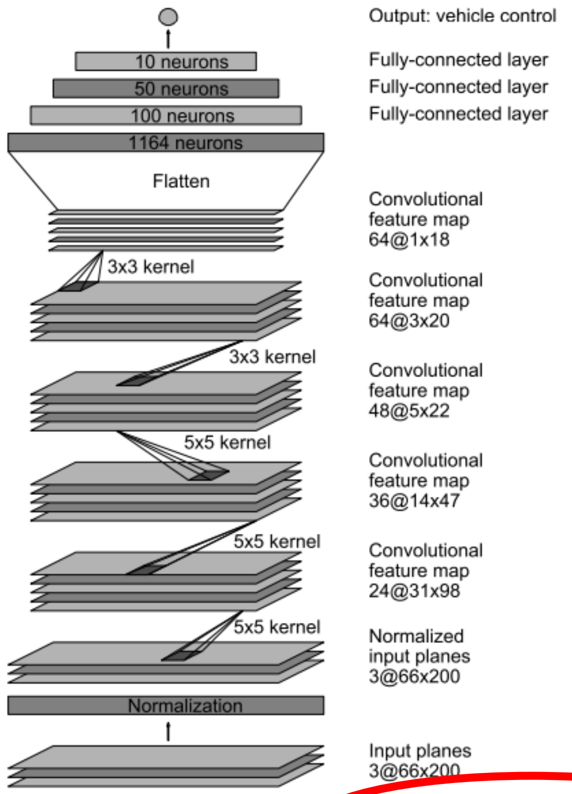


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

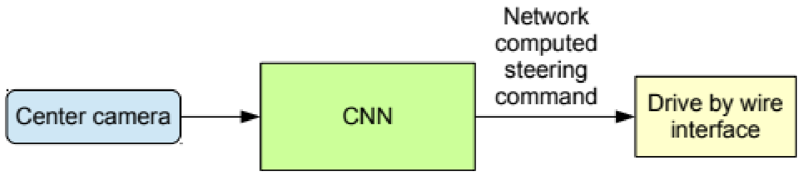


Figure 3: The trained network is used to generate steering commands from a single front-facing center camera.

End to End Learning for Self-Driving Cars, Bojarski et al, 2016

What has changed?

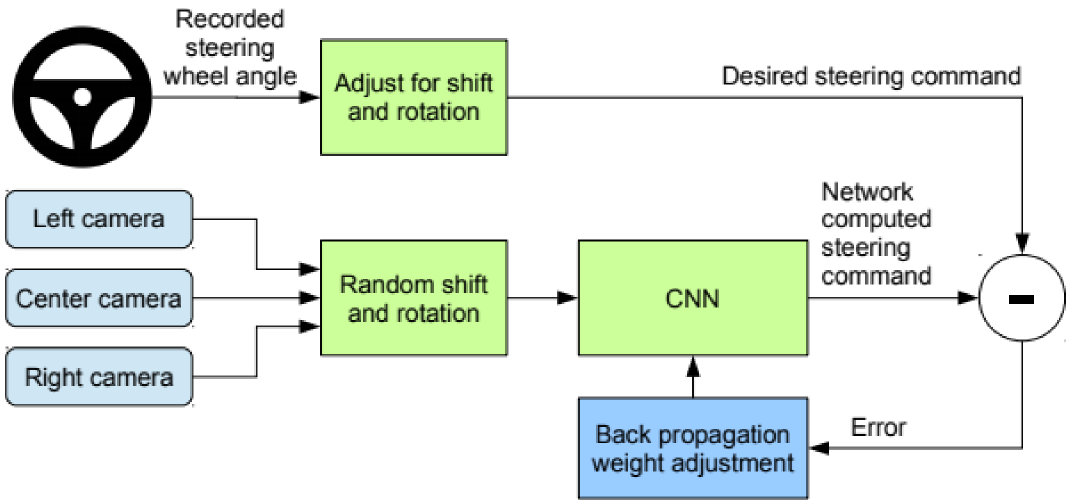


Figure 2: Training the neural network.

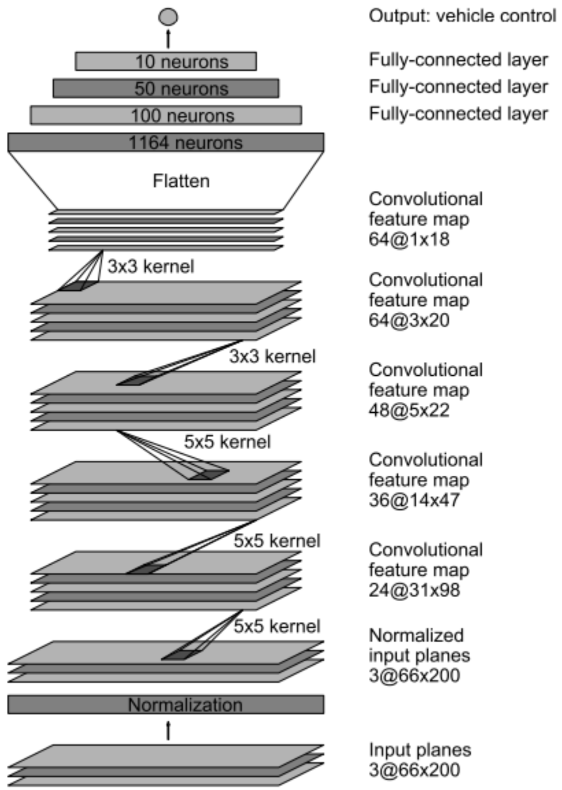


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

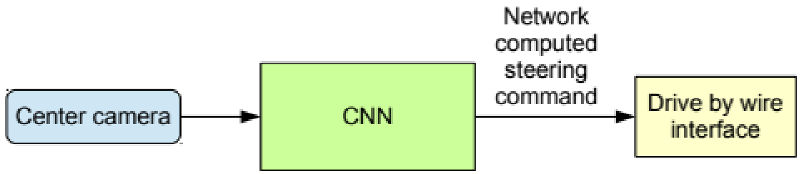


Figure 3: The trained network is used to generate steering commands from a single front-facing center camera.

"Our collected data is labeled with road type, weather condition, and the driver's activity (staying in a lane, switching lanes, turning, and so forth)."

End to End Learning for Self-Driving Cars, Bojarski et al, 2016

What has changed?

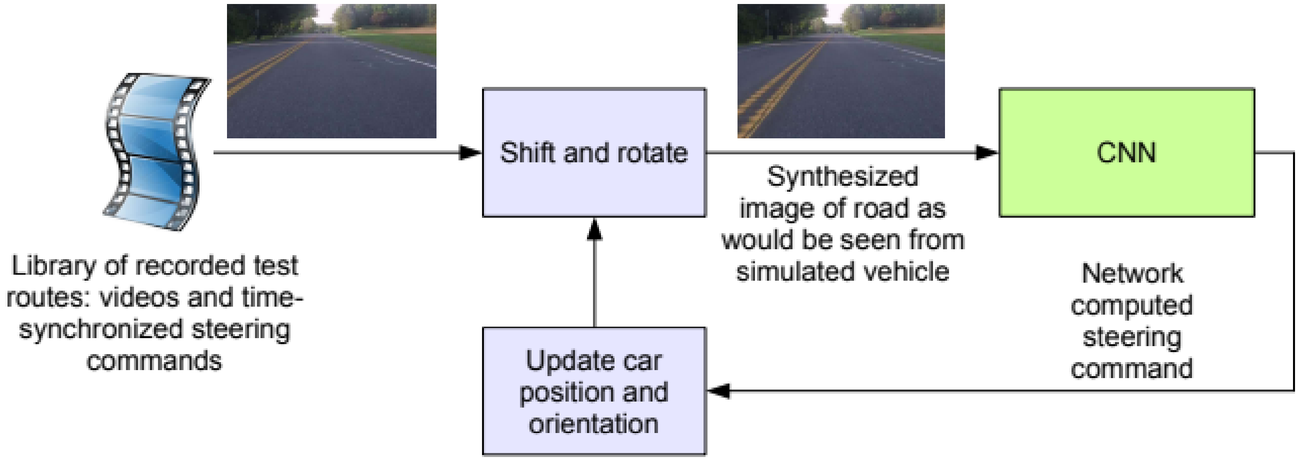


Figure 5: Block-diagram of the drive simulator.

How much has changed?

A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots, Giusti et al., 2016

https://www.youtube.com/watch?v=umRdt3zGgpU

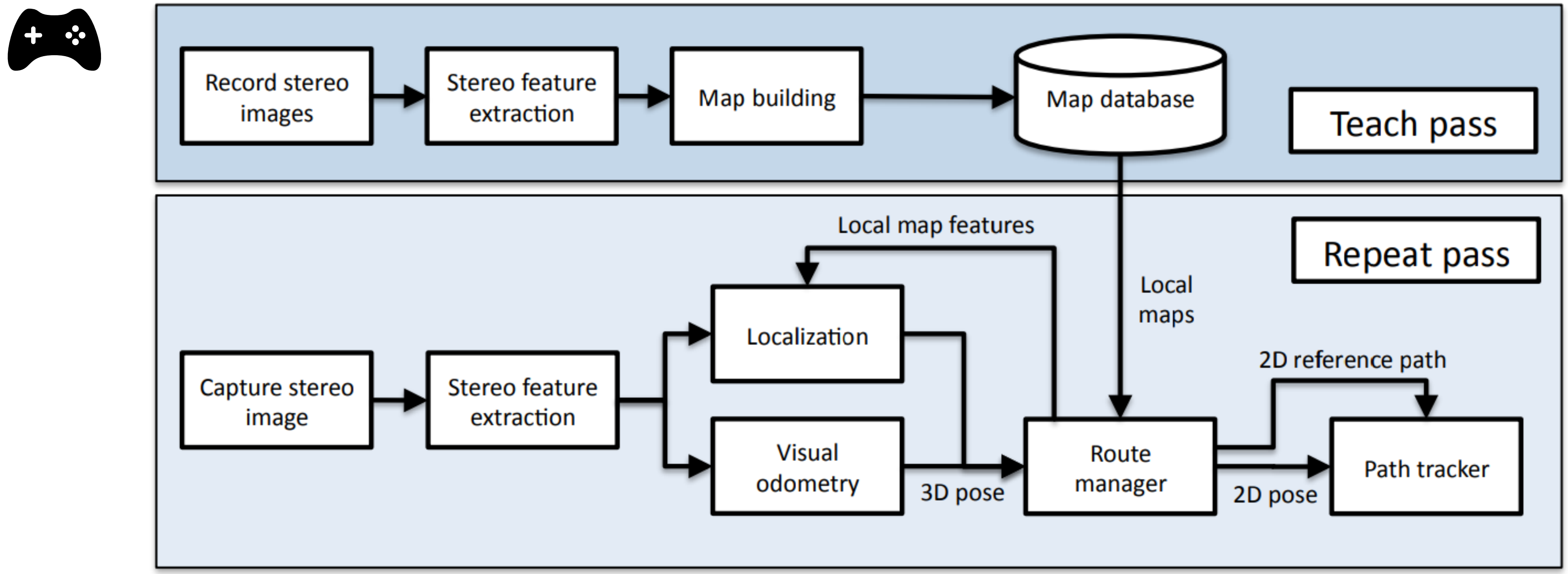
How much has changed?

Not a lot for learning lane following with neural networks.

But, there are a few other beautiful ideas that do not involve end-to-end learning.

Visual Teach & Repeat

Human Operator or
Planning Algorithm



Visual Path Following on a Manifold in Unstructured Three-Dimensional Terrain, Furgale & Barfoot, 2010

Visual Teach & Repeat

Key Idea #1: Manifold Map

Build local maps relative to the path. No global coordinate frame.

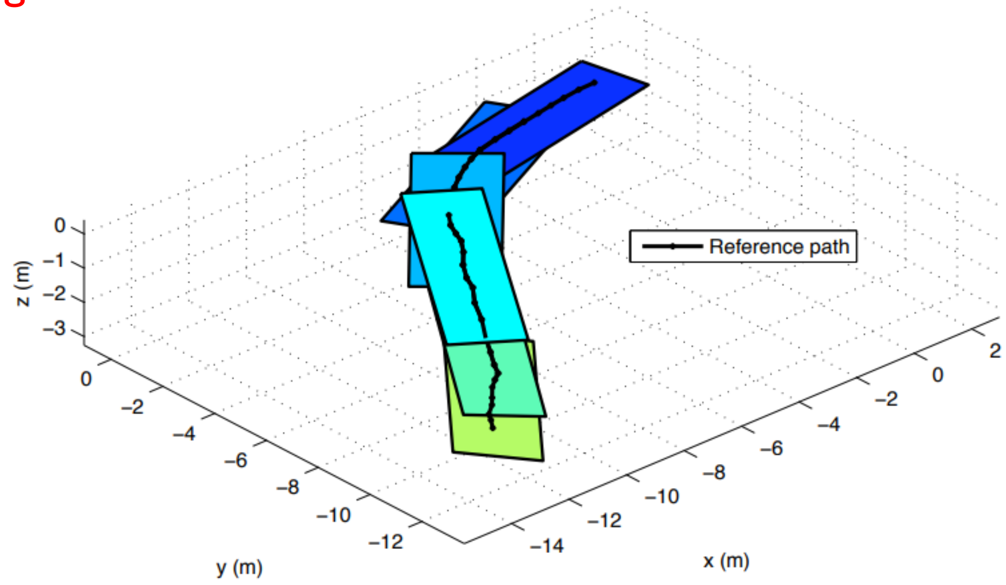


Fig. 5. A view of six overlapping submaps with the reference path plotted above.

Visual Path Following on a Manifold in Unstructured Three-Dimensional Terrain, Furgale & Barfoot, 2010

Visual Teach & Repeat

Key Idea #1: Manifold Map

Build local maps relative to the path. No global coordinate frame.

Key Idea #2: Visual Odometry

Given two consecutive images, how much has the camera moved? Relative motion.

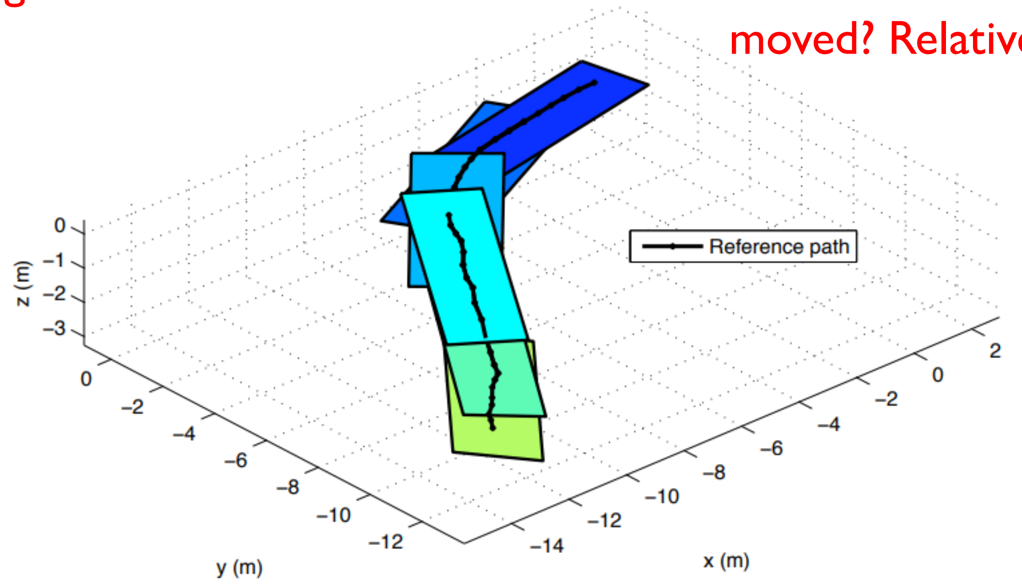


Fig. 5. A view of six overlapping submaps with the reference path plotted above.

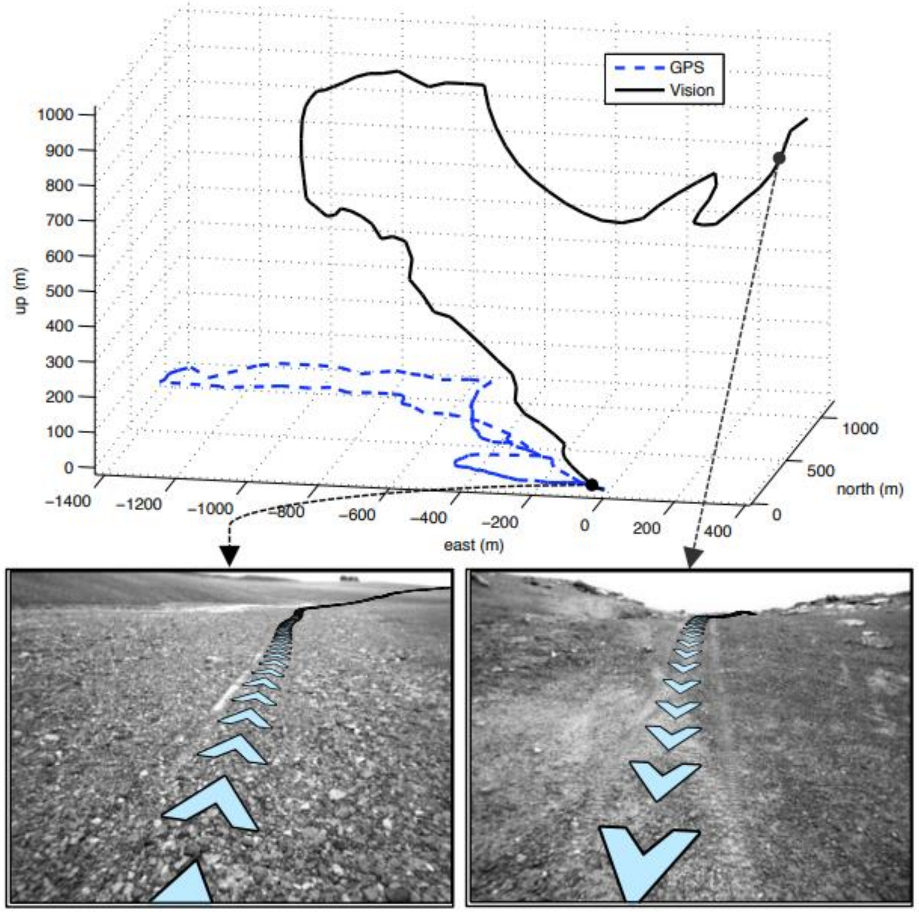


Fig. 6. The visual reconstruction of a five kilometer rover traverse plotted against GPS (Top). Although the reconstruction is wildly inaccurate at this scale, locally it is good enough to enable retracing of the route. The bottom images show views from either end of the path, with the reference path plotted as a series of chevrons. To the rover, the map is locally Euclidean.

Visual Path Following on a Manifold in Unstructured Three-Dimensional Terrain, Furgale & Barfoot, 2010

Visual Teach & Repeat

https://www.youtube.com/watch?v=_ZdBfU4xJnQ

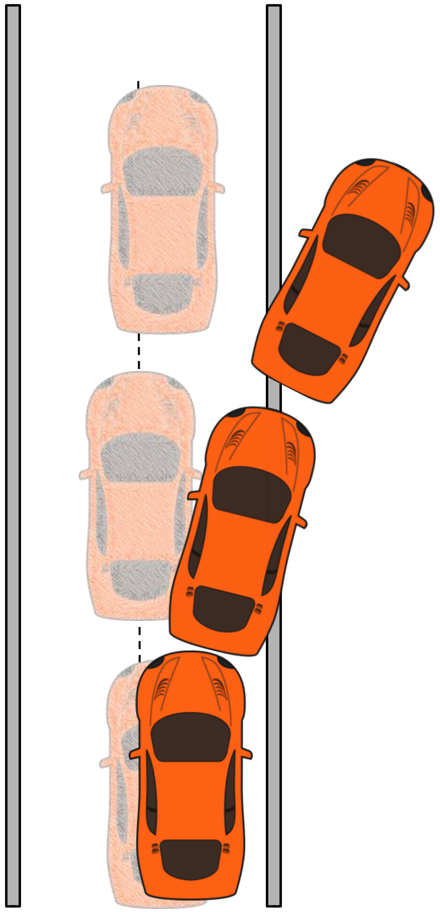
https://www.youtube.com/watch?v=9dN0wwXDuqo

Centimeter-level precision in tracking the demonstrated path over kilometers-long trails.

Today's agenda

- Administrivia
- Topics covered by the course
- Behavioral cloning
- Imitation learning
- Quiz about background and interests
- (Time permitting) Query the expert only when policy is uncertain

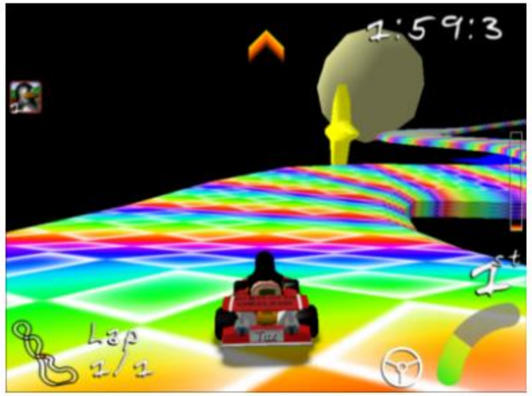
Back to Pomerleau



Test distribution is different from training distribution (covariate shift)

(Ross & Bagnell, 2010): How are we sure these errors are not due to overfitting or underfitting?

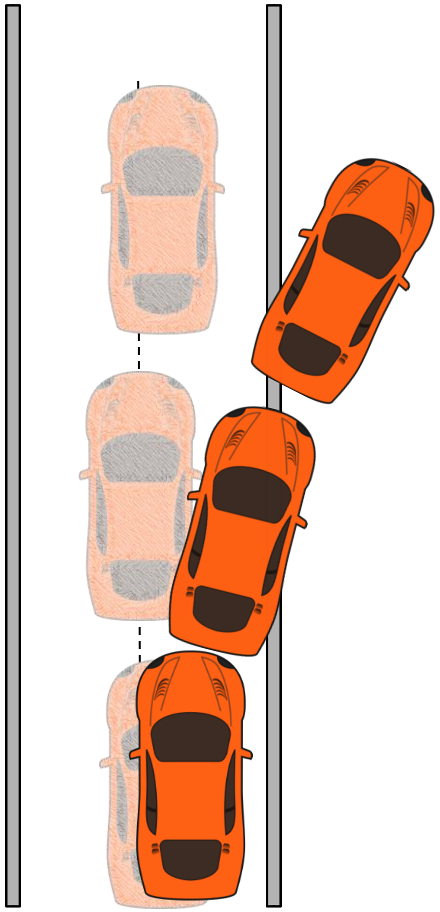
1. Maybe the network was too small (underfitting)
2. Maybe the dataset was too small and the network overfit it



Steering commands $\pi_\theta(s) = \theta^\top s$
where s are image features

Efficient reductions for imitation learning. Ross & Bagnell, AISTATS 2010.

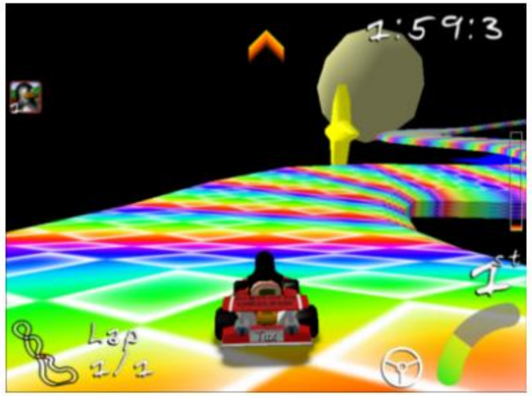
Back to Pomerleau



Test distribution is different from training distribution (covariate shift)

(Ross & Bagnell, 2010): How are we sure these errors are not due to overfitting or underfitting?

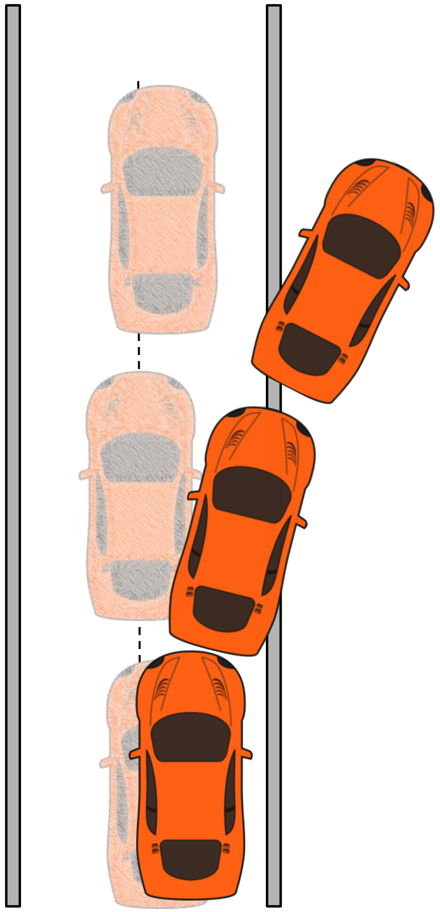
1. Maybe the network was too small (underfitting)
2. Maybe the dataset was too small and the network overfit it



Steering commands $\pi_\theta(s) = \theta^\top s$
where s are image features

It was not 1: they showed that even a linear policy can work well.
It was not 2: their error on held-out data was close to training error.

Imitation learning $\neq$ Supervised learning



Test distribution is different from training distribution (covariate shift)

(Ross & Bagnell, 2010): IL is a sequential decision-making problem.

- Your actions affect future observations/data.
- This is not the case in supervised learning

Supervised Learning

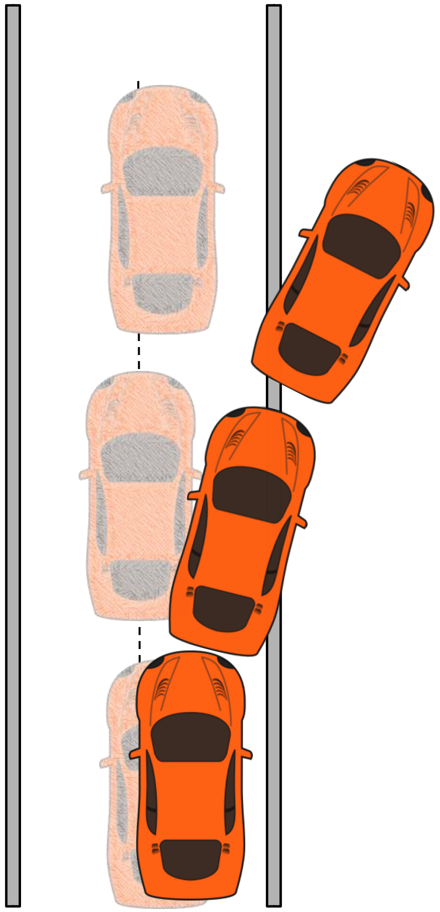
Assumes train/test data are i.i.d.

If expected training error is ϵ
Expected test error after T decisions

$$T\epsilon$$

Errors are independent

Imitation learning $\neq$ Supervised learning



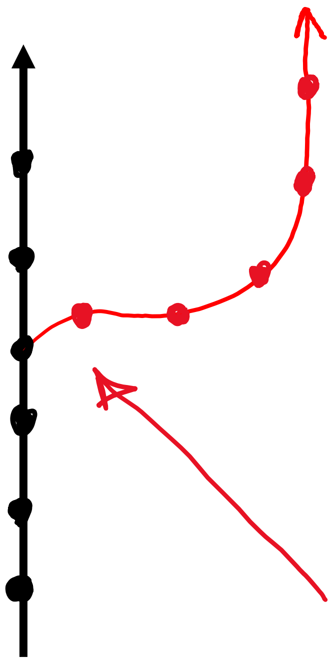
Test distribution is different from training distribution (covariate shift)

(Ross & Bagnell, 2010): IL is a sequential decision-making problem.

- Your actions affect future observations/data.
- This is not the case in supervised learning

Imitation Learning	←	**Supervised Learning**
Train/test data are not i.i.d.		Assumes train/test data are i.i.d.
If expected training error is ϵ Expected test error after T decisions is up to $T^2\epsilon$		If expected training error is ϵ Expected test error after T decisions $T\epsilon$
Errors compound		Errors are independent

Why do errors accumulate quadratically if we use a behavioral cloning policy?



first error at time t with probability ε leads to $T-t$ errors

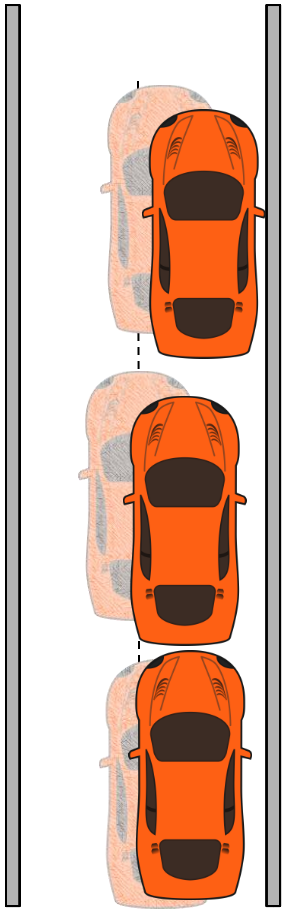
$$\mathbb{E}\left[\begin{array}{c}\#\text{ errors after}\\ T\text{ actions}\end{array}\right] \leq \varepsilon\left(T + (T-1) + \ldots + 1\right)$$

$$= \varepsilon \frac{T(T+1)}{2}$$

$$\leq \varepsilon T^2$$

Efficient reductions for imitation learning. Ross & Bagnell, AISTATS 2010.

DAgger



(Ross & Gordon & Bagnell, 2011): DAgger, or Dataset Aggregation

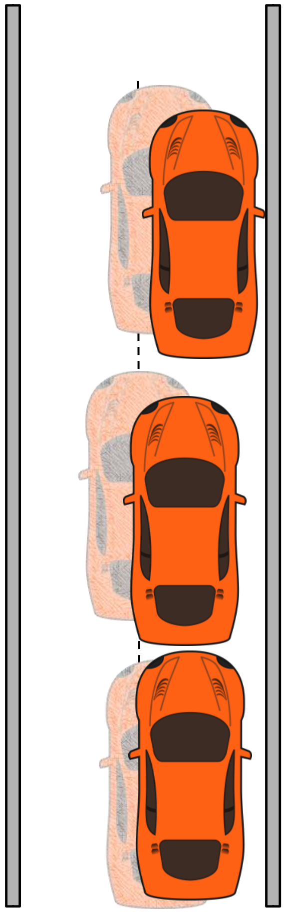
- **Imitation learning as interactive supervision**
- **Aggregate training data from expert with test data from execution**

Algorithm 1 DAgger

$D = \{(s, a)\}$ initial expert demonstrations
$\theta_1 \leftarrow$ train learner's policy parameters on D
for $i = 1 \ldots N$ **do**
    Execute learner's policy π_{θ_i}, get visited states $S_{\theta_i} = \{s_0, \ldots, s_T\}$
    Query the expert at those states to get actions $A = \{a_0, \ldots, a_T\}$
    Aggregate dataset $D = D \cup \{(s, a) \mid s \in S_{\theta_i},\ a \in A\}$
    Train learner's policy $\pi_{\theta_{i+1}}$ on dataset D
Return one of the policies π_{θ_i} that performs best on validation set

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. Ross, Gordon, Bagnell, AISTATS 2010.

DAgger



(Ross & Gordon & Bagnell, 2011): DAgger, or Dataset Aggregation

- **Imitation learning as interactive supervision**
- **Aggregate training data from expert with test data from execution**

Imitation Learning via DAgger

Train/test data are not i.i.d.

If expected training error on aggr. dataset is ϵ
Expected test error after T decisions is

$$O(T\epsilon)$$

Errors do not compound

Supervised Learning

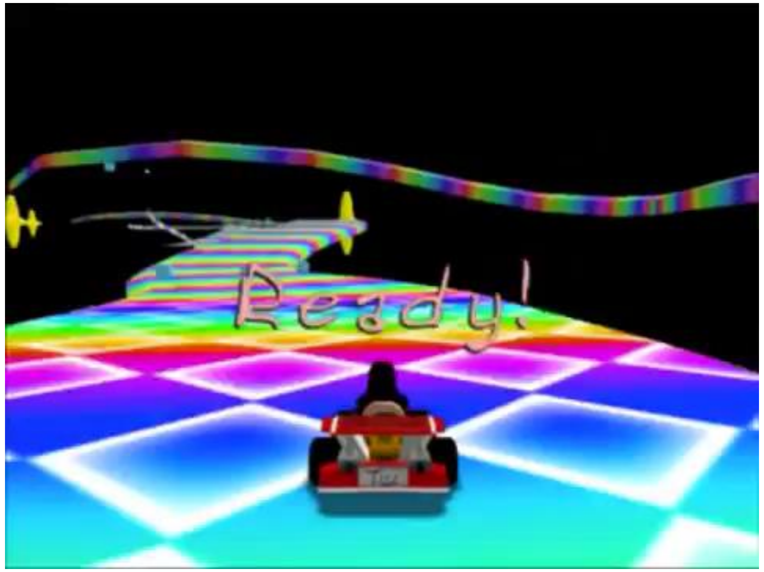
Assumes train/test data are i.i.d.

If expected training error is ϵ
Expected test error after T decisions

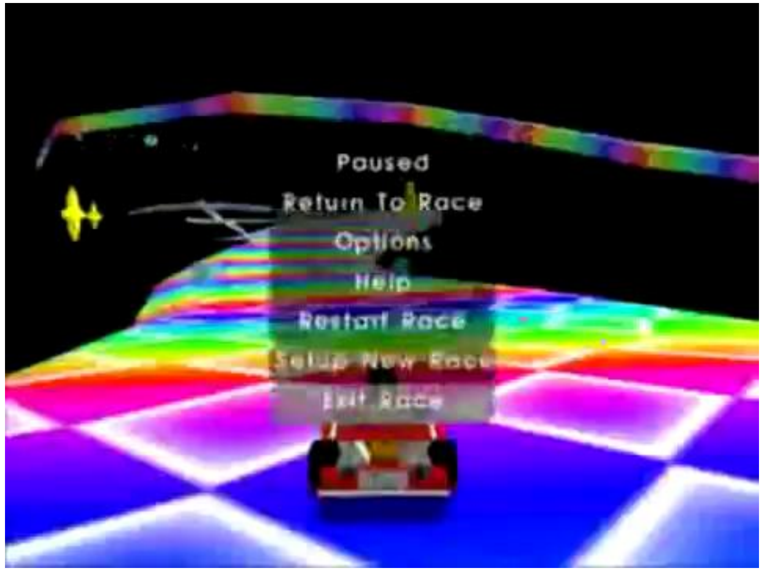
$$T\epsilon$$

Errors are independent

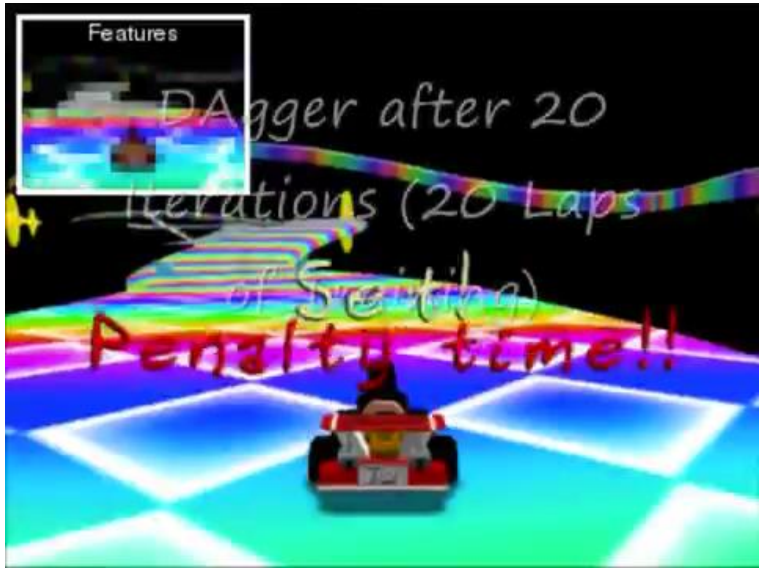
DAgger



Initial expert trajectories



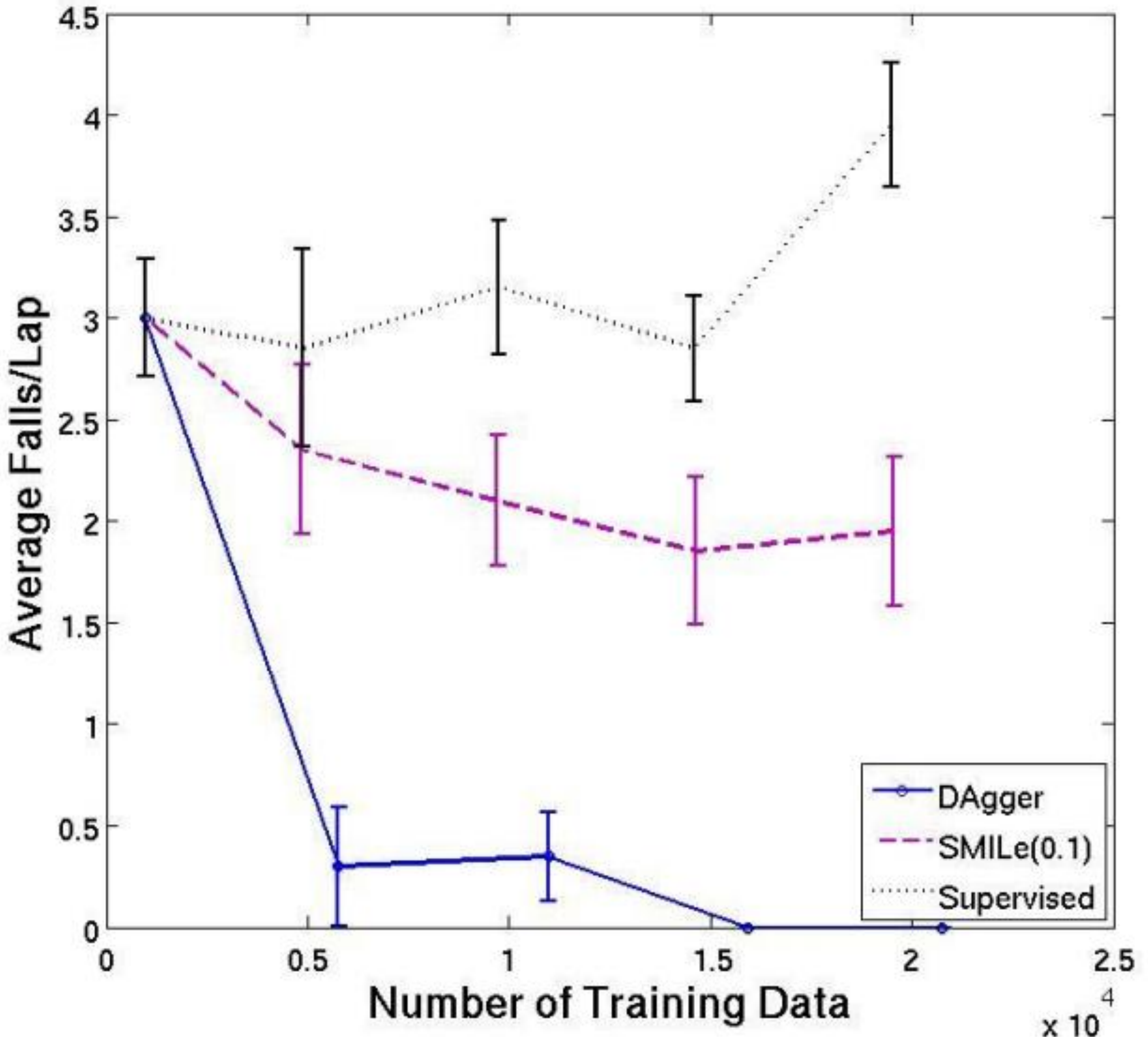
Supervised learning



DAgger

https://www.youtube.com/watch?v=V00npNnWzSU

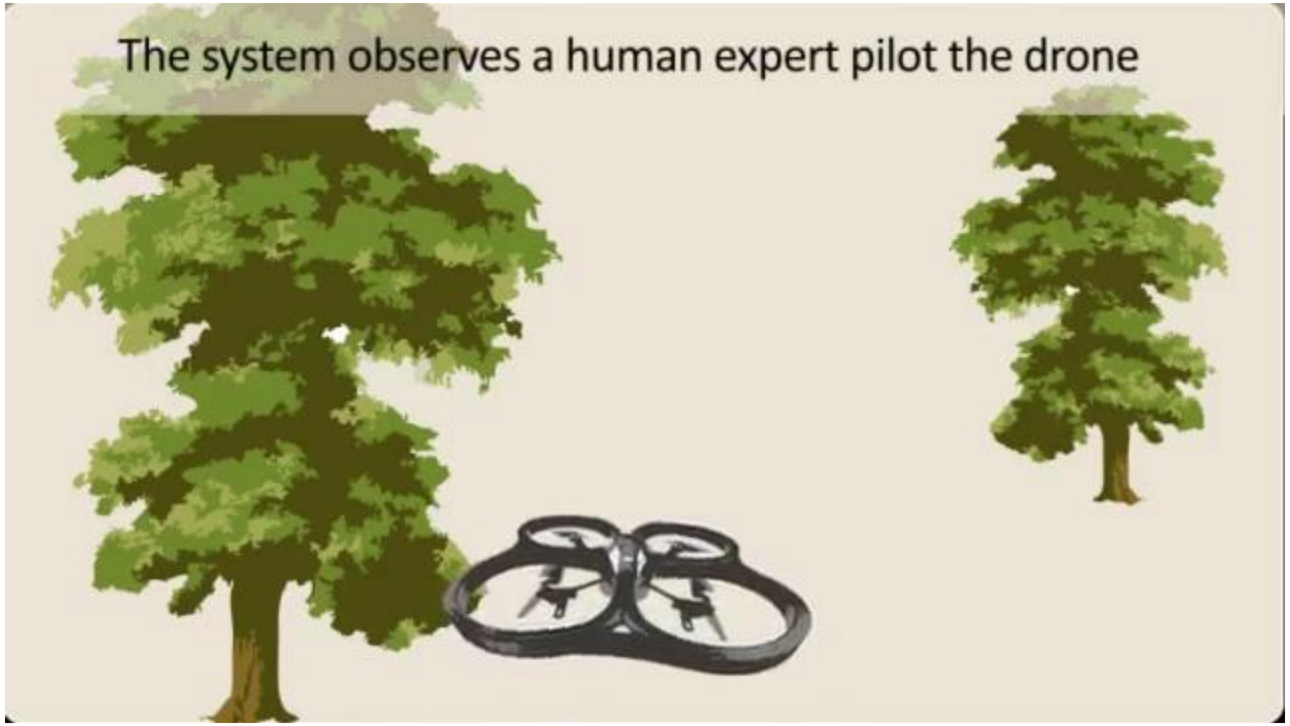
DAgger


Average Falls/Lap
Number of Training Data
x 10^4
DAgger
SMILe(0.1)
Supervised
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0
0.5
1
1.5
2
2.5

DAgger

Q: Any drawbacks of using it in a robotics setting?

DAgger



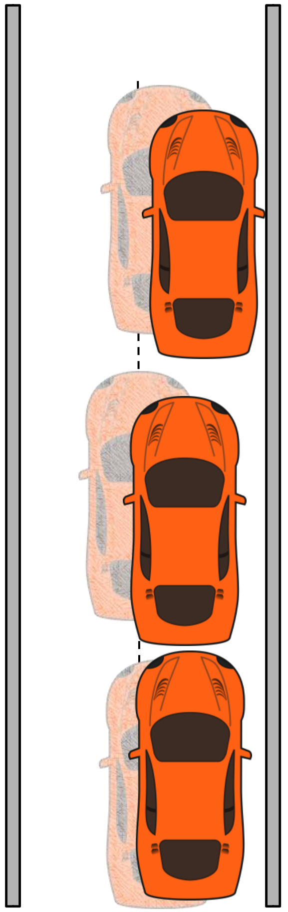
https://www.youtube.com/watch?v=hNsP6-K3Hn4

Learning Monocular Reactive UAV Control in Cluttered Natural Environments, Ross et al, 2013

Today’s agenda

- Administrivia
- Topics covered by the course
- Behavioral cloning
- Imitation learning
- Quiz about background and interests
- (Time permitting) Query the expert only when policy is uncertain

DAgger: Assumptions for theoretical guarantees



Strongly convex loss
No-regret online learner

(Ross & Gordon & Bagnell, 2011): DAgger, or Dataset Aggregation

- Imitation learning as interactive supervision
- Aggregate training data from expert with test data from execution

Imitation Learning via DAgger	**Supervised Learning**
Train/test data are not i.i.d.	Assumes train/test data are i.i.d.
If expected training error on aggr. dataset is ϵ Expected test error after T decisions is	If expected training error is ϵ Expected test error after T decisions
$O(T\epsilon)$	$T\epsilon$
Errors do not compound	Errors are independent

Appendix 1: No-Regret Online Learners

Intuition: No matter what the distribution of input data, your online policy/classifier will do asymptotically as well as the best-in-hindsight policy/classifier.

$$r_N = \frac{1}{N}\sum_{i=1}^{N} L_i(\theta_i) - \min_{\theta\in\Theta}\left[\frac{1}{N}\sum_{i=1}^{N} L_i(\theta)\right]$$

Policy has access to data up to round i

Policy has access to data up to round N

No-regret: $\lim_{N\to\infty} r_N = 0$

Appendix 1: No-Regret Online Learners

Intuition: No matter what the distribution of input data, your online policy/classifier will do asymptotically as well as the best-in-hindsight policy/classifier.

$$r_N = \frac{1}{N}\sum_{i=1}^{N} L_i(\theta_i) - \min_{\theta\in\Theta}\left[\frac{1}{N}\sum_{i=1}^{N} L_i(\theta)\right]$$

Policy has access to data up to round i

Policy has access to data up to round N

No-regret: $\lim_{N\to\infty} r_N = 0$

Another way to say this: a no-regret online algorithm is one that outputs a sequence of policies $\pi_1, \ldots, \pi_N$ such that the average loss with respect to the best-in-hindsight policy goes to 0 as $N \to \infty$

DAgger is a no-regret online learning algorithm

Appendix 1: No-Regret Online Learners

Intuition: No matter what the distribution of input data, your online policy/classifier will do asymptotically as well as the best-in-hindsight policy/classifier.

$$r_N = \frac{1}{N}\sum_{i=1}^{N} L_i(\theta_i) - \min_{\theta\in\Theta}\left[\frac{1}{N}\sum_{i=1}^{N} L_i(\theta)\right]$$

Policy has access to data up to round i

Policy has access to data up to round N

No-regret: $\lim_{N\to\infty} r_N = 0$

We can see Dagger as an adversarial game between the imitation learner (policy) and an adversary (environment):

Learner

Adversary

policy π_{θ_0} → loss $L_0(\theta_0)$

policy π_{θ_1} ←

→ loss $L_1(\theta_1)$

policy π_{θ_2} ←

⋮

Appendix 2: Why do behavioral cloning errors accumulate quadratically?

The traditional approach to imitation learning trains a classifier that learns to replicate the expert's policy under the state distribution induced by the expert. Formally, the traditional approach minimizes 0-1 loss under distribution d_{π^*}: $\hat{\pi} = \text{argmin}_{\pi \in \Pi} \mathbb{E}_{s \sim d_{\pi^*}}(e_\pi(s))$. Now assume that the resulting classifier (policy) $\hat{\pi}$ makes a mistake with probability ϵ under d_{π^*}, i.e. $\mathbb{E}_{s \sim d_{\pi^*}}(e_{\hat{\pi}}(s)) = \epsilon$. Then we have the following guarantee:

Theorem 2.1. *Let $\hat{\pi}$ be such that $\mathbb{E}_{s \sim d_{\pi^*}}[e_{\hat{\pi}}(s)] \leq \epsilon$. Then $J(\hat{\pi}) \leq J(\pi^*) + T^2\epsilon$. (Proof in Supplementary Material)*

Proof of Theorem 2.1

Let $\epsilon_i = \mathbb{E}_{s \sim d^i_{\pi^*}}[e_{\hat{\pi}}(s)]$ for $i = 1, 2, \ldots, T$ the expected 0-1 loss at time i of $\hat{\pi}$, such that $\epsilon = \frac{1}{T}\sum_{i=1}^T \epsilon_i$. Note that ϵ_t corresponds to the probability that $\hat{\pi}$ makes a mistake under distribution $d^t_{\pi^*}$. Let p_t represent the probability $\hat{\pi}$ hasn't made a mistake (w.r.t. π^*) in the first t-step, and d_t the distribution of state $\hat{\pi}$ is in at time t conditioned on the fact it hasn't made a mistake so far. If d'_t represents the distribution of states at time t obtained by following π^* but conditioned on the fact that $\hat{\pi}$ made at least one mistake in the first $t-1$ visited states. Then $d^t_{\pi^*} = p_{t-1}d_t + (1 - p_{t-1})d'_t$. Now at time t, the expected cost of $\hat{\pi}$ is at most 1 if it has made a mistake so far, or $\mathbb{E}_{s \sim d_t}(C_{\hat{\pi}}(s))$ if it hasn't make a mistake yet. So $J(\hat{\pi}) \leq \sum_{t=1}^T [p_{t-1}\mathbb{E}_{s \sim d_t}(C_{\hat{\pi}}(s)) + (1 - p_{t-1})]$. Let e_t and e'_t represent the probability of mistake of $\hat{\pi}$ in distribution d_t and d'_t. Then $\mathbb{E}_{s \sim d_t}(C_{\hat{\pi}}(s)) \leq \mathbb{E}_{s \sim d_t}(C_{\pi^*}(s)) + e_t$, and since $\epsilon_t = p_{t-1}e_t + (1 - p_{t-1})e'_t$, then $p_{t-1}e_t \leq \epsilon_t$. Additionnally since $p_t = (1 - e_t)p_{t-1}$, $p_t \geq p_{t-1} - \epsilon_t \geq 1 - \sum_{i=1}^t \epsilon_i$, i.e. $1 - p_t \leq \sum_{i=1}^t \epsilon_i$. Finally note that $J(\pi^*) = \sum_{t=1}^T [p_{t-1}\mathbb{E}_{s \sim d_t}(C_{\pi^*}(s)) + (1 - p_{t-1})\mathbb{E}_{s \sim d'_t}(C_{\pi^*}(s))]$, so that $\sum_{t=1}^T p_{t-1}\mathbb{E}_{s \sim d_t}(C_{\pi^*}(s)) \leq J(\pi^*)$. Using these facts we obtain:

$$\begin{aligned}
J(\hat{\pi}) &\leq \textstyle\sum_{t=1}^T [p_{t-1}\mathbb{E}_{s \sim d_t}(C_{\hat{\pi}}(s)) + (1 - p_{t-1})] \\
&\leq \textstyle\sum_{t=1}^T [p_{t-1}\mathbb{E}_{s \sim d_t}(C_{\pi^*}(s)) + p_{t-1}e_t + (1 - p_{t-1})] \\
&\leq J(\pi^*) + \textstyle\sum_{t=1}^T \sum_{i=1}^t \epsilon_i \\
&\leq J(\pi^*) + T\textstyle\sum_{t=1}^T \epsilon_t \\
&= J(\pi^*) + T^2\epsilon
\end{aligned}$$

Efficient reductions for imitation learning. Ross & Bagnell, AISTATS 2010.

Appendix 3: Types of Uncertainty & Query-Efficient Imitation

Let's revisit the two main ideas from query-efficient imitation:

1. DropoutDAgger:
   Keep an ensemble of learner policies, and only query the expert when they significantly disagree

2. SHIV, SafeDagger, MMD-IL:
   (Roughly) Query expert only if input is too close to the decision boundary of the learner's policy

Need to review a few concepts about different types of uncertainty.

Biased Coin

$$p(\text{heads}_3 \mid \underbrace{\text{heads}_1, \text{heads}_2}_{\text{observations}}) \;= ?$$

Biased Coin

$$p(\text{heads}_3 \mid \text{heads}_1, \text{heads}_2) \;=\; \int p(\text{heads}_3 \mid \theta) \underbrace{p(\theta \mid \text{heads}_1, \text{heads}_2)}_{\text{how biased is the coin?}} d\theta$$

Biased Coin

$$p(\text{heads}_3 \mid \text{heads}_1, \text{heads}_2) = \int p(\text{heads}_3 \mid \theta)\underbrace{p(\theta \mid \text{heads}_1, \text{heads}_2)}_{\text{how biased is the coin?}}d\theta$$

Induces uncertainty in the model, or epistemic uncertainty,
which asymptotically goes to 0 with infinite observations

Biased Coin

$$p(\text{heads}_3 \mid \text{heads}_1, \text{heads}_2) \;=\; \int p(\text{heads}_3 \mid \theta) p(\theta \mid \text{heads}_1, \text{heads}_2) d\theta$$

Q: Even if you eventually discover the true model, can you predict if the next flip will be heads?

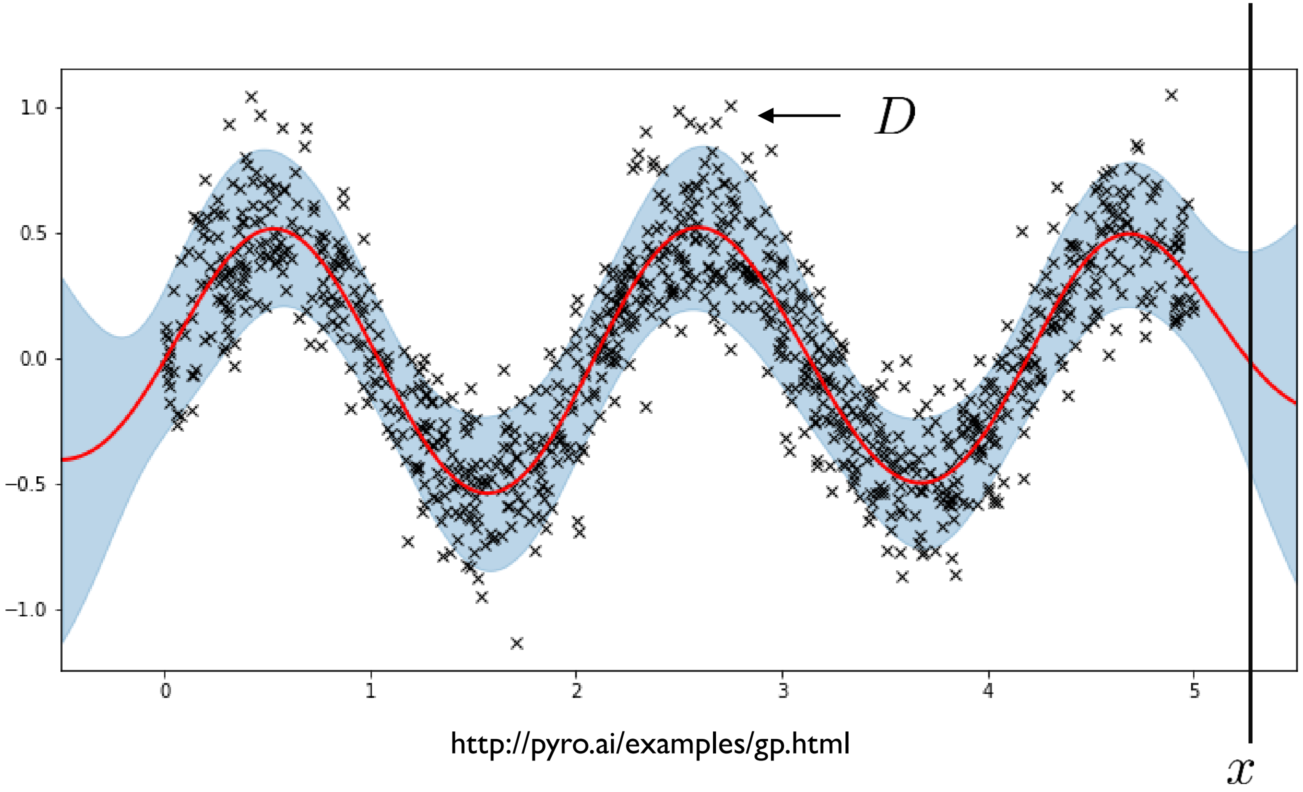
Biased Coin

$$p(\text{heads}_3 \mid \text{heads}_1, \text{heads}_2) \; = \int p(\text{heads}_3 \mid \theta) p(\theta \mid \text{heads}_1, \text{heads}_2) d\theta$$

Q: Even if you eventually discover the true model, can you predict if the next flip will be heads?

A: No, there is irreducible uncertainty / observation noise in the system. This is called aleatoric uncertainty.

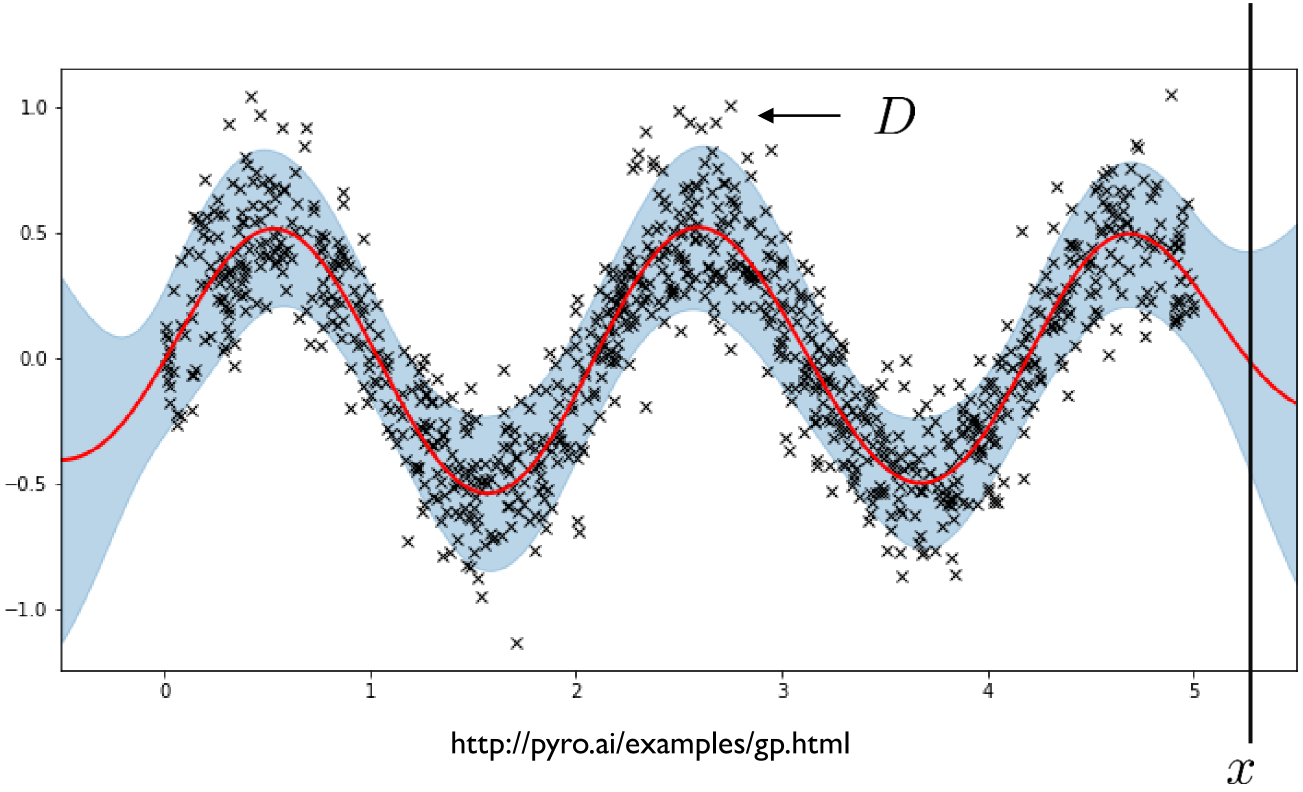
Gaussian Process Regression



http://pyro.ai/examples/gp.html

$p(y|x, D) = ?$

Gaussian Process Regression



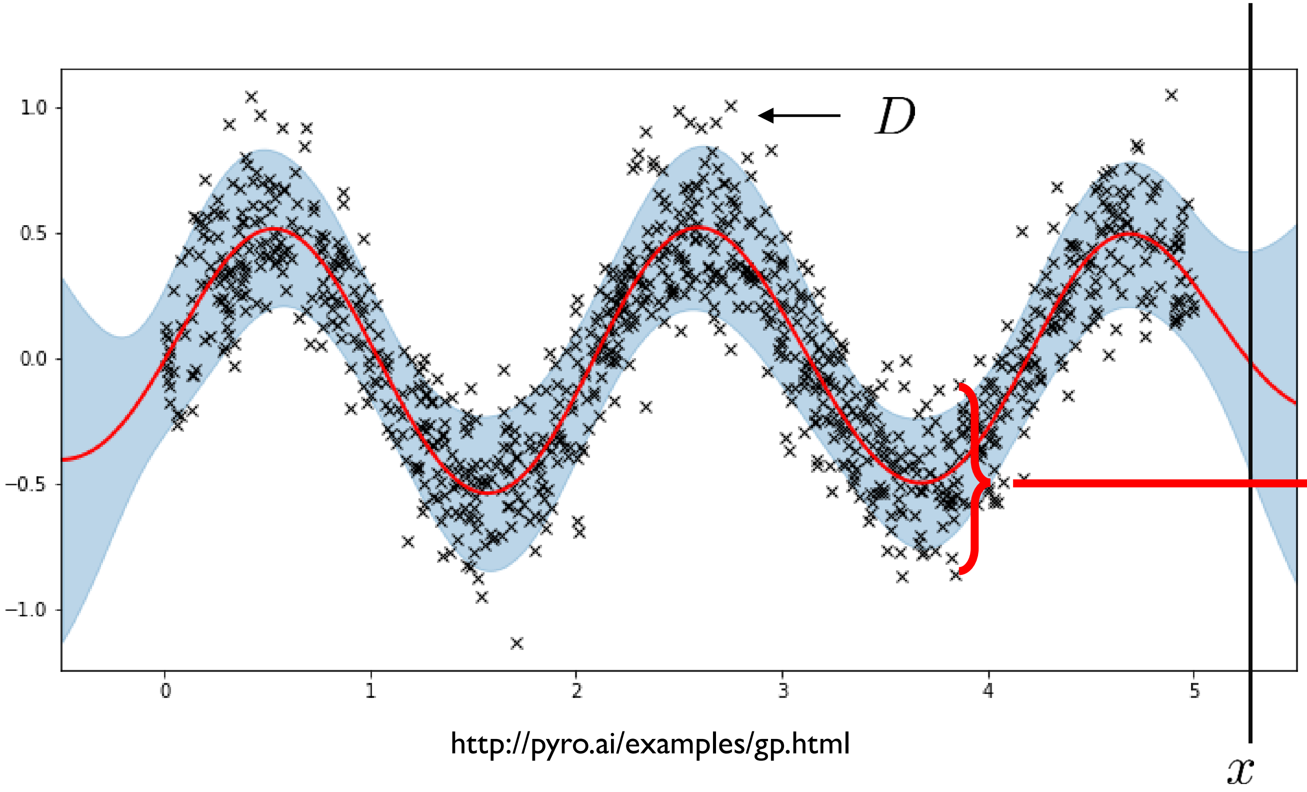
http://pyro.ai/examples/gp.html

$$p(y|x, D) = \int p(y|f)\; p(f|x, D) df$$

$f|x, D \sim \mathcal{N}(f; 0, K)$ Zero mean prior over functions

$y|f \sim \mathcal{N}(y; f, \sigma^2)$ Noisy observations

Gaussian Process Regression



http://pyro.ai/examples/gp.html

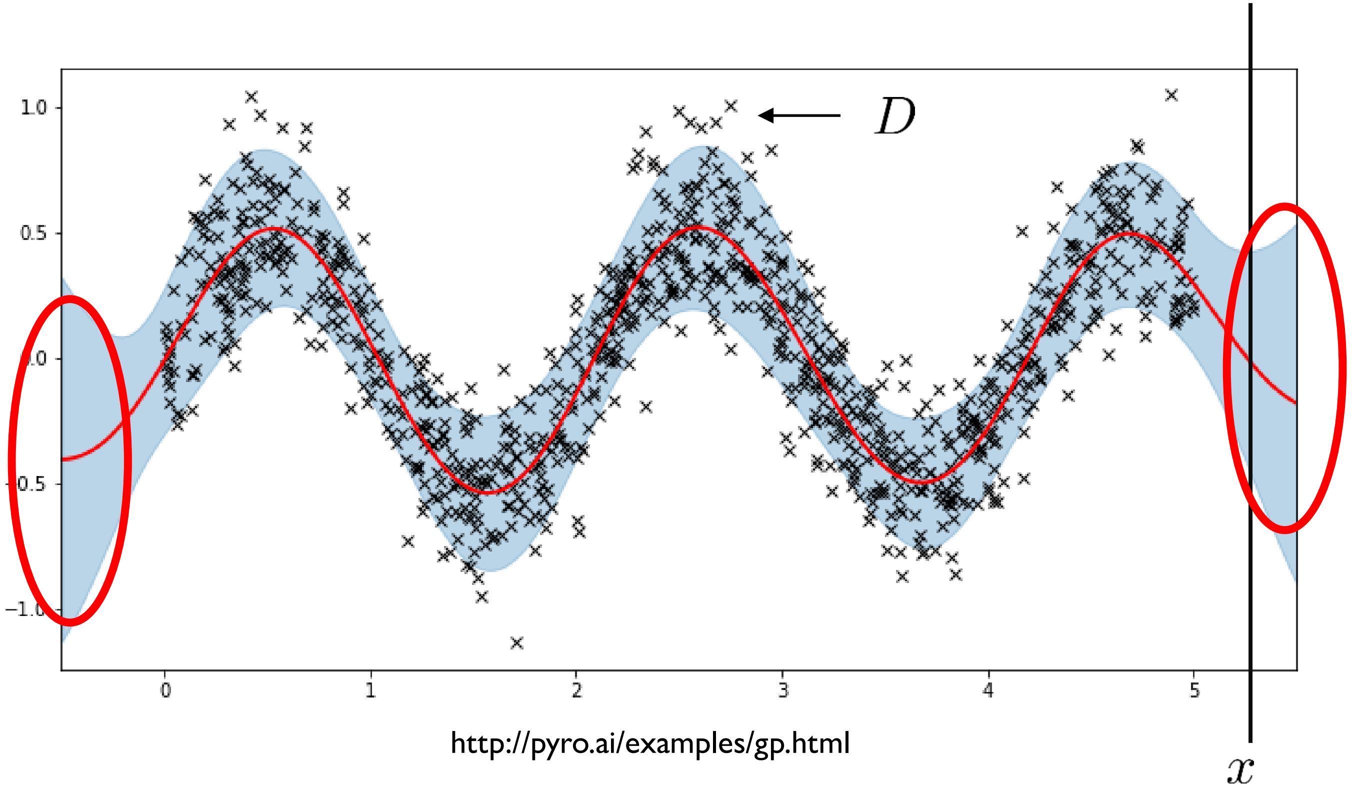
No matter how much data we get, this observation noise will not go to zero

$$p(y|x, D) = \int p(y|f)\ p(f|x, D)df$$

$f|x, D \sim \mathcal{N}(f; 0, K)$ Zero mean prior over functions

$y|f \sim \mathcal{N}(y; f, \sigma^2)$ Noisy observations

Gaussian Process Regression



http://pyro.ai/examples/gp.html

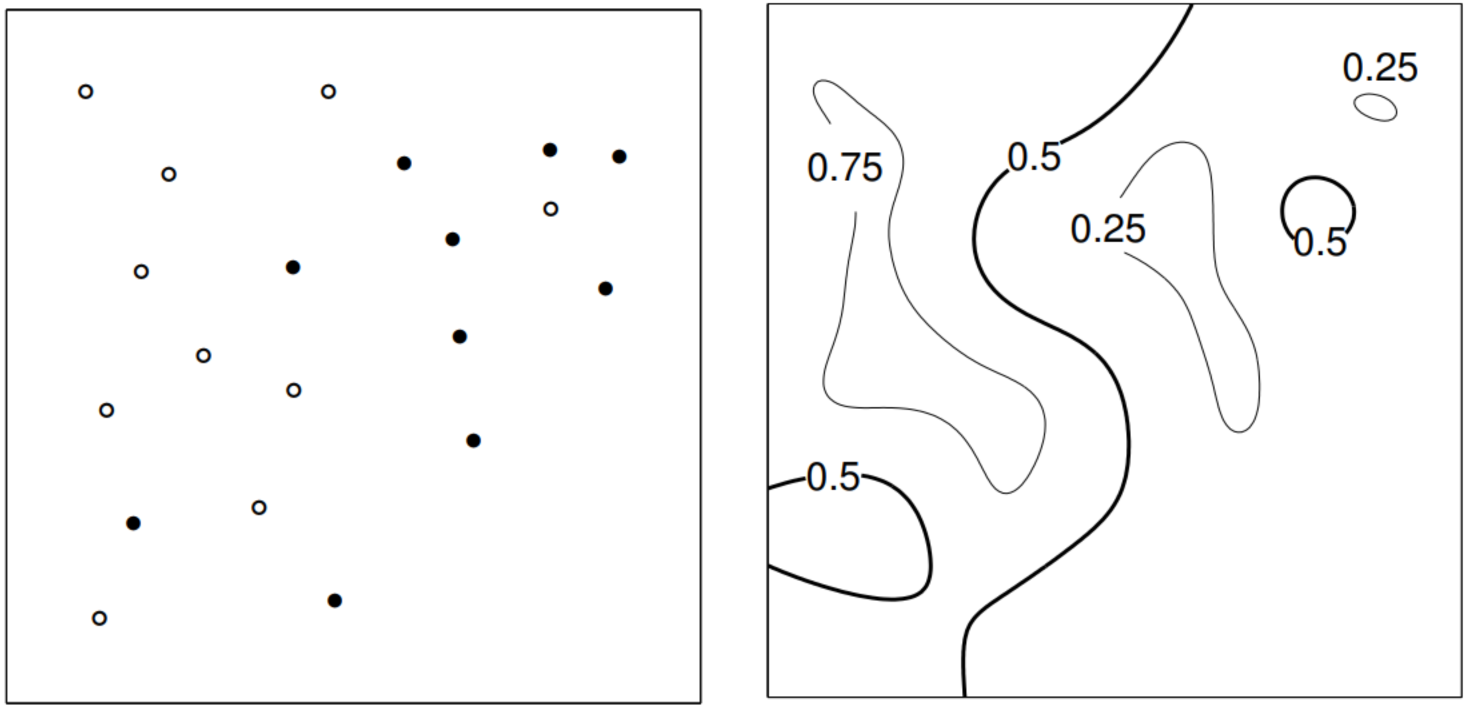
If we get data here we can reduce model / epistemic uncertainty

$$p(y|x, D) = \int p(y|f)\, p(f|x, D)df$$

$f|x, D \sim \mathcal{N}(f; 0, K)$ Zero mean prior over functions

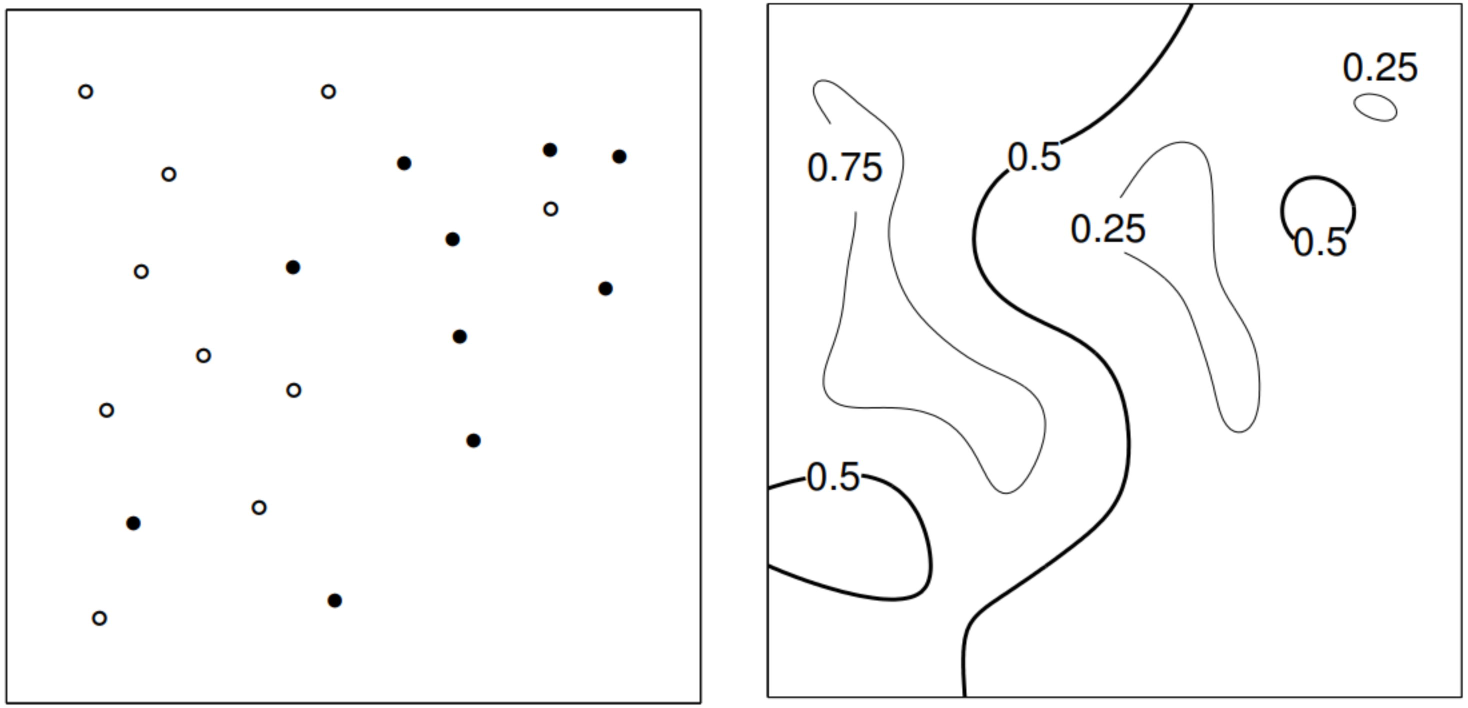
$y|f \sim \mathcal{N}(y; f, \sigma^2)$ Noisy observations

Gaussian Process Classification



Gaussian Processes for Machine Learning, chapter 2

Gaussian Process Classification vs SVM



Gaussian Processes for Machine Learning, chapter 2

GP handles uncertainty in f by averaging
while SVM considers only best f for classification.

Model Uncertainty in Neural Networks

Want $p(y|x, D) = \int p(y|x, f)\, p(f|D) df$

But easier to control network weights $p(y|x, D) = \int p(y|x, w)\, p(w|D) dw$

Model Uncertainty in Neural Networks

Want $p(y|x, D) = \int p(y|x, f)\, p(f|D)df$

But easier to control network weights $p(y|x, D) = \int p(y|x, w) p(w|D) dw$

How do we represent posterior over network weights?
How do we quickly sample from it?

Model Uncertainty in Neural Networks

Want $p(y|x, D) = \int p(y|x, f)\, p(f|D) df$

But easier to control network weights $p(y|x, D) = \int p(y|x, w) p(w|D) dw$

How do we represent posterior over network weights?
How do we quickly sample from it?

Main ideas:

1. Use an ensemble of networks trained on different copies of D (bootstrap method)

2. Use an approximate distribution over weights (Dropout, Bayes by Backprop, …)

3. Use MCMC to sample weights

Model Uncertainty in Neural Networks

Want $p(y|x, D) = \int p(y|x, f)\, p(f|D) df$

But easier to control network weights $p(y|x, D) = \int p(y|x, w) p(w|D) dw$

How do we represent posterior over network weights?
How do we quickly sample from it?

Main ideas:

1. Use an ensemble of networks trained on different copies of D (bootstrap method)

2. Use an approximate distribution over weights (Dropout, Bayes by Backprop, …)

$\text{argmin}_\theta\ KL(q_\theta(w)||p(w|D))$

3. Use MCMC to sample weights

Model Uncertainty in Neural Networks

Want $p(y|x, D) = \int p(y|x, f)\, p(f|D) df$

But easier to control network weights $p(y|x, D) = \int p(y|x, w) p(w|D) dw$

How do we represent posterior over network weights?
How do we quickly sample from it?

Main ideas:

1. Use an ensemble of networks trained on different copies of D (bootstrap method)
2. Use an approximate distribution over weights (Dropout, Bayes by Backprop, …)
3. Use MCMC to sample weights

$$q(y|x) = \int p(y|x, w)\, q_{\theta^*}(w) dw$$

Variational inference

$$\theta^* = \text{argmin}_\theta\, KL(q_\theta(w)||p(w|D))$$

Model Uncertainty in Neural Networks

Want $p(y|x, D) = \int p(y|x, f)\, p(f|D) df$

But easier to control network weights $p(y|x, D) = \int p(y|x, w) p(w|D) dw$

approximates

$$q(y|x) = \int p(y|x, w)\; q_{\theta^*}(w) dw$$

Variational inference

$$\theta^* = \text{argmin}_\theta\; KL(q_\theta(w)||p(w|D))$$

How do we represent posterior over network weights?
How do we quickly sample from it?

Main ideas:

1. Use an ensemble of networks trained on different copies of D (bootstrap method)
2. Use an approximate distribution over weights (Dropout, Bayes by Backprop, …)
3. Use MCMC to sample weights