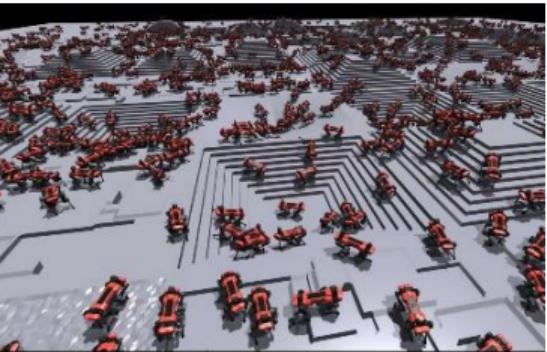
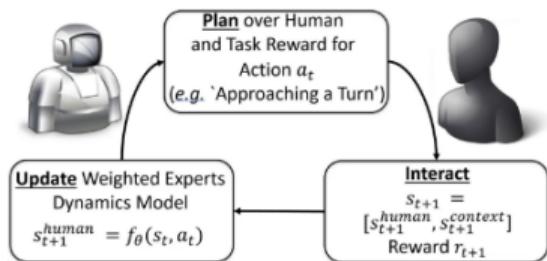


# EC518: Robot Learning



Eshed Ohn-Bar



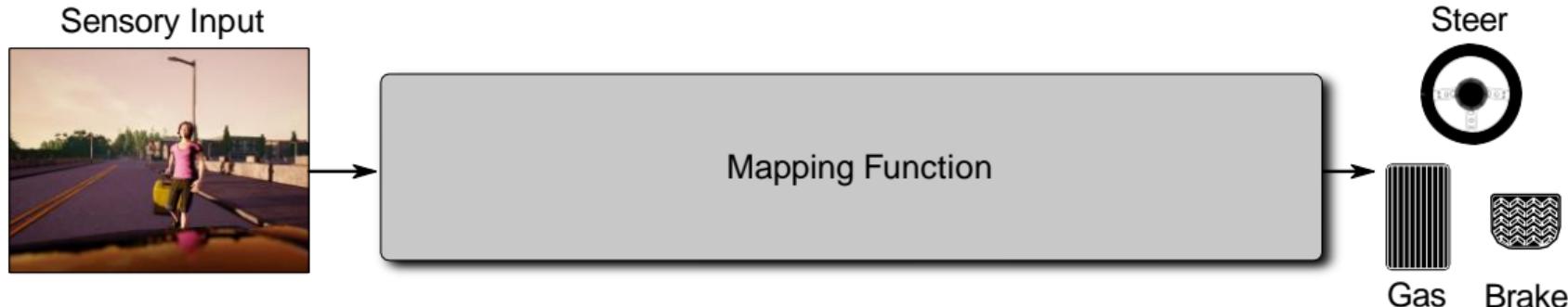
Sep. 13, 2023



## Announcements

- HW1 difficult, will take lots of time, plan appropriately.
- Questionnaires – those received mentioned experience with python and ML.
- Office Hours are on Mondays after class

# Approaches for Sensorimotor Navigation



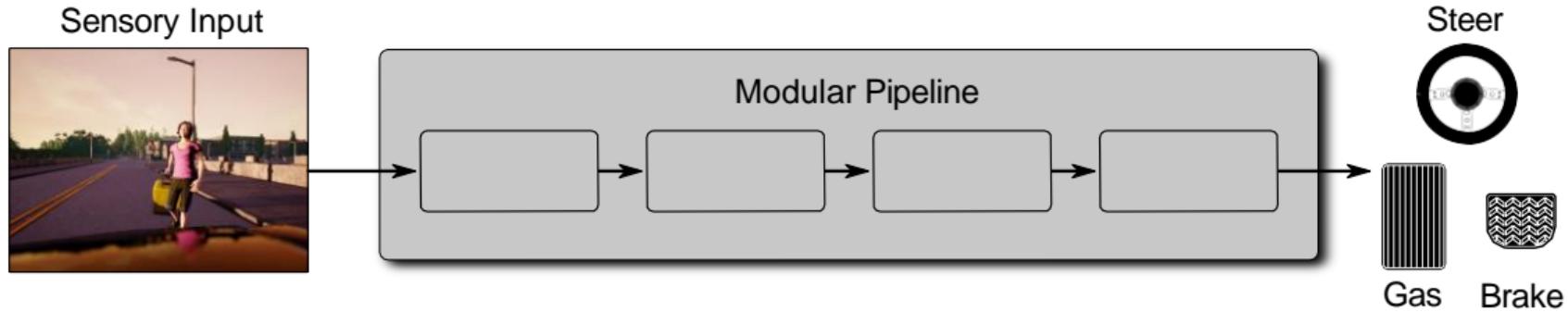
## Major Paradigms:

- Modular Pipelines
- End-to-End Learning (Deep Imitation Learning, Reinforcement Learning)
- Direct Perception

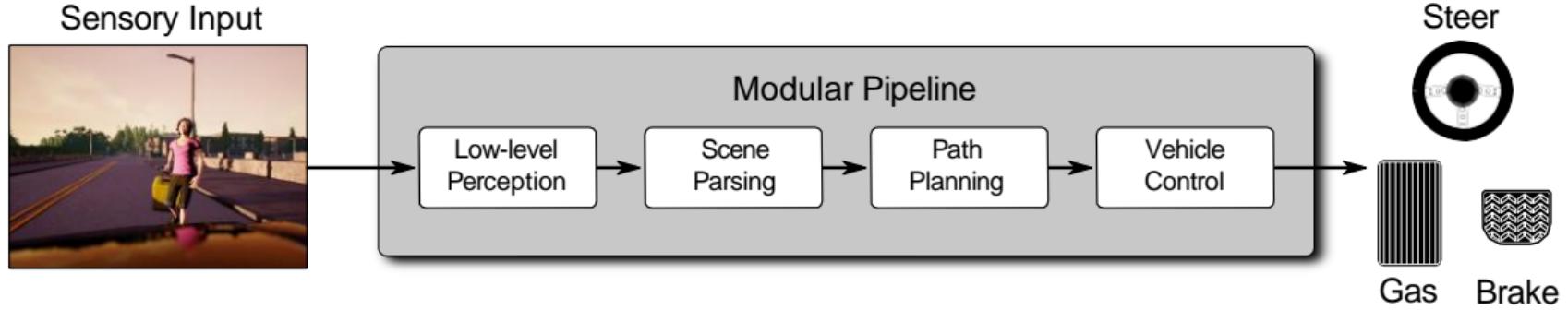
# Modular Pipeline



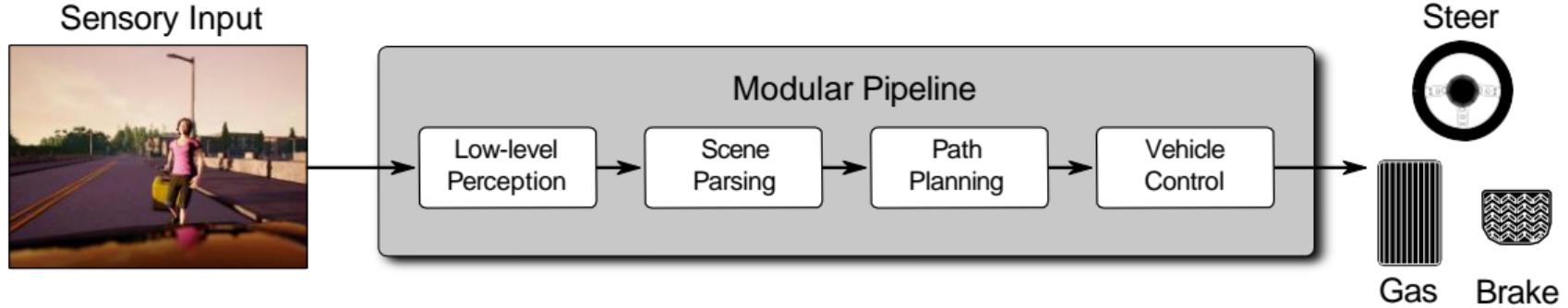
## Modular Pipeline



# Modular Pipeline



# Modular Pipeline



## Examples:

- [Montemerlo et al., JFR 2008]
- [Urmson et al., JFR 2008]
- Waymo, Uber, Tesla, Zoox, ...



Videocore

HDMI

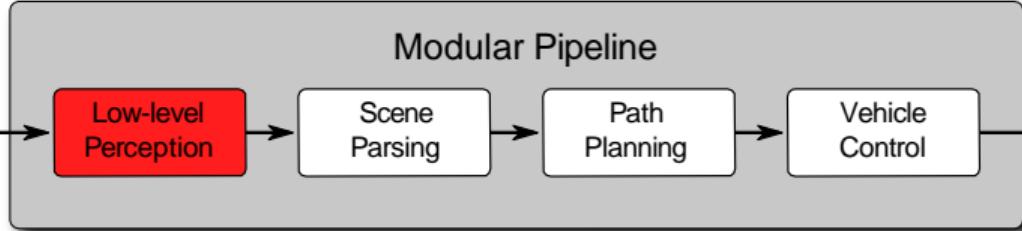
Audio : LPCM

Video : 1080p 24Hz



# Modular Pipeline

Sensory Input



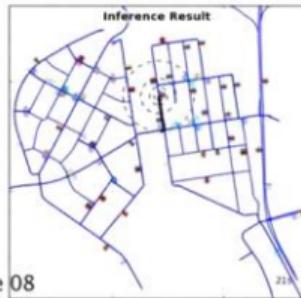
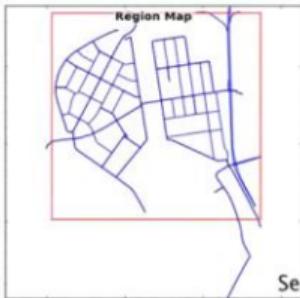
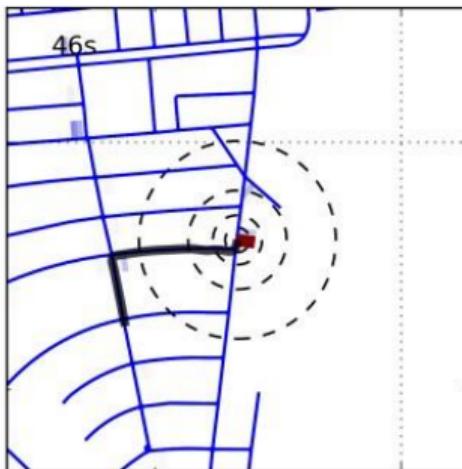
Steer



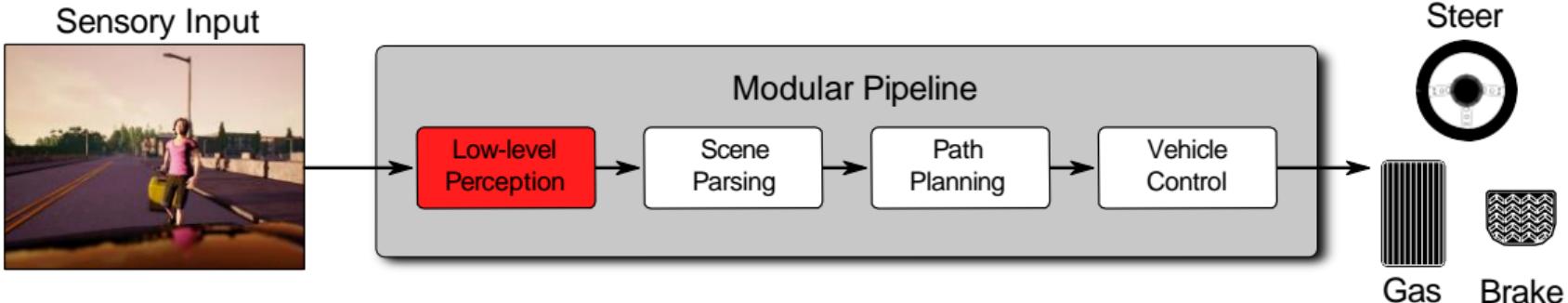
Gas



Brake

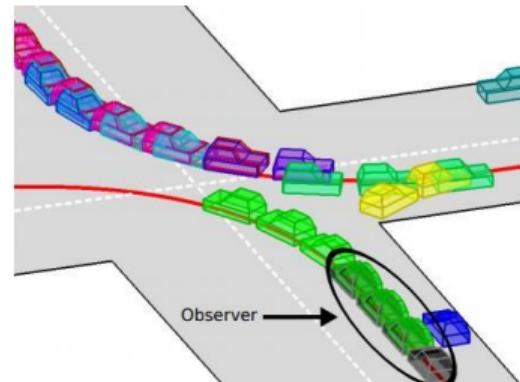
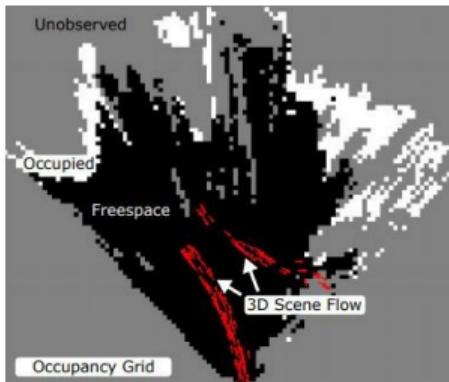
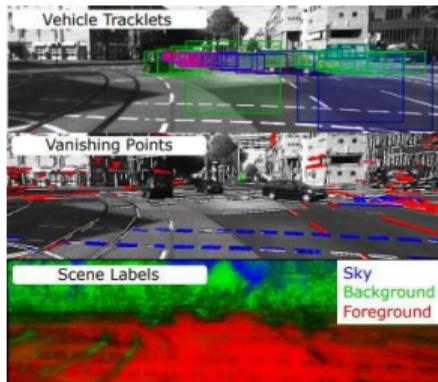
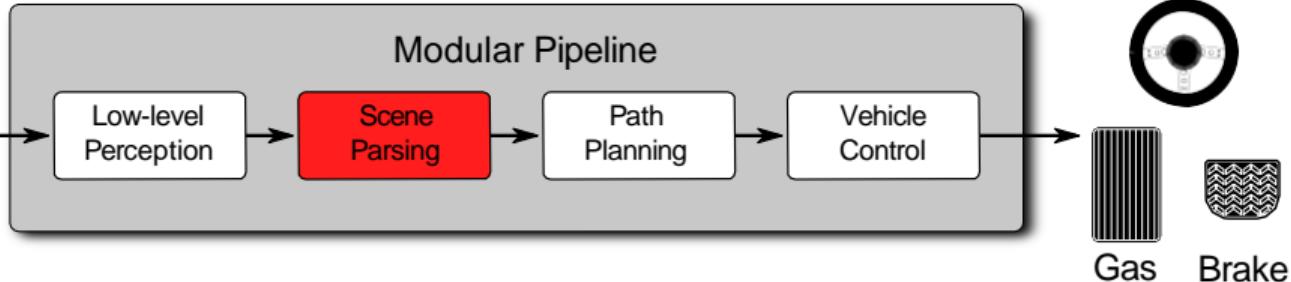


# Modular Pipeline



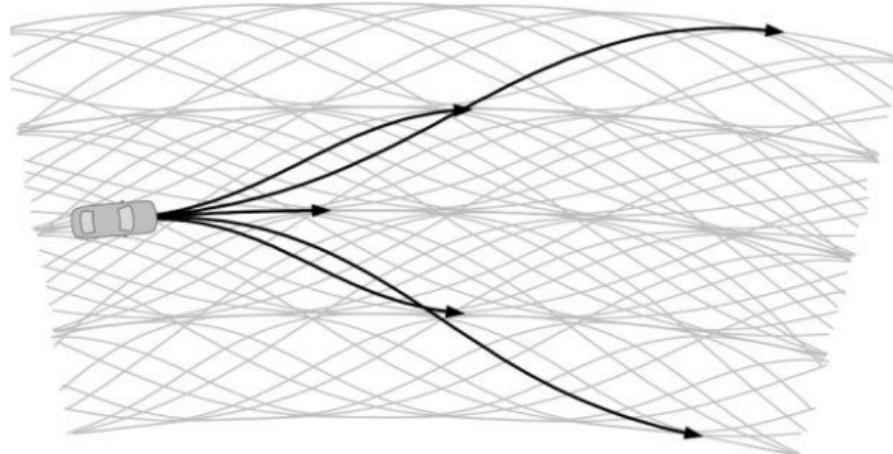
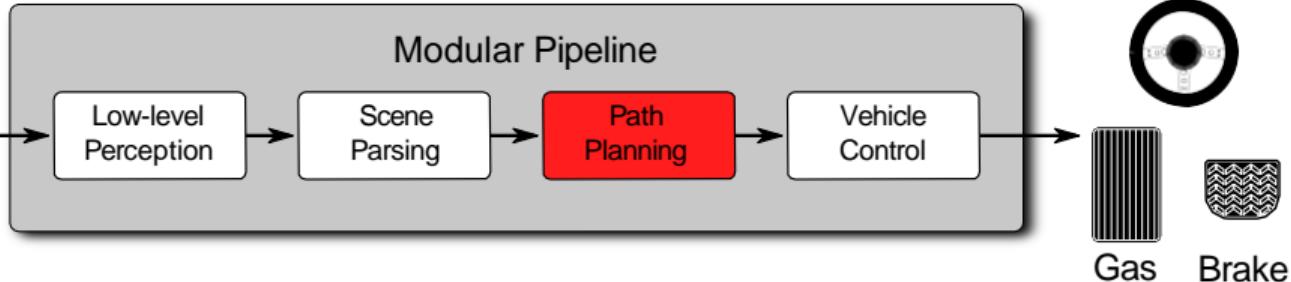
# Modular Pipeline

Sensory Input

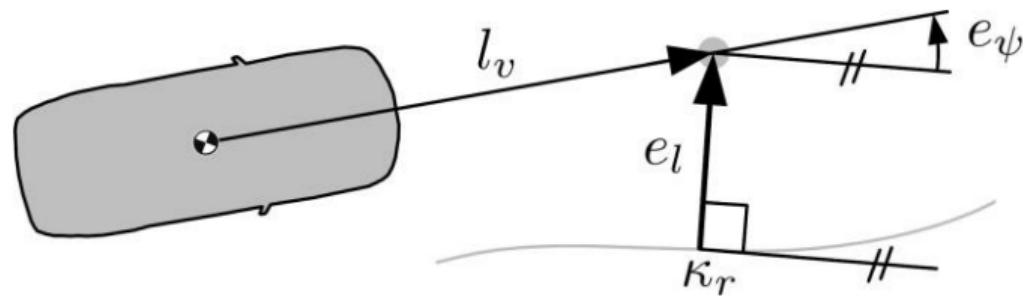
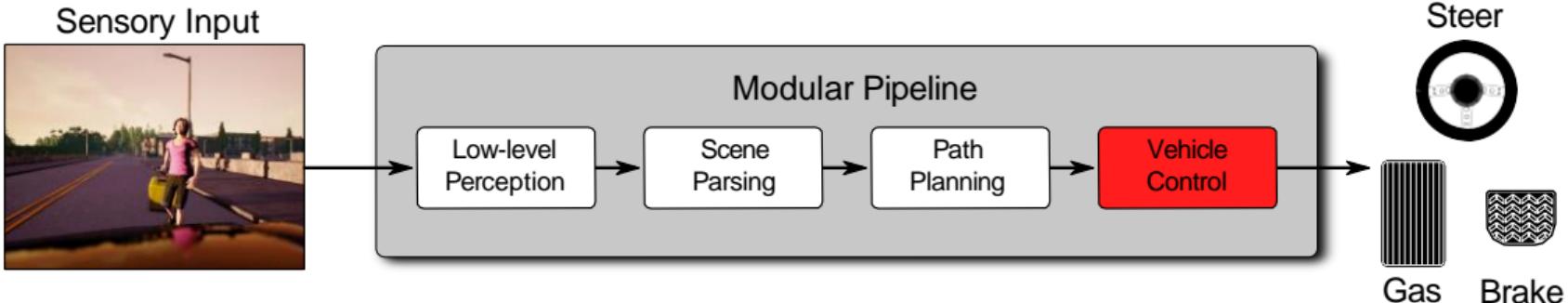


# Modular Pipeline

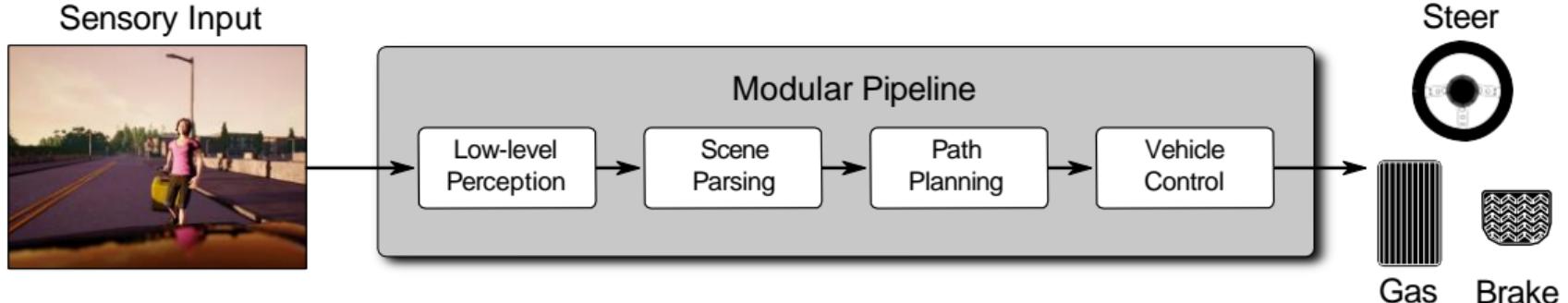
Sensory Input



# Modular Pipeline



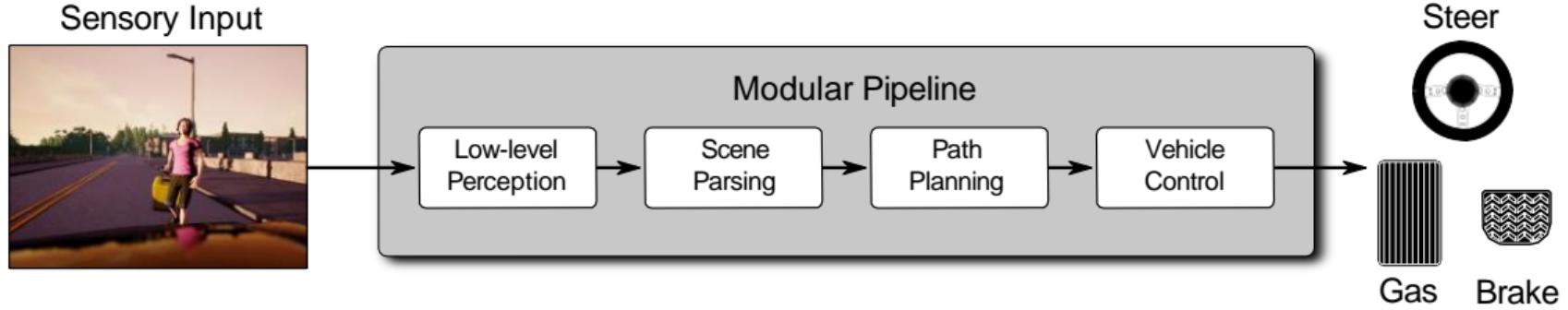
# Modular Pipeline



**Pros:**

**Cons:**

# Modular Pipeline

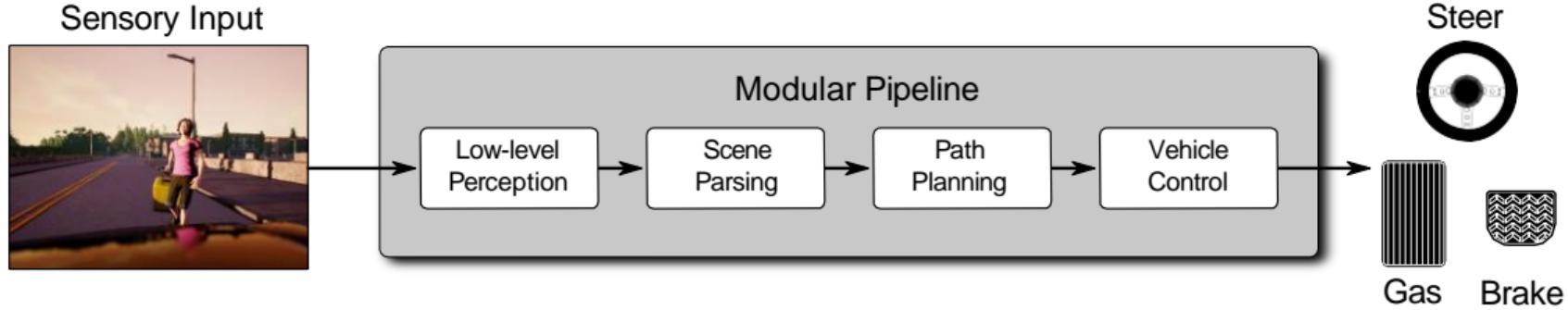


## Pros:

- Small components, easy to develop in parallel

## Cons:

# Modular Pipeline

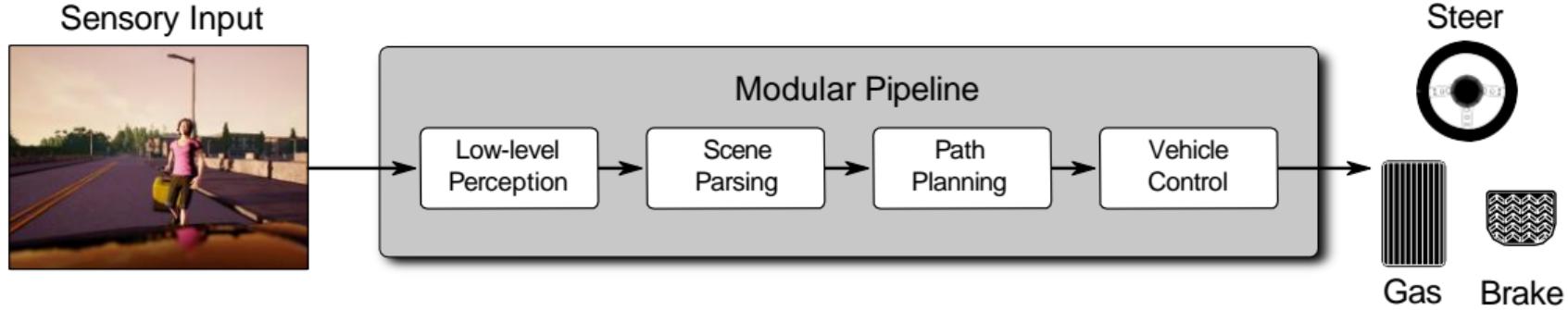


## Pros:

- Small components, easy to develop in parallel
- Interpretability

## Cons:

# Modular Pipeline

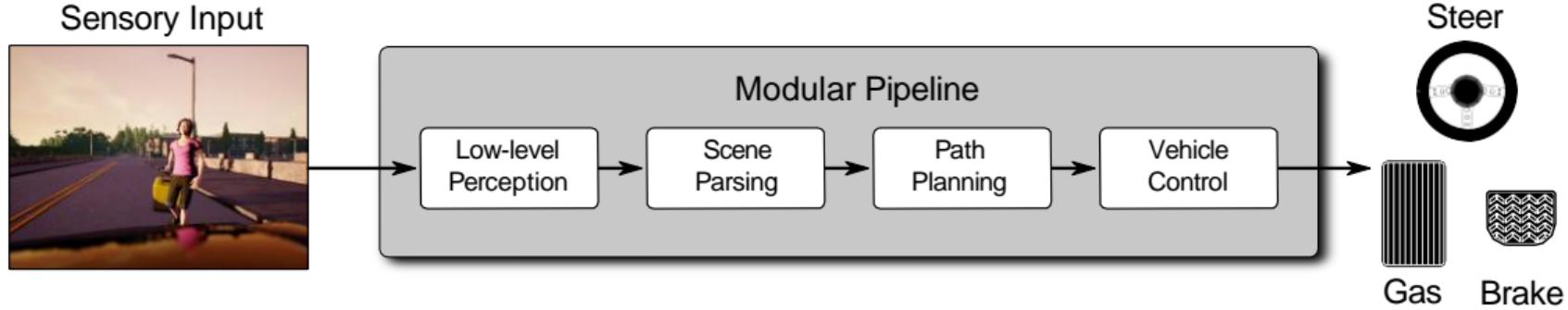


## Pros:

- Small components, easy to develop in parallel
- Interpretability
- Engineered bias/prior, inductive structure

## Cons:

# Modular Pipeline

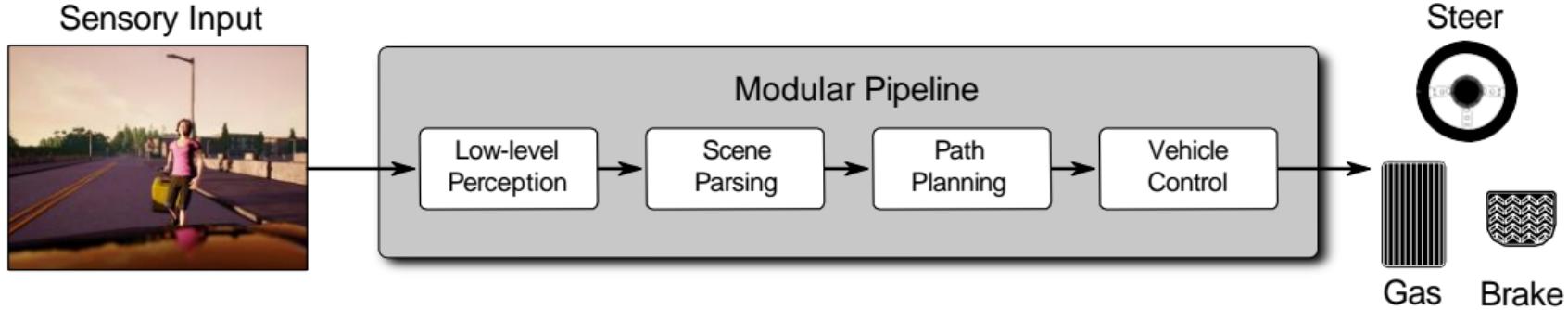


## Pros:

- Small components, easy to develop in parallel
- Interpretability
- Engineered bias/prior, inductive structure

## Cons:

# Modular Pipeline



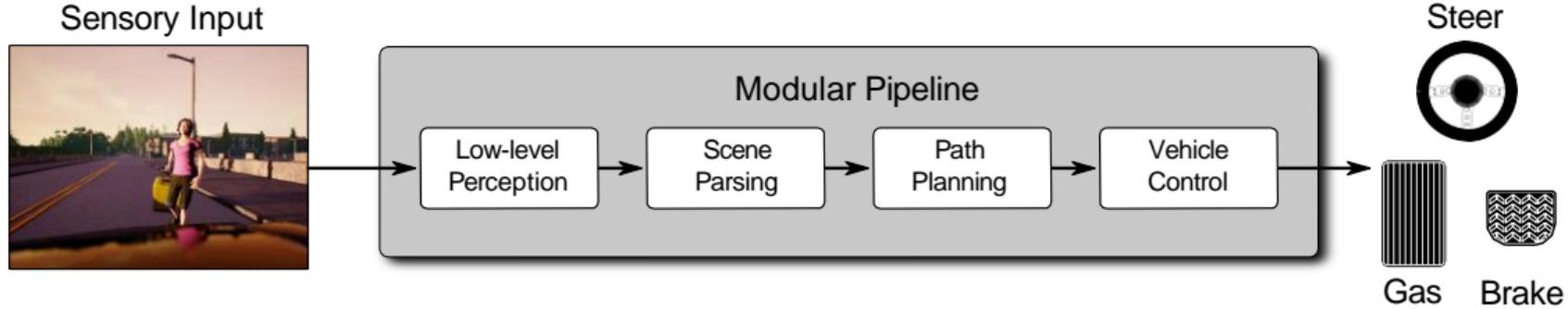
## Pros:

- Small components, easy to develop in parallel
- Interpretability
- Engineered bias/prior, inductive structure

## Cons:

- Piece-wise training (not jointly)

# Modular Pipeline



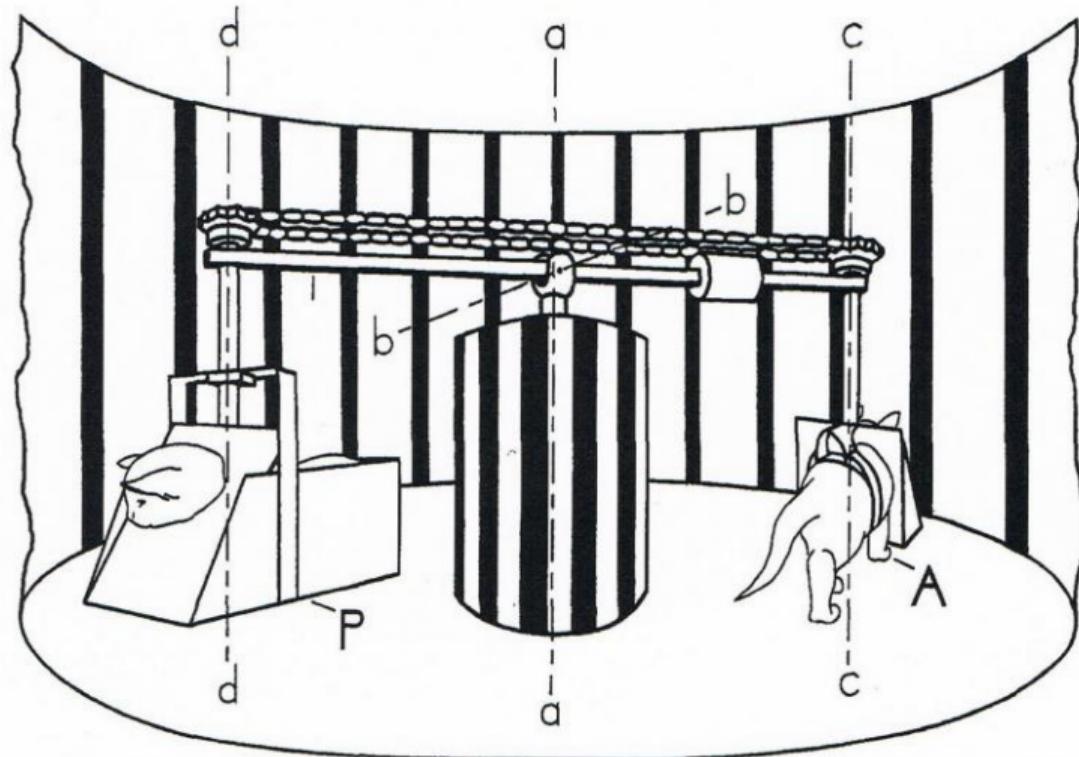
## Pros:

- Small components, easy to develop in parallel
- Interpretability
- Engineered bias/prior, inductive structure

## Cons:

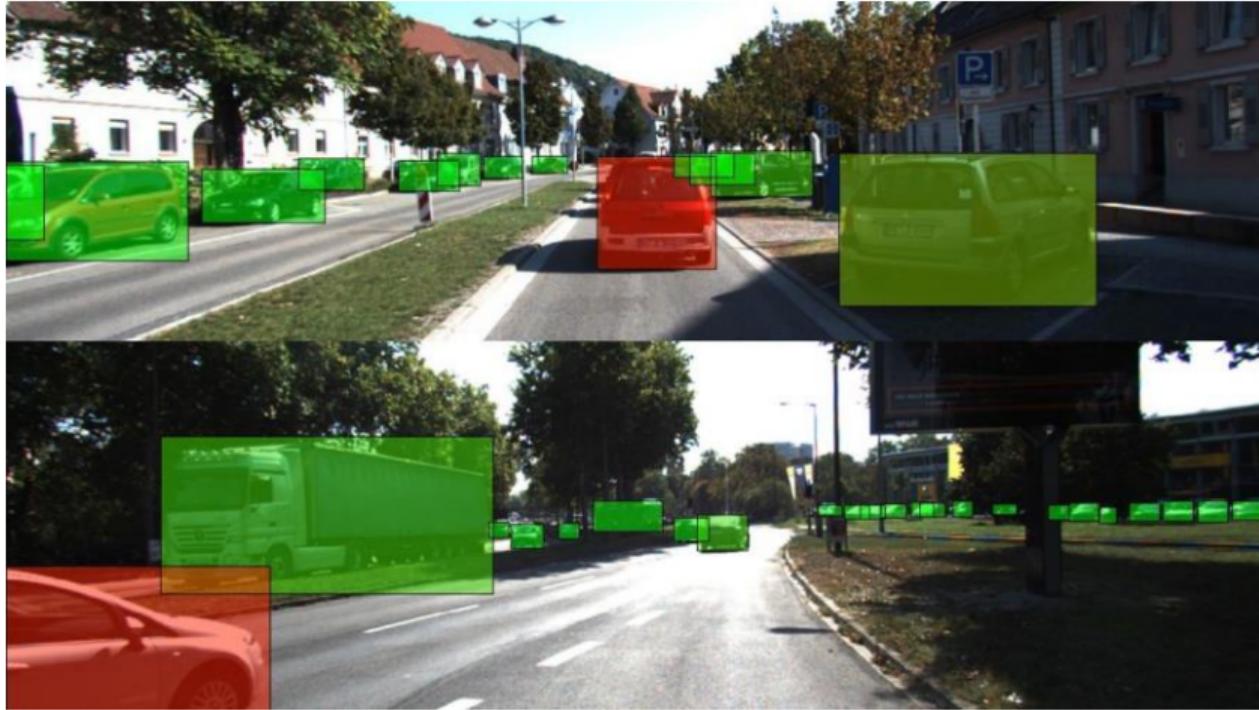
- Piece-wise training (not jointly)
- Requires tedious and costly annotations

# Experiment by Held and Hein: Coupling Perception and Action



Held and Hein: Movement-produced stimulation in the development of visually guided behavior. Journal of Comparative and Physiological Psychology, 1963.

# Modular Pipeline



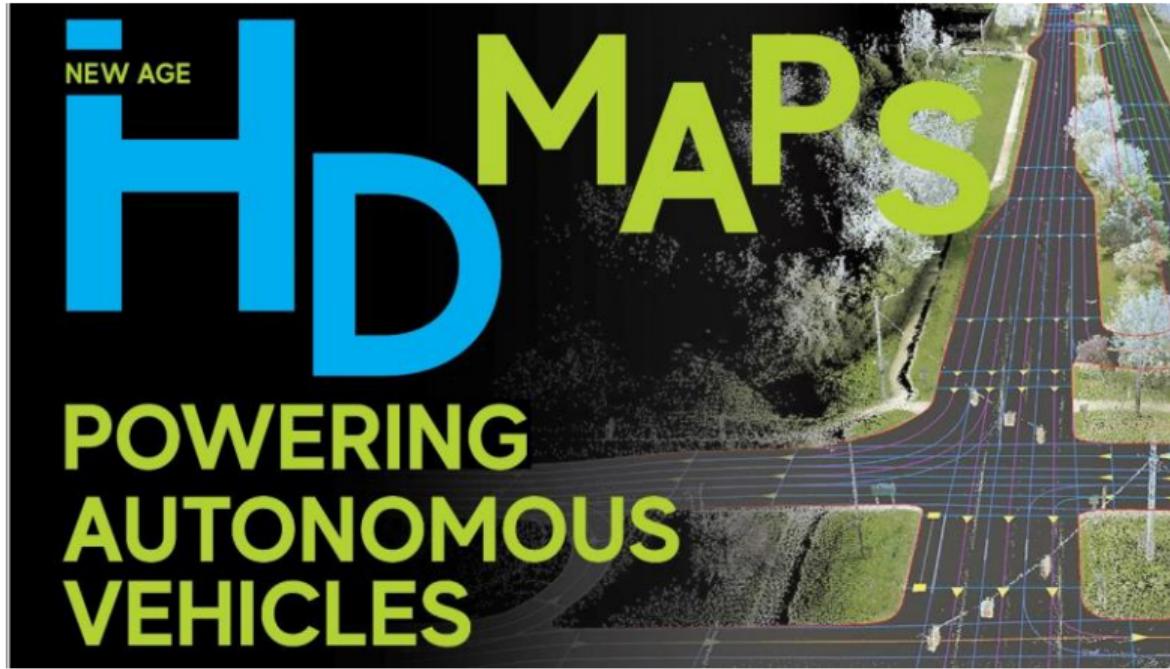
- Piece-wise training difficult: not all objects are equally important!

# Modular Pipeline



Cityscapes Dataset

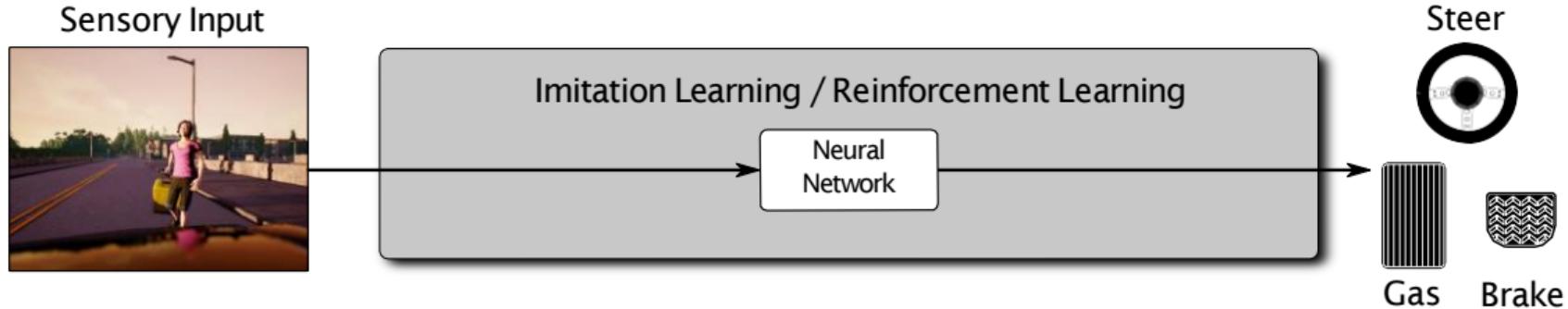
## Modular Pipeline



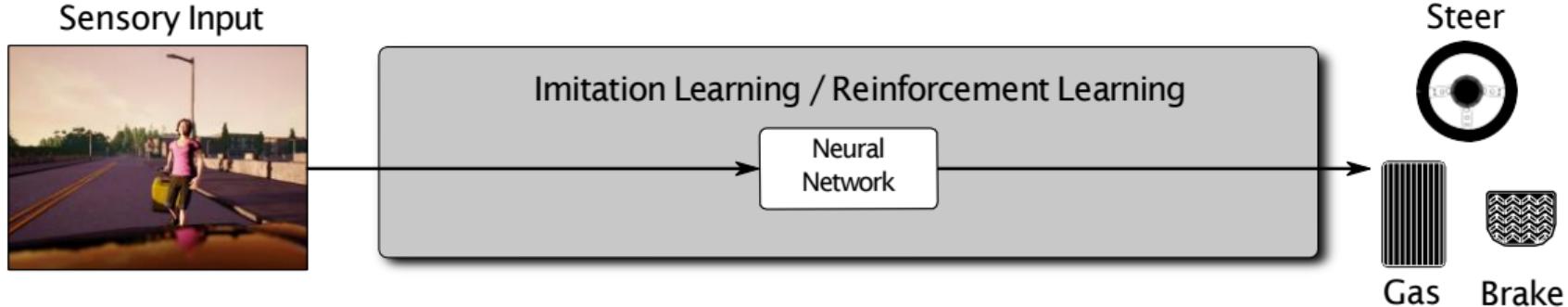
<https://www.geospatialworld.net/article/hd-maps-autonomous-vehicles/>

- HD Maps are expensive to create (data collection & annotation effort)
- HD maps: Centimeter precision lanes, markings, traffic lights/signs, human annotated

# Deep End-to-End Learning



# Deep End-to-End Learning for Navigation



## Examples:

[Pomerleau, NeurIPS 1989]

[Bojarski, Arxiv 2016]

[Codevilla et al., ICRA 2018]



# **Neural Network-Based Autonomous Driving**

**23 November 1992**

# Imitation Learning



# End-to-end Reinforcement learning



Speed x3

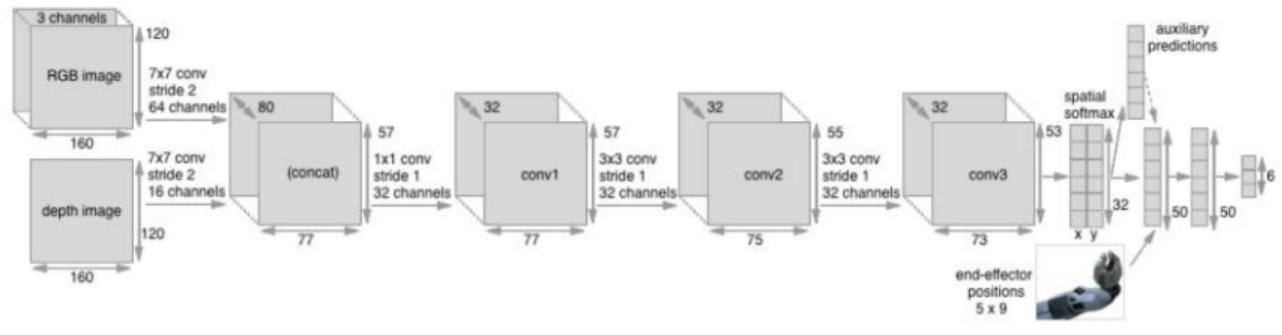
Muelling et al., Learning to select and generalize striking movements in robot table tennis, 2013

# **Robot Motor Skill Coordination with EM-based Reinforcement Learning**

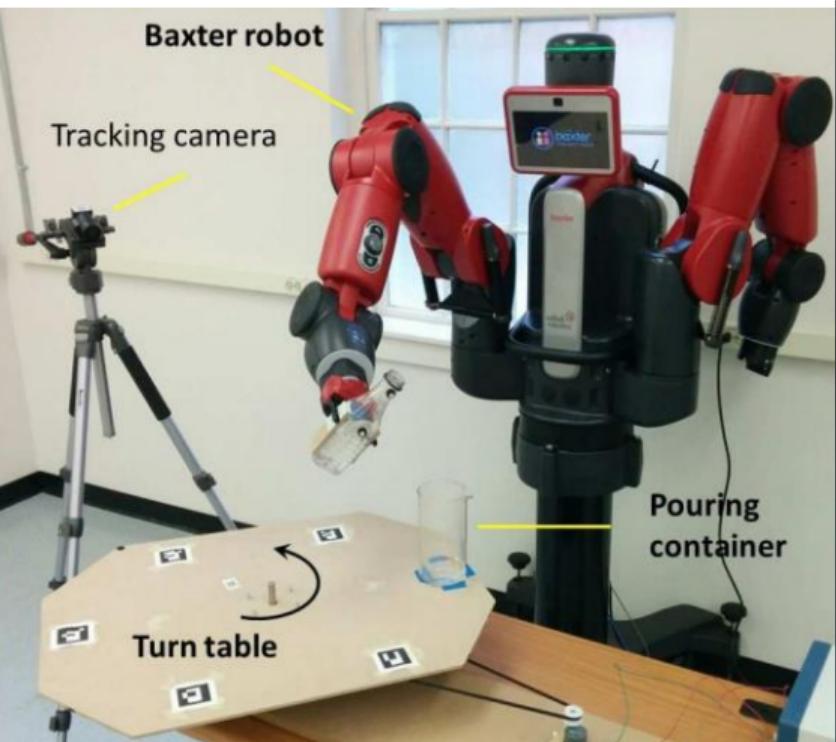
**Petar Kormushev, Sylvain Calinon,  
and Darwin G. Caldwell**

**Italian Institute of Technology**

# Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation, ICRA 2018



# Imitation Learning: Manipulation



## Imitation Learning of Motion Parameters for Dynamic Manipulation Tasks

Joshua Langsfeld, Krishnanand N. Kaipa, Rodolphe J. Gentili and Satyandra K. Gupta

Maryland Robotics Center  
University of Maryland, College Park, MD, USA

# ASSISTER: Assistive Navigation via Conditional Instruction Generation, ECCV 2022

Time: 0:00:03

Speed: 0.90 m/s.

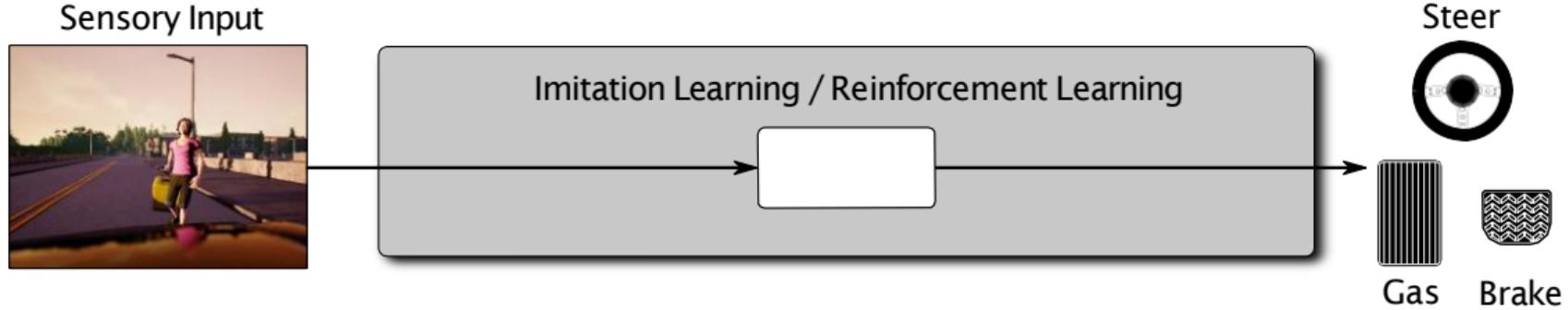
Num of collisions: 0

Turn to your one, then i want you to go forward 3 feet

Pole on your right



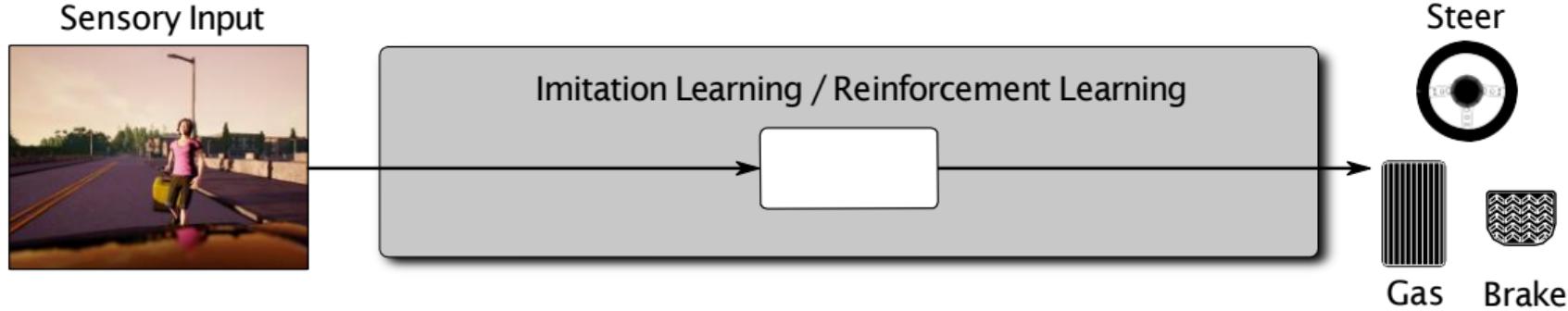
# Deep End-to-End Learning for Navigation



**Pros:**

**Cons:**

# Deep End-to-End Learning for Navigation

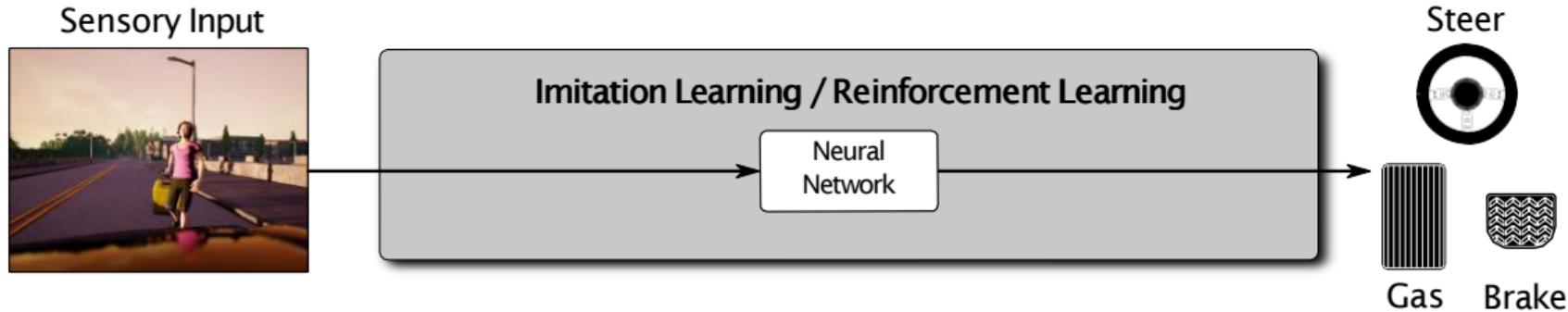


## Pros:

- Task-driven training/optimization
- Cheap annotations

## Cons:

# Deep End-to-End Learning for Navigation



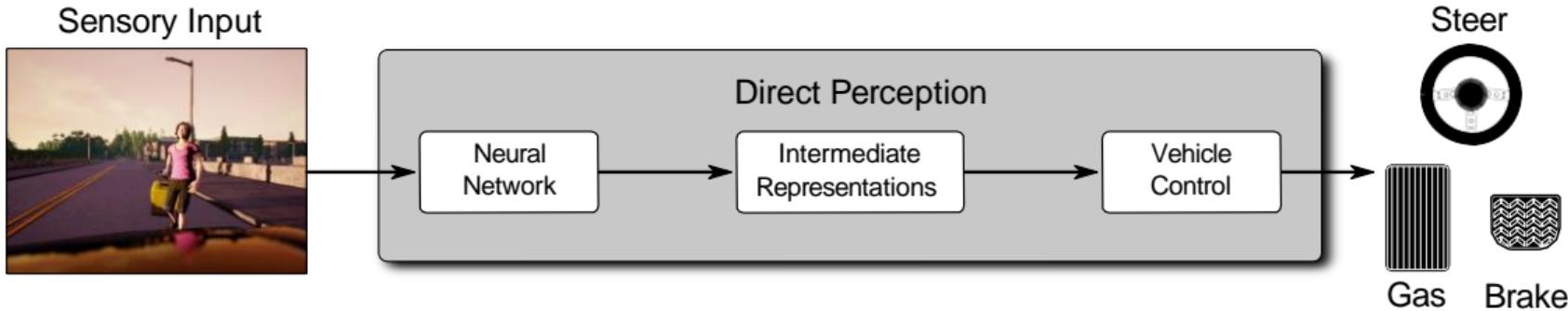
## Pros:

- Task-driven training/optimization
- Cheap annotations

## Cons:

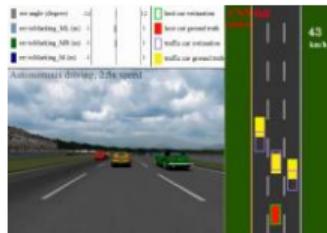
- Overfitting / Generalization
- Interpretability?

# Direct Perception for Navigation



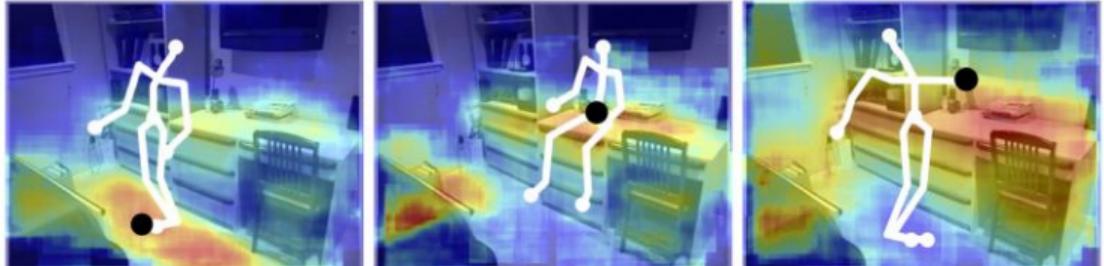
## Examples:

- [Chen et al., ICCV 2015]
  - [Sauer et al., CoRL 2018]
  - Affordances [J. J. Gibson]





Input



Affordance Labels

Figure 1: When humans look at a scene, we can instantly recognize the scene’s *affordances*, or what we can do in it: we can stand on the floor and the desk if necessary; we can sit on the desk too, but not on the bookshelf. We illustrate these by placing the appropriate human pose (white) and contact point (black) on top of likely parts of a per-pixel functional map estimated by one of our algorithms. This paper proposes to revisit the notion of recognizing functional properties directly from pixel data. Note the humans are for illustrative purposes only; our approach generates the heatmap.

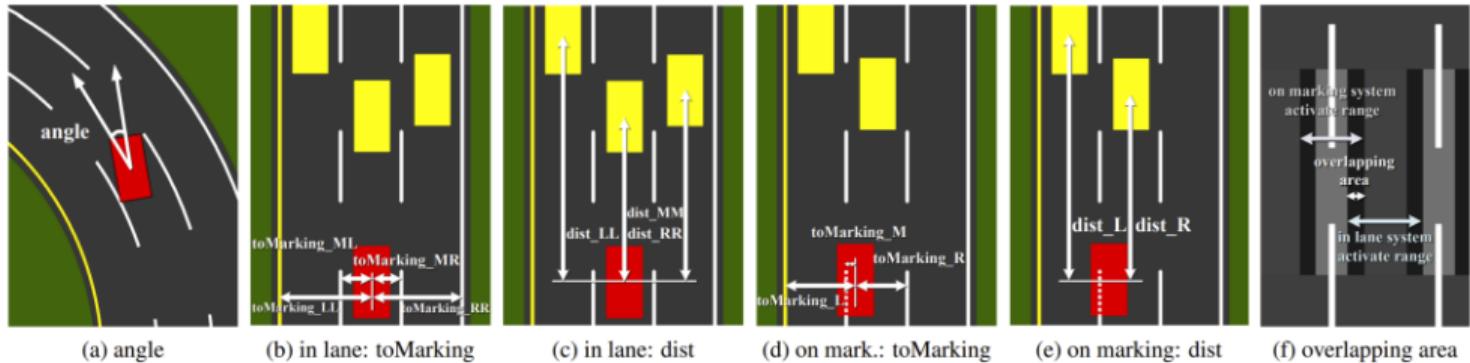


Figure 3: **Illustration of our affordance representation.** A lane changing maneuver needs to traverse the “in lane system” and the “on marking system”. (f) shows the designated overlapping area used to enable smooth transitions.

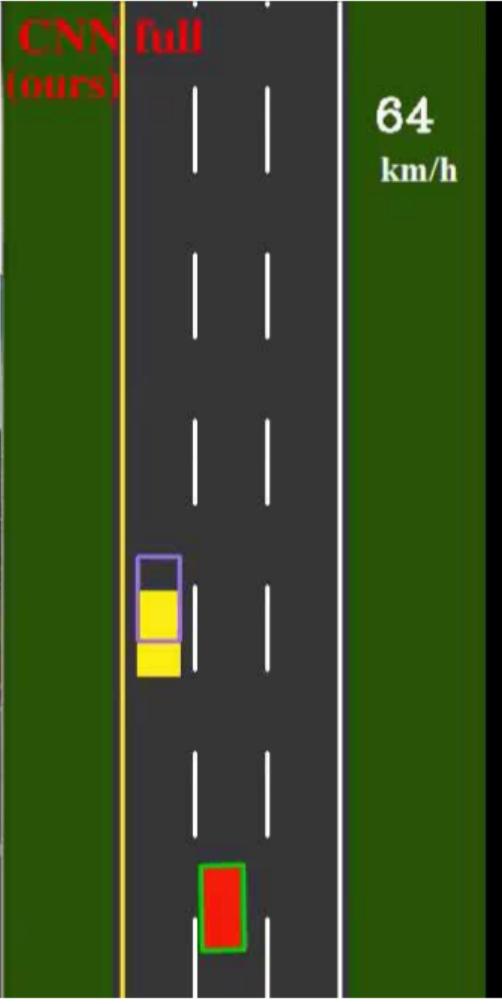
err angle (degree)	-12
err toMarking_ML (m)	-1
err toMarking_MR (m)	-1
err toMarking_M (m)	-1

12	host car estimation
1	host car ground truth
1	traffic car estimation
1	traffic car ground truth

CNN full  
(ours)

64  
km/h

Autonomous driving, 2.5x speed



FPS: 10

Speed: 0 km/h  
Gear: N

Speed Limit: 30 km/h  
Traffic Light: Green

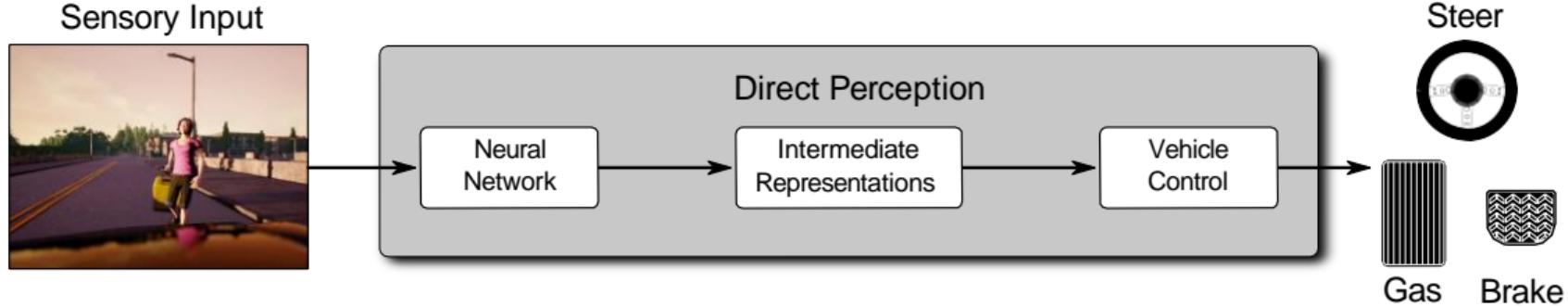
Location: (24, 410, 202, 3, 879)  
Orientation: (1.00, -0.00, 0.00)  
Acceleration: (-0.00, 0.00, 0.00)

Collision (Cars): 0  
Collision (Pedestrian): 0  
Collision (Other): 0

Intersection (Lane): 0%  
Intersection (OffRoad): 0%



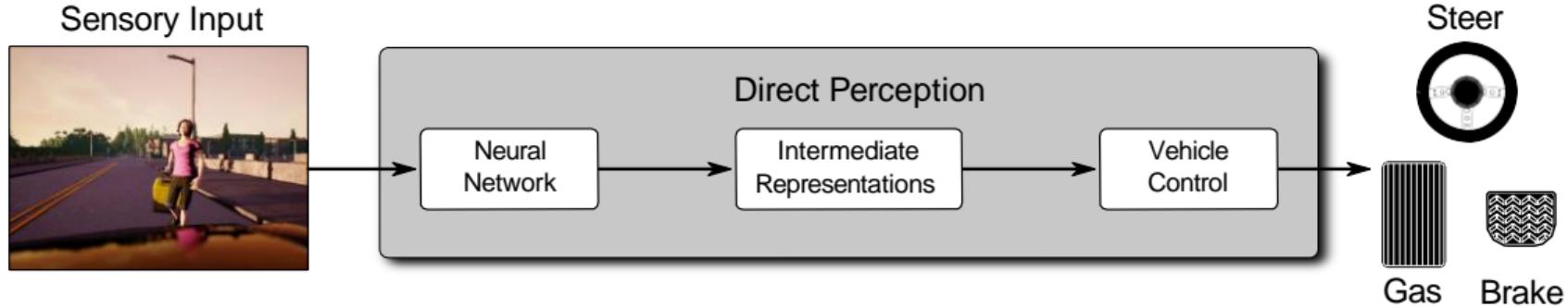
# Direct Perception for Navigation



**Pros:**

**Cons:**

# Direct Perception for Navigation

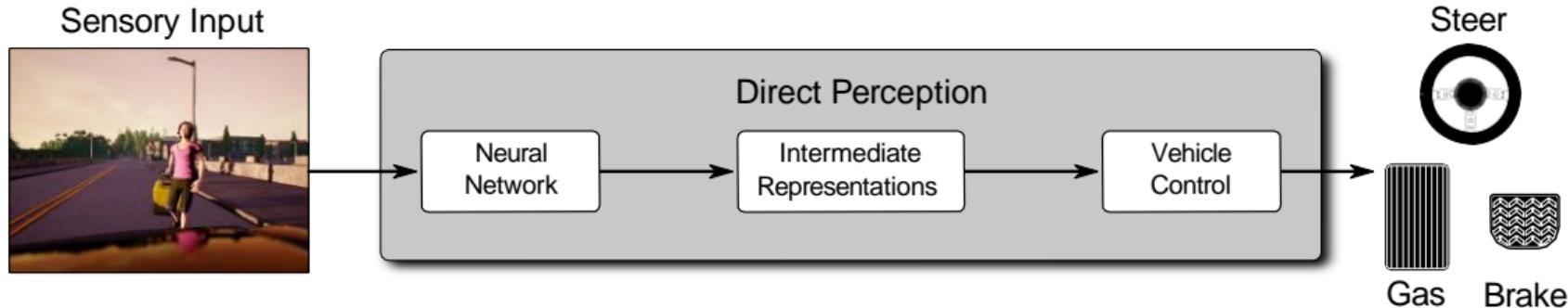


## Pros:

- Compact Representation
- Interpretability

## Cons:

# Direct Perception for Navigation



## Pros:

- Compact Representation
- Interpretability

## Cons:

- What if representation recognition fails?
- **How to choose representations?**

## Further Readings

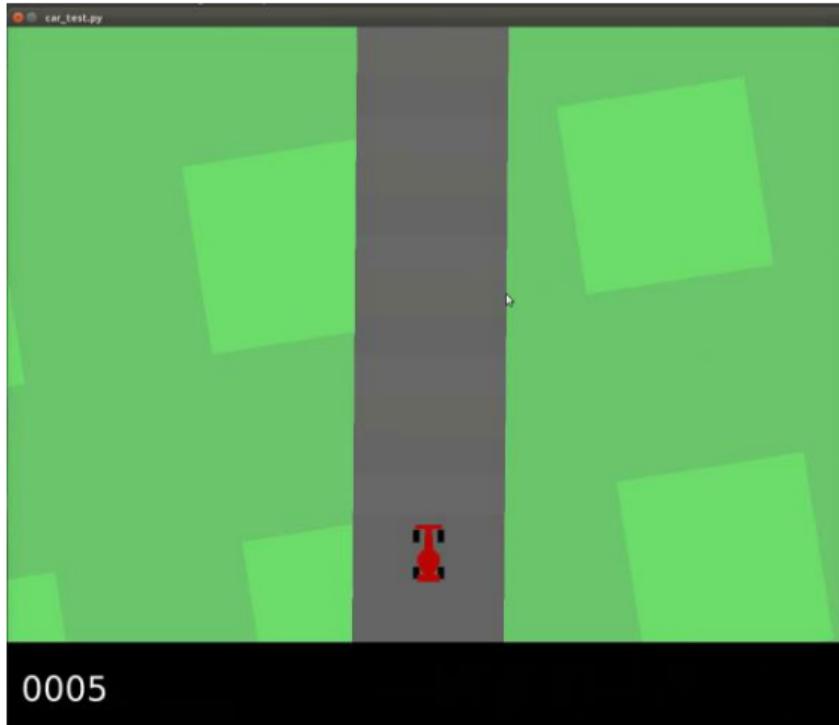
- ▶ Jochem, Pomerleau, Kumar and Armstrong: PANS: A Portable Navigation Platform. IV, 1995.
- ▶ Pomerleau: ALVINN: An Autonomous Land Vehicle in a Neural Network. NIPS, 1988.
- ▶ Chen, Jochem and Pomerleau: AURORA: A Vision-Based Roadway Departure Warning System. IROS, 1995.
- ▶ Dickmanns and Mysliwetz: Recursive 3-D Road and Relative Ego-State Recognition. PAMI, 1992.
- ▶ Ernst Dickmanns, "The development of machine vision for road vehicles in the last decade." IEEE Intelligent Vehicles Symposium, 2002
- ▶ Geiger, Lenz and Urtasun: Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. CVPR, 2012.
- ▶ Dosovitskiy, Ros, Codevilla, Lopez, Kolton CARLA: An Open Urban Driving Simulator, CoRL, 2017

# Imitation Learning

## Imitation in Babies



# Imitation Learning: Car Racing

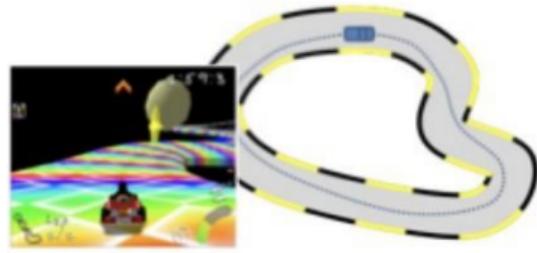


Trainer  
(Human Driver)



Trainee  
(Neural Network)

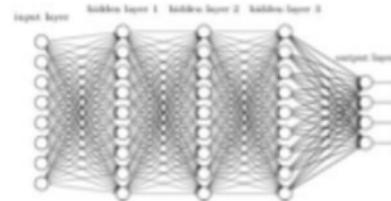
# Ingredients of Imitation Learning



Demonstrations or Demonstrator



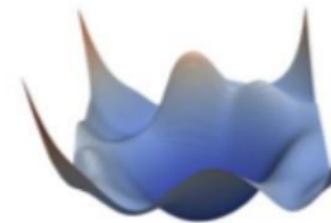
Environment / Simulator



Policy Class



Loss Function



Learning Algorithm

# Formal Definition of the Imitation Learning Problem

**Components:**

# Formal Definition of the Imitation Learning Problem

## Components:

- State:  $s \in \mathcal{S}$  may be partially observed (e.g., game screen)

# Formal Definition of the Imitation Learning Problem

## Components:

- ▶ State:  $s \in \mathcal{S}$  may be partially observed (e.g., game screen)
- ▶ Action:  $a \in \mathcal{A}$  may be discrete or continuous (e.g., turn angle, speed)

# Formal Definition of the Imitation Learning Problem

## Components:

- ▶ State:  $s \in \mathcal{S}$  may be partially observed (e.g., game screen)
- ▶ Action:  $a \in \mathcal{A}$  may be discrete or continuous (e.g., turn angle, speed)
- ▶ Policy:  $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$  we want to learn the policy parameters  $\theta$

# Formal Definition of the Imitation Learning Problem

## Components:

- ▶ State:  $s \in \mathcal{S}$  may be partially observed (e.g., game screen)
- ▶ Action:  $a \in \mathcal{A}$  may be discrete or continuous (e.g., turn angle, speed)
- ▶ Policy:  $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$  we want to learn the policy parameters  $\theta$
- ▶ Optimal action:  $a^* \in \mathcal{A}$  provided by expert demonstrator

# Formal Definition of the Imitation Learning Problem

## Components:

- ▶ State:  $s \in \mathcal{S}$  may be partially observed (e.g., game screen)
- ▶ Action:  $a \in \mathcal{A}$  may be discrete or continuous (e.g., turn angle, speed)
- ▶ Policy:  $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$  we want to learn the policy parameters  $\theta$
- ▶ Optimal action:  $a^* \in \mathcal{A}$  provided by expert demonstrator
- ▶ Optimal policy:  $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$  provided by expert demonstrator

# Formal Definition of the Imitation Learning Problem

## Components:

- ▶ State:  $s \in \mathcal{S}$  may be partially observed (e.g., game screen)
- ▶ Action:  $a \in \mathcal{A}$  may be discrete or continuous (e.g., turn angle, speed)
- ▶ Policy:  $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$  we want to learn the policy parameters  $\theta$
- ▶ Optimal action:  $a^* \in \mathcal{A}$  provided by expert demonstrator
- ▶ Optimal policy:  $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$  provided by expert demonstrator
- ▶ State dynamics:  $P(s_{i+1}|s_i, a_i)$  simulator, typically not known to policy  
Often deterministic:  $s_{i+1} = T(s_i, a_i)$  deterministic mapping

# Formal Definition of the Imitation Learning Problem

## Components:

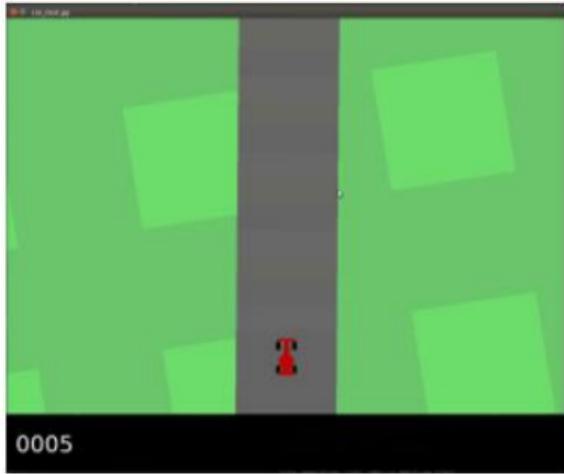
- ▶ State:  $s \in \mathcal{S}$  may be partially observed (e.g., game screen)
- ▶ Action:  $a \in \mathcal{A}$  may be discrete or continuous (e.g., turn angle, speed)
- ▶ Policy:  $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$  we want to learn the policy parameters  $\theta$
- ▶ Optimal action:  $a^* \in \mathcal{A}$  provided by expert demonstrator
- ▶ Optimal policy:  $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$  provided by expert demonstrator
- ▶ State dynamics:  $P(s_{i+1}|s_i, a_i)$  simulator, typically not known to policy  
Often deterministic:  $s_{i+1} = T(s_i, a_i)$  deterministic mapping
- ▶ Rollout: Given  $s_0$ , sequentially execute  $a_i = \pi_\theta(s_i)$  & sample  $s_{i+1} \sim P(s_{i+1}|s_i, a_i)$   
yields trajectory  $\tau = (s_0, a_0, s_1, a_1, \dots)$

# Formal Definition of the Imitation Learning Problem

## Components:

- ▶ State:  $s \in \mathcal{S}$  may be partially observed (e.g., game screen)
- ▶ Action:  $a \in \mathcal{A}$  may be discrete or continuous (e.g., turn angle, speed)
- ▶ Policy:  $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$  we want to learn the policy parameters  $\theta$
- ▶ Optimal action:  $a^* \in \mathcal{A}$  provided by expert demonstrator
- ▶ Optimal policy:  $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$  provided by expert demonstrator
- ▶ State dynamics:  $P(s_{i+1}|s_i, a_i)$  simulator, typically not known to policy  
Often deterministic:  $s_{i+1} = T(s_i, a_i)$  deterministic mapping
- ▶ Rollout: Given  $s_0$ , sequentially execute  $a_i = \pi_\theta(s_i)$  & sample  $s_{i+1} \sim P(s_{i+1}|s_i, a_i)$   
yields trajectory  $\tau = (s_0, a_0, s_1, a_1, \dots)$
- ▶ Loss function:  $\mathcal{L}(a^*, a)$  loss of action  $a$  given optimal action  $a^*$

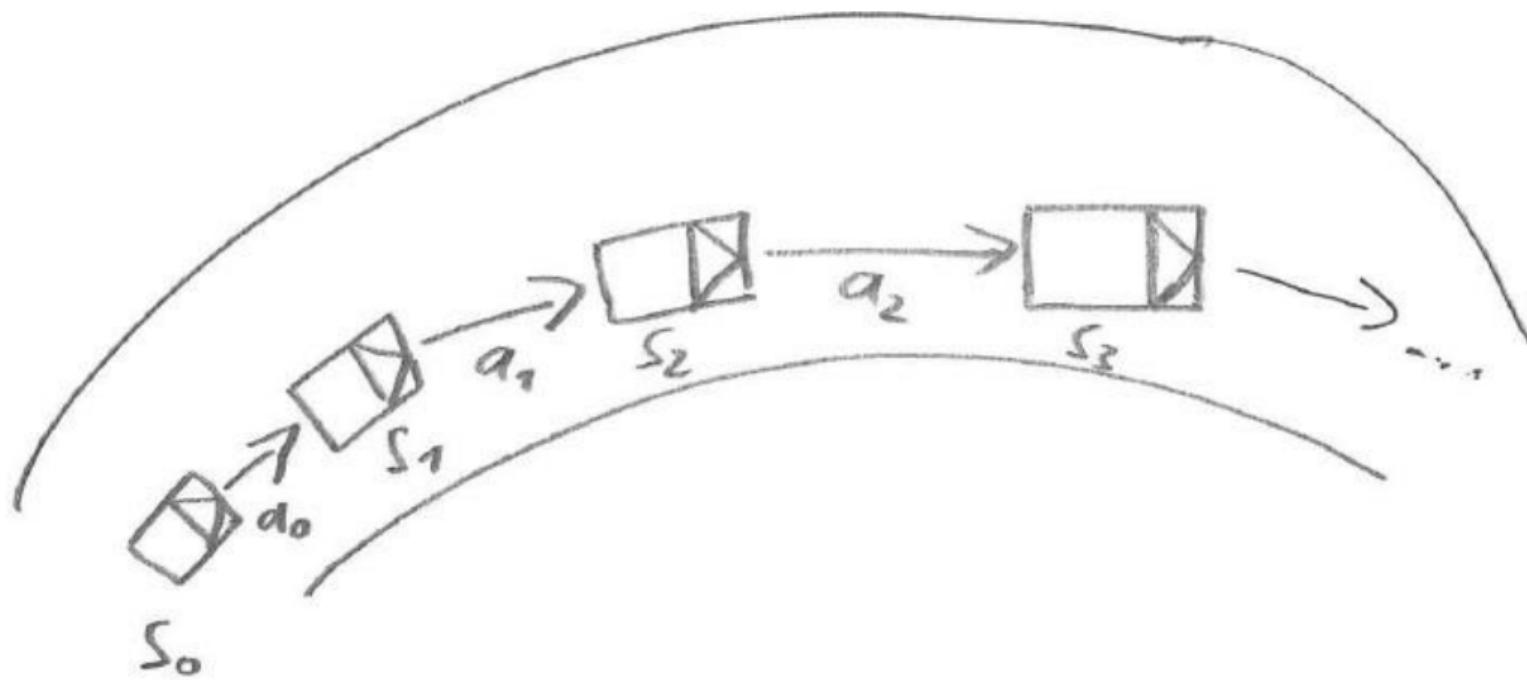
# Deterministic vs. Stochastic



$$s_{i+1} = T(s_i, a_i)$$

$$s_{i+1} \sim P(s_{i+1} | s_i, a_i)$$

## Rollout - Example

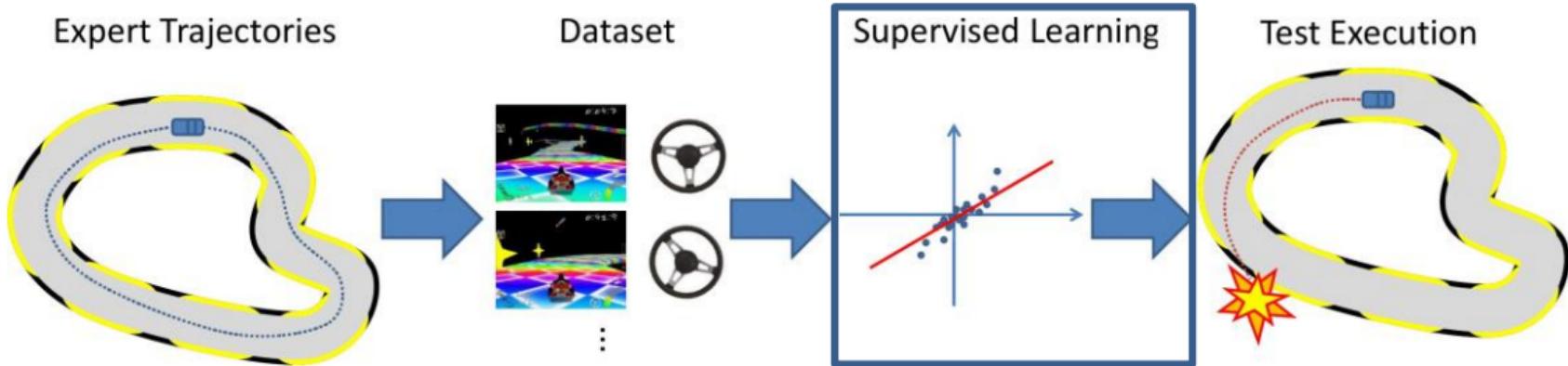


# Formal Definition of the Imitation Learning Problem

## Components:

- ▶ State:  $s \in \mathcal{S}$  may be partially observed (e.g., game screen)
- ▶ Action:  $a \in \mathcal{A}$  may be discrete or continuous (e.g., turn angle, speed)
- ▶ Policy:  $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$  we want to learn the policy parameters  $\theta$
- ▶ Optimal action:  $a^* \in \mathcal{A}$  provided by expert demonstrator
- ▶ Optimal policy:  $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$  provided by expert demonstrator
- ▶ State dynamics:  $P(s_{i+1}|s_i, a_i)$  simulator, typically not known to policy  
Often deterministic:  $s_{i+1} = T(s_i, a_i)$  deterministic mapping
- ▶ Rollout: Given  $s_0$ , sequentially execute  $a_i = \pi_\theta(s_i)$  & sample  $s_{i+1} \sim P(s_{i+1}|s_i, a_i)$   
yields trajectory  $\tau = (s_0, a_0, s_1, a_1, \dots)$
- ▶ Loss function:  $\mathcal{L}(a^*, a)$  loss of action  $a$  given optimal action  $a^*$

# Behavior Cloning in a Nutshell



Hard coding policies is often difficult  $\Rightarrow$  Rather use a data-driven approach!

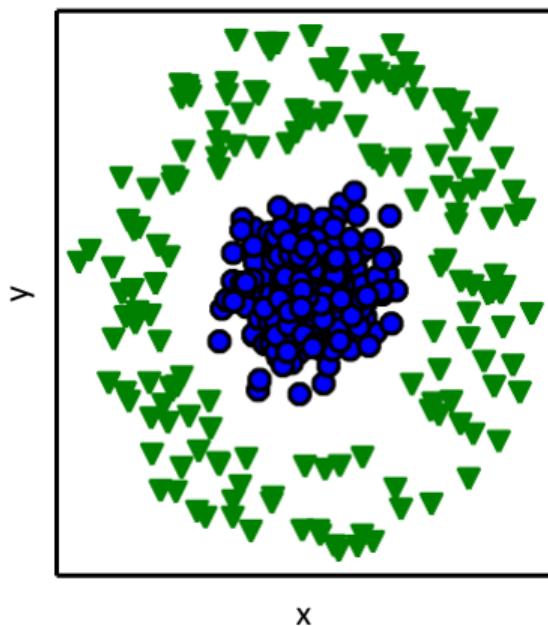
- **Given:** demonstrations or demonstrator
- **Goal:** train a policy to mimic decision
- **Variants:** behavior cloning (this lecture), inverse optimal control, ...
- Learning is safe
- Applications in driving, flying, manipulation, teaching robots, teleoperation...

A close-up, high-resolution 3D rendering of a neural network. The central focus is a dark, multi-lobed neuron cell body with numerous long, thin, dark brown processes extending outwards. These processes, known as axons, terminate in small, glowing yellowish-orange spheres representing synaptic terminals. The background is filled with similar neurons, creating a complex, web-like texture against a warm, golden-yellow gradient.

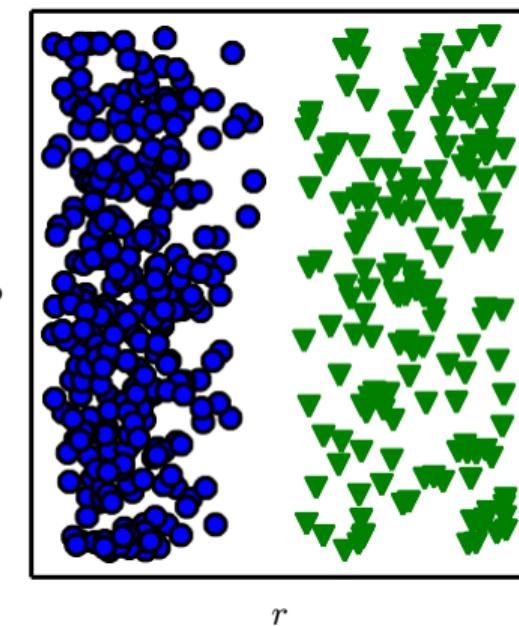
# Deep Learning

# Motivation: Representation Matters

Cartesian Coordinates



Polar Coordinates



How to separate the blue points from the green points with a linear classifier?

## Example: Multi-Class Classification

Input

Model

Output



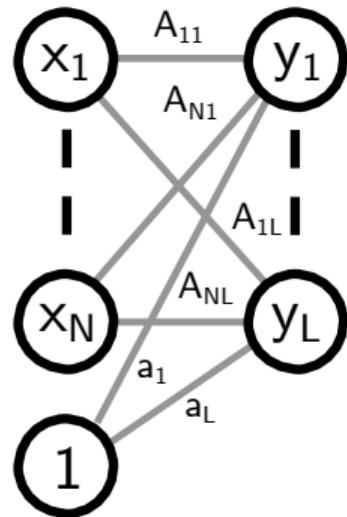
"Beach"

$$f: \mathbf{x} \mapsto y \in \{1, \dots, L\}$$

$$f: \mathbf{x} \mapsto y = [0, \dots, 1, \dots, 0] \in \{0, 1\}^L$$

# Linear Regression

- Mapping:  
 $\mathbf{x}$  = Image  
 $\mathbf{y} = \mathbf{Ax} + \mathbf{a}$



# Linear Regression

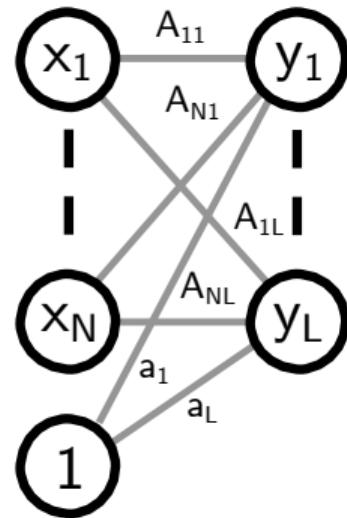
- ▶ Mapping:

$\mathbf{x}$  = Image

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{a}$$

- ▶ Classification:

$$L^* = \operatorname{argmax}_l y_l$$



# Linear Regression

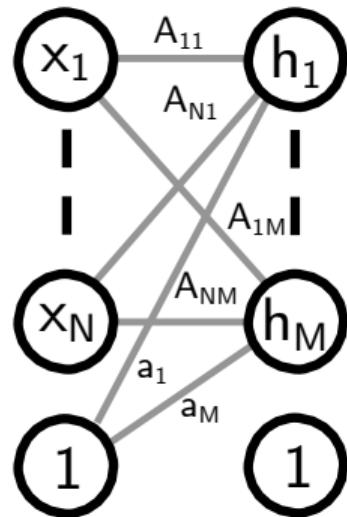


# Linear Regression

3 Layers:

$\mathbf{x}$  = Image

$$\mathbf{h} = \mathbf{Ax} + \mathbf{a}$$



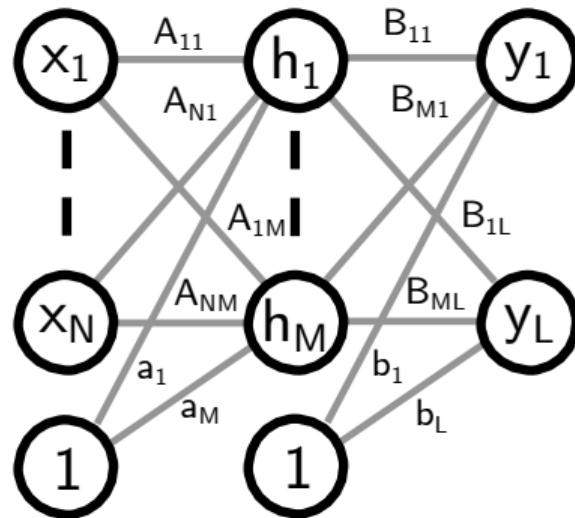
# Linear Regression

3 Layers:

$\mathbf{x}$  = Image

$\mathbf{h} = \mathbf{Ax} + \mathbf{a}$

$\mathbf{y} = \mathbf{Bh} + \mathbf{b}$



# Linear Regression

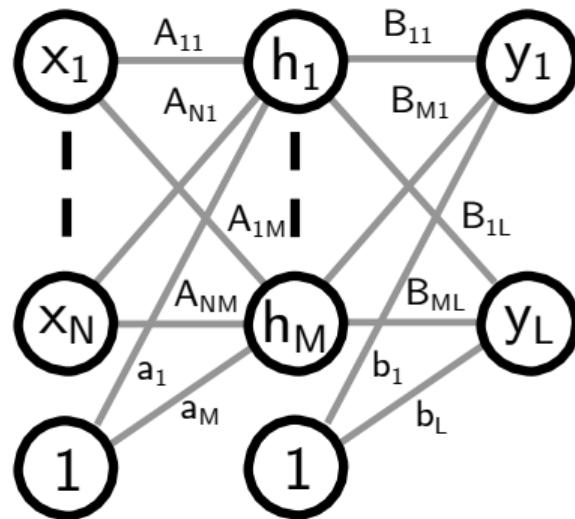
3 Layers:

$\mathbf{x}$  = Image

$$\mathbf{h} = \mathbf{Ax} + \mathbf{a}$$

$$\mathbf{y} = \mathbf{Bh} + \mathbf{b}$$

Is this model better?



# Linear Regression

3 Layers:

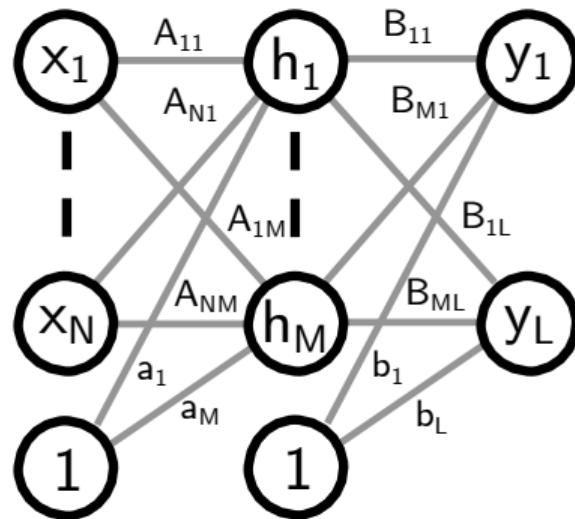
$\mathbf{x}$  = Image

$$\mathbf{h} = \mathbf{Ax} + \mathbf{a}$$

$$\mathbf{y} = \mathbf{Bh} + \mathbf{b}$$

Is this model better?

$$\mathbf{y} = \mathbf{Bh} + \mathbf{b}$$



# Linear Regression

3 Layers:

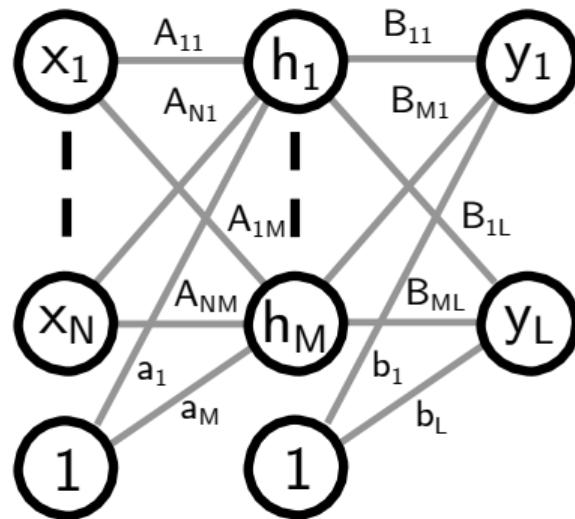
$\mathbf{x}$  = Image

$$\mathbf{h} = \mathbf{Ax} + \mathbf{a}$$

$$\mathbf{y} = \mathbf{Bh} + \mathbf{b}$$

Is this model better?

$$\mathbf{y} = \mathbf{B}(\mathbf{Ax} + \mathbf{a}) + \mathbf{b}$$



# Linear Regression

3 Layers:

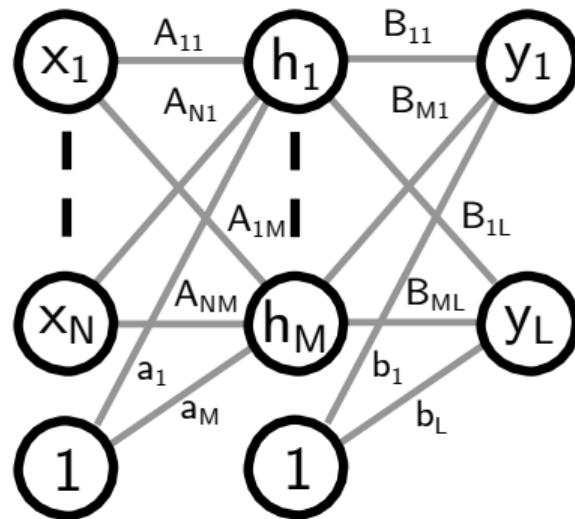
$\mathbf{x}$  = Image

$$\mathbf{h} = \mathbf{Ax} + \mathbf{a}$$

$$\mathbf{y} = \mathbf{Bh} + \mathbf{b}$$

Is this model better?

$$\mathbf{y} = \mathbf{BAx} + \mathbf{Ba} + \mathbf{b}$$



# Linear Regression

3 Layers:

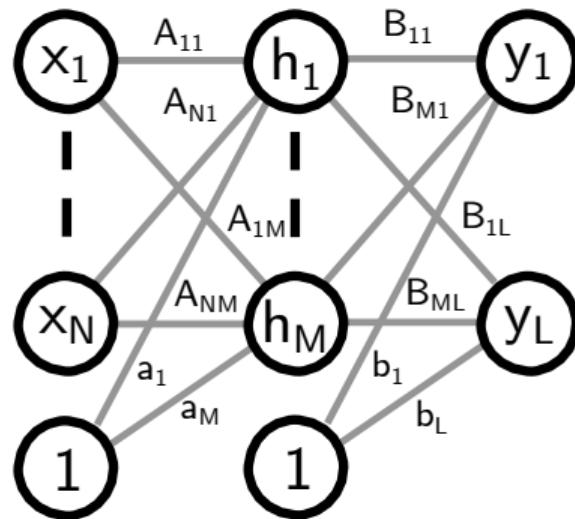
$$\mathbf{x} = \text{Image}$$

$$\mathbf{h} = \mathbf{Ax} + \mathbf{a}$$

$$\mathbf{y} = \mathbf{Bh} + \mathbf{b}$$

Is this model better?

$$\mathbf{y} = \underbrace{\mathbf{BA} \mathbf{x}}_{=C} + \underbrace{\mathbf{Ba} + \mathbf{b}}_{=c}$$



# Linear Regression

3 Layers:

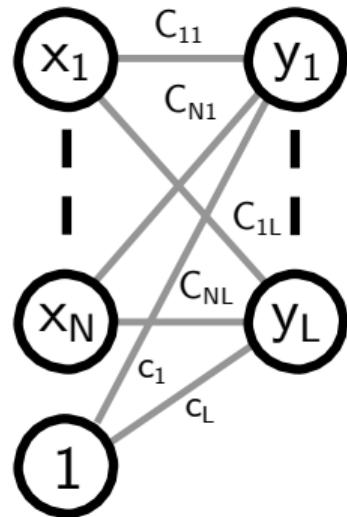
$\mathbf{x}$  = Image

$$\mathbf{h} = \mathbf{Ax} + \mathbf{a}$$

$$\mathbf{y} = \mathbf{Bh} + \mathbf{b}$$

Is this model better?

$$\mathbf{y} = \mathbf{Cx} + \mathbf{c}$$



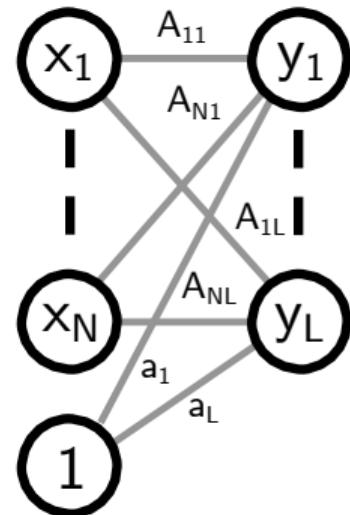
# Logistic Regression

- Mapping:  
 $\mathbf{x}$  = Image



# Logistic Regression

- Mapping:  
 $\mathbf{x}$  = Image  
 $\mathbf{y} = \sigma(\mathbf{Ax} + \mathbf{a})$



# Logistic Regression

- Mapping:

$\mathbf{x}$  = Image

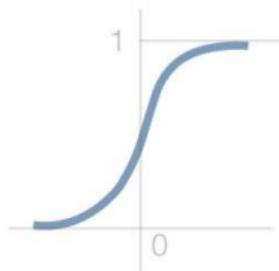
$$\mathbf{y} = \sigma(\mathbf{Ax} + \mathbf{a})$$

- With (elementwise):

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

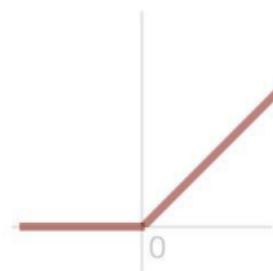
(a) Sigmoid

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



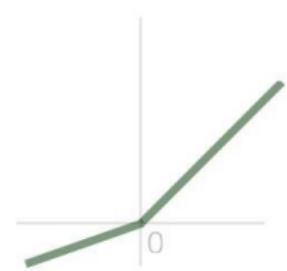
(b) ReLU

$$\sigma(z) = \begin{cases} z, & z > 0 \\ 0, & z \leq 0 \end{cases}$$



(c) PReLU

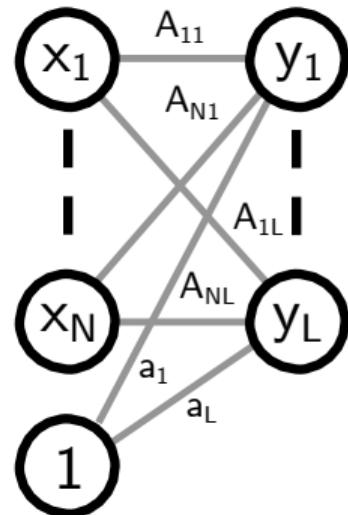
$$\sigma(z) = \begin{cases} z, & z > 0 \\ az, & z \leq 0 \end{cases}$$



# Logistic Regression

- Mapping:  
 $\mathbf{x}$  = Image  
 $\mathbf{y} = \sigma(\mathbf{Ax} + \mathbf{a})$
- With (elementwise):

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$



# Multilayer Perceptron

- 3 Layers:

$\mathbf{x}$  = Image

$$\mathbf{h} = \sigma(\mathbf{Ax} + \mathbf{a})$$

$$\mathbf{y} = \sigma(\mathbf{Bh} + \mathbf{b})$$



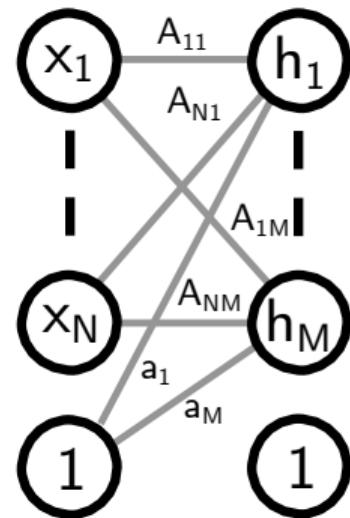
# Multilayer Perceptron

- 3 Layers:

$\mathbf{x}$  = Image

$$\mathbf{h} = \sigma(\mathbf{Ax} + \mathbf{a})$$

$$\mathbf{y} = \sigma(\mathbf{Bh} + \mathbf{b})$$



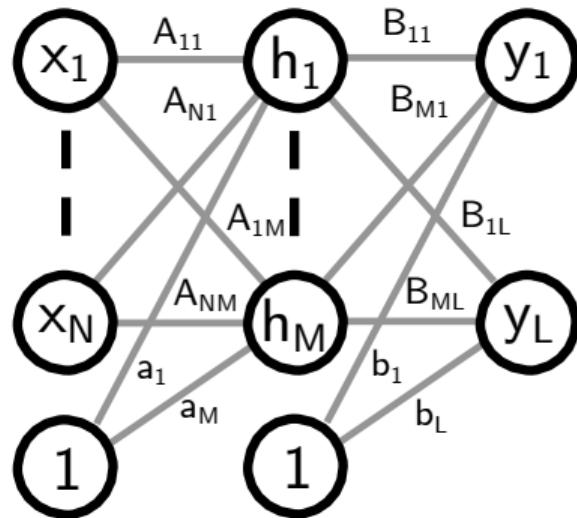
# Multilayer Perceptron

- 3 Layers:

$\mathbf{x}$  = Image

$$\mathbf{h} = \sigma(\mathbf{Ax} + \mathbf{a})$$

$$\mathbf{y} = \sigma(\mathbf{Bh} + \mathbf{b})$$



# Multilayer Perceptron

- 3 Layers:

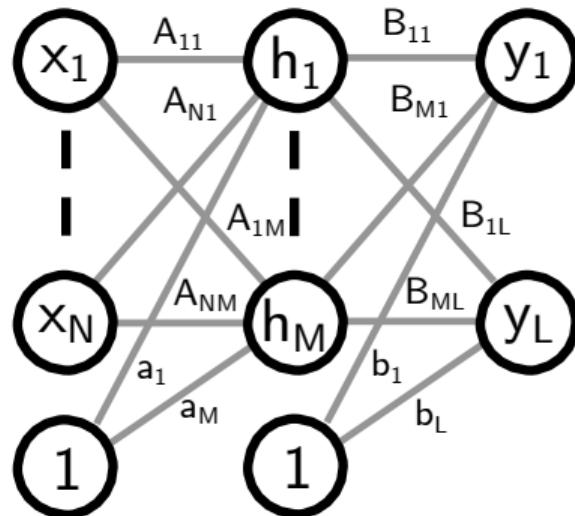
$\mathbf{x}$  = Image

$$\mathbf{h} = \sigma(\mathbf{Ax} + \mathbf{a})$$

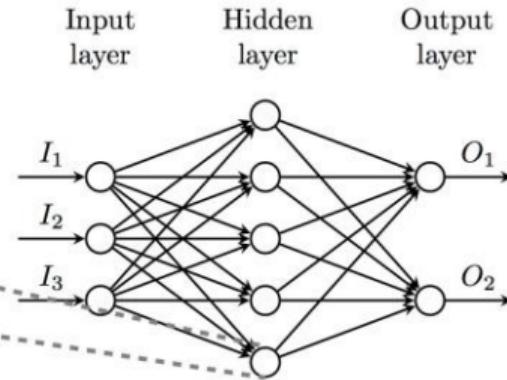
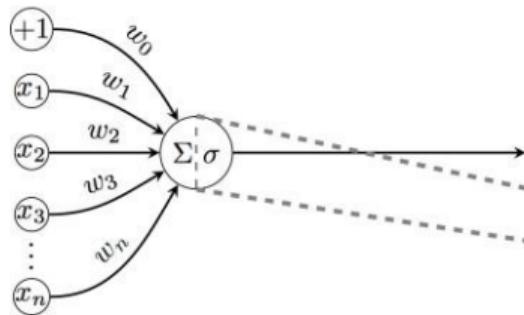
$$\mathbf{y} = \sigma(\mathbf{Bh} + \mathbf{b})$$

- Now:

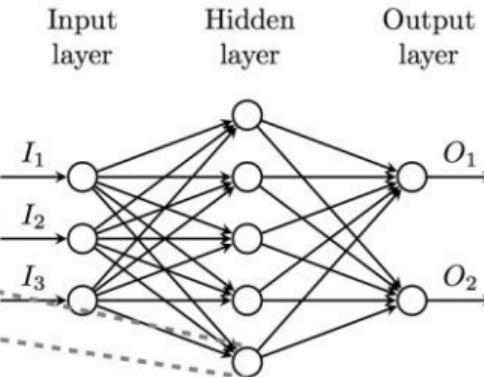
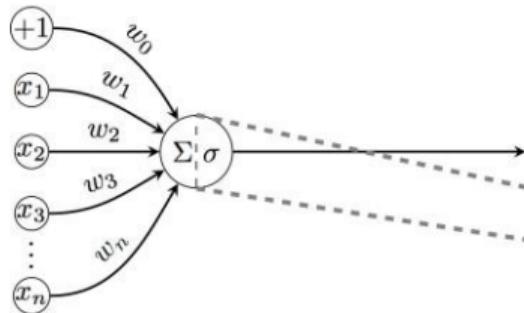
$$\mathbf{y} = \sigma(\mathbf{B}(\sigma(\mathbf{Ax} + \mathbf{a})) + \mathbf{b})$$



# Summary: Multilayer Perceptron in a Nutshell



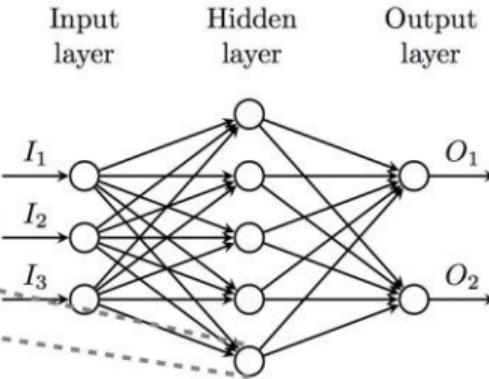
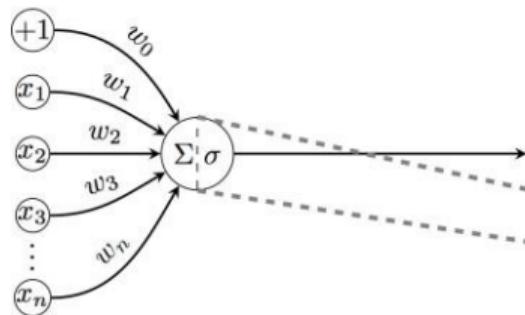
# Summary: Multilayer Perceptron in a Nutshell



**Fully Connected  
Layer  $j$  in a  
Multilayer Perceptron:**

$$x_j = \underbrace{\sum_i w_{ji} y_i}_{\text{weighted sum}}$$

# Summary: Multilayer Perceptron in a Nutshell

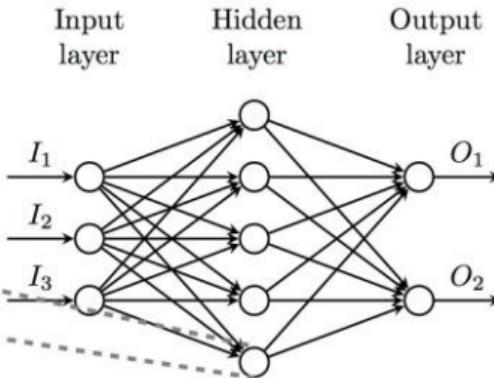
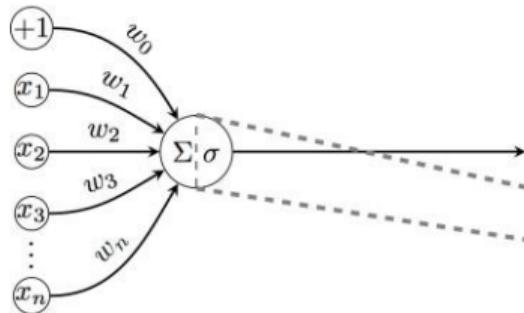


Previous Layer  
Outputs ↓

Fully Connected  
Layer  $j$  in a  
Multilayer Perceptron:

$$x_j = \underbrace{\sum_i w_{ji} y_i}_{\text{weighted sum}}$$

# Summary: Multilayer Perceptron in a Nutshell



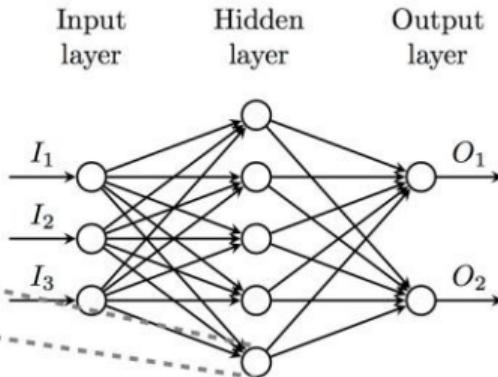
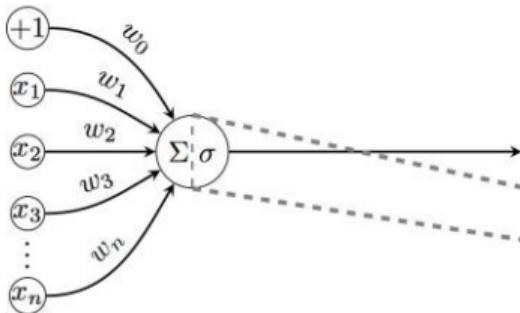
**Fully Connected Layer  $j$  in a Multilayer Perceptron:**

Previous Layer  
Outputs ↓

$$x_j = \underbrace{\sum_i w_{ji} y_i}_{\text{weighted sum}}$$

$$y_j = \sigma(x_j) = \underbrace{\frac{1}{1 + \exp\{-x_j\}}}_{\text{activation function}}$$

# Summary: Multilayer Perceptron in a Nutshell



Fully Connected Layer  $j$  in a Multilayer Perceptron:

Previous Layer Outputs ↓

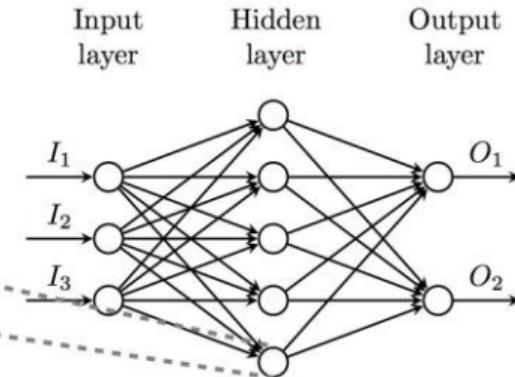
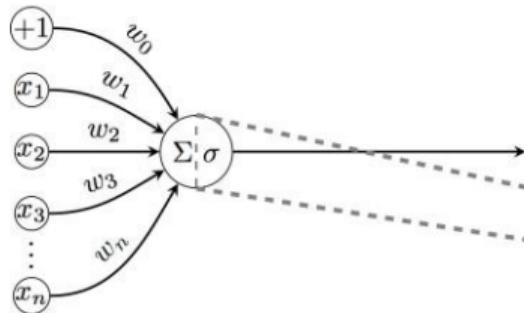
$$x_j = \underbrace{\sum_i w_{ji} y_i}_{\text{weighted sum}}$$

$$y_j = \sigma(x_j) = \frac{1}{1 + \exp\{-x_j\}}$$

activation function

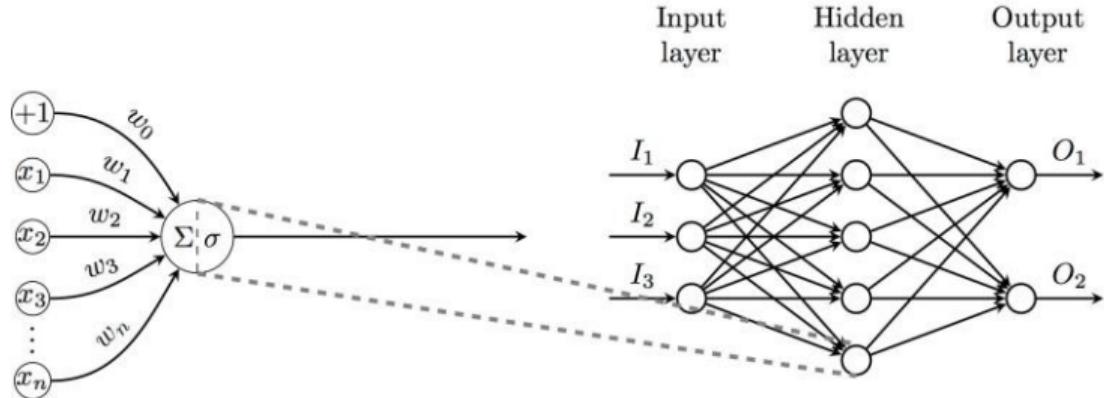
New Outputs!

# Summary: Multilayer Perceptron in a Nutshell



What Happens When Input is Images?

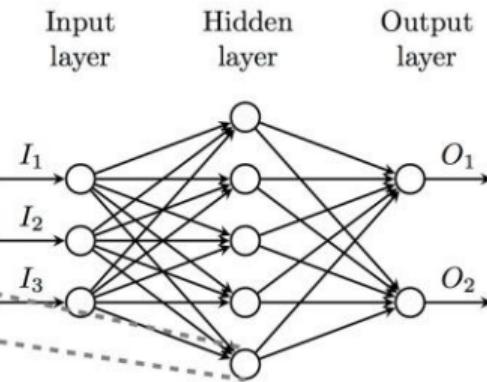
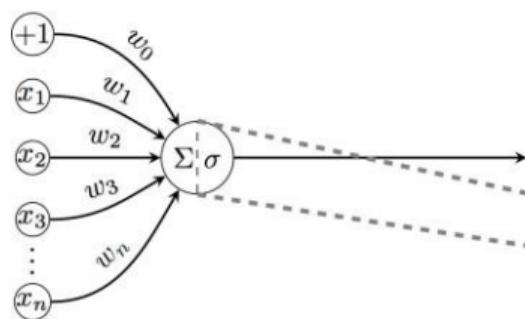
# Summary: Multilayer Perceptron in a Nutshell



What Happens When Input is Images?

Total Parameters Per Layer:

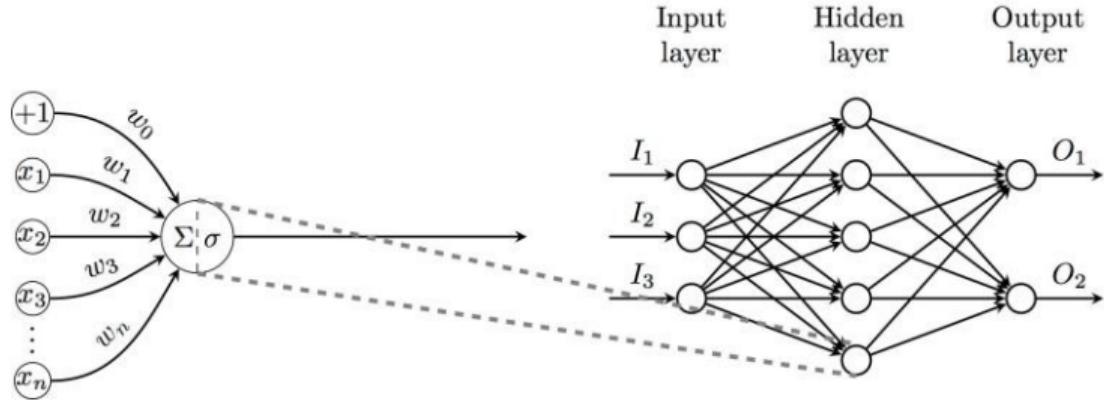
## Summary: Multilayer Perceptron in a Nutshell



What Happens When Input is Images?

Total Parameters Per Layer: (Input Size)\*(Output Size)

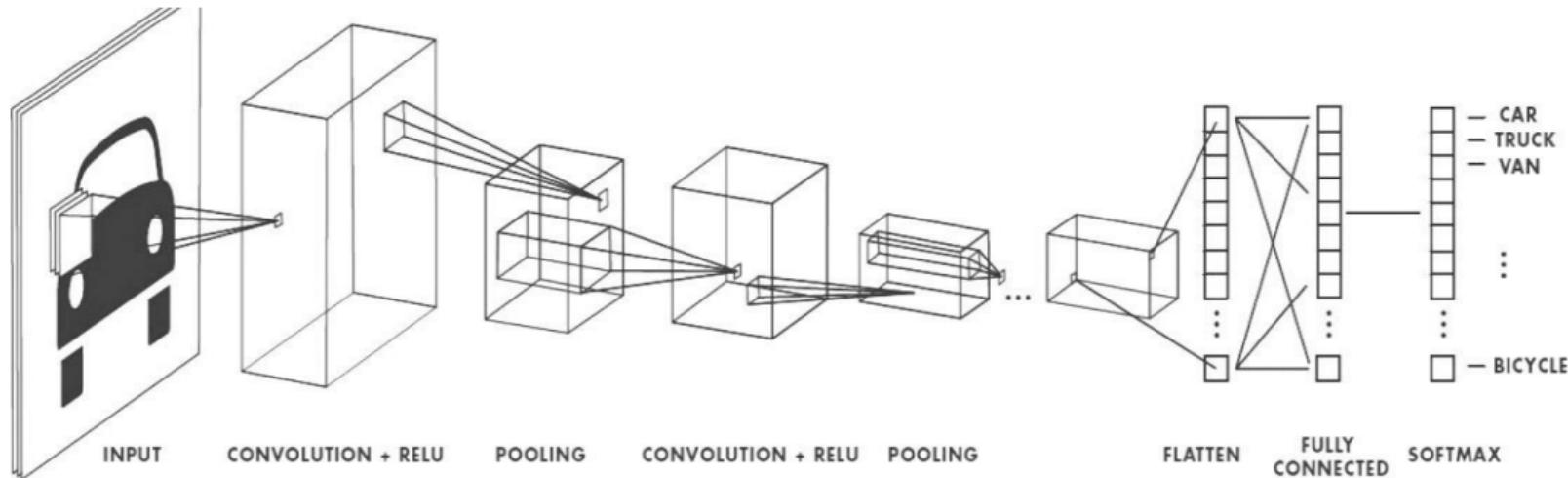
## Summary: Multilayer Perceptron in a Nutshell



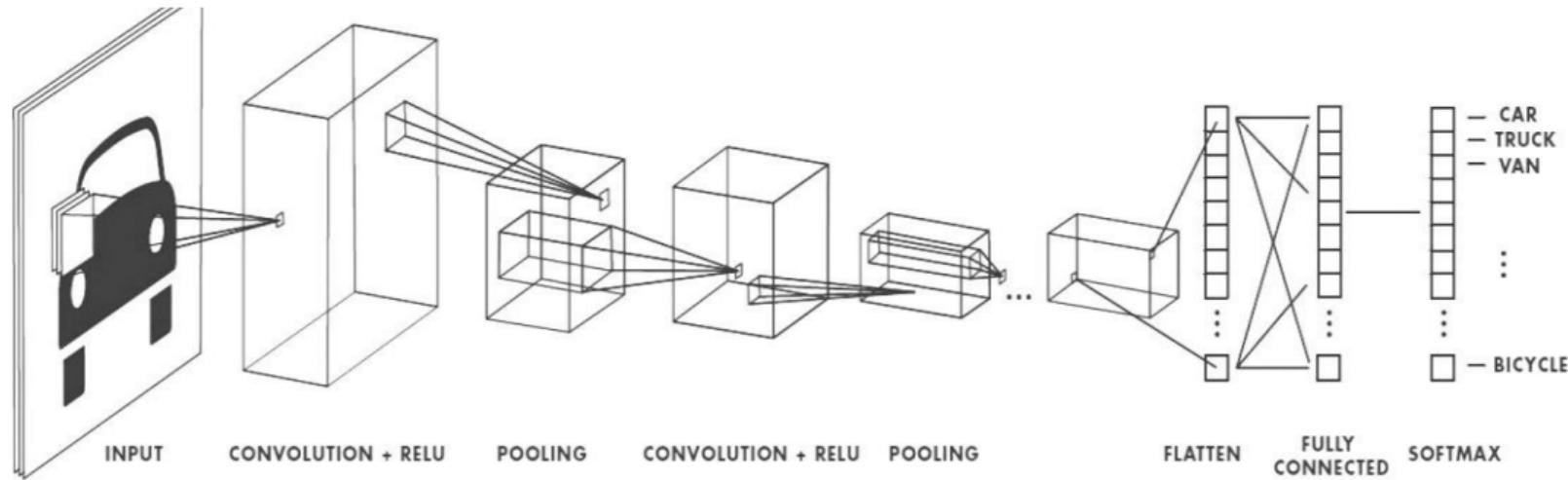
What Happens When Input is Images?

Total Parameters Per Layer: (Input Size)\*(Output Size)  
e.g.,  $(512 \times 512 \times 3) * (512 \times 1)$

# Convolutional Neural Networks

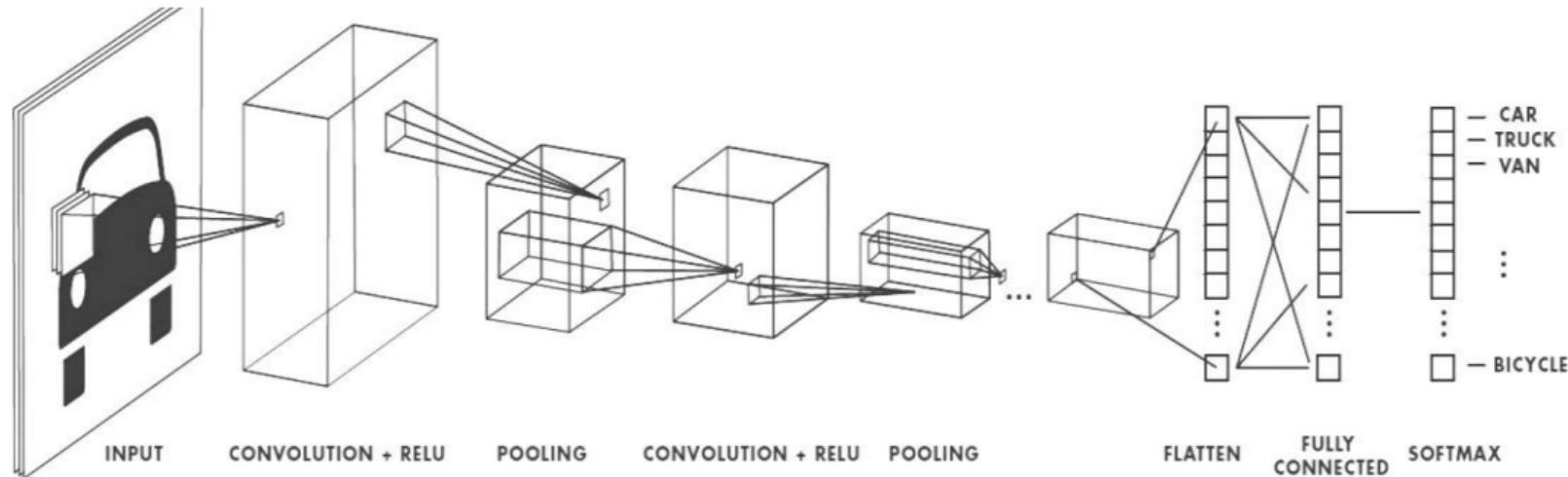


# Convolutional Neural Networks



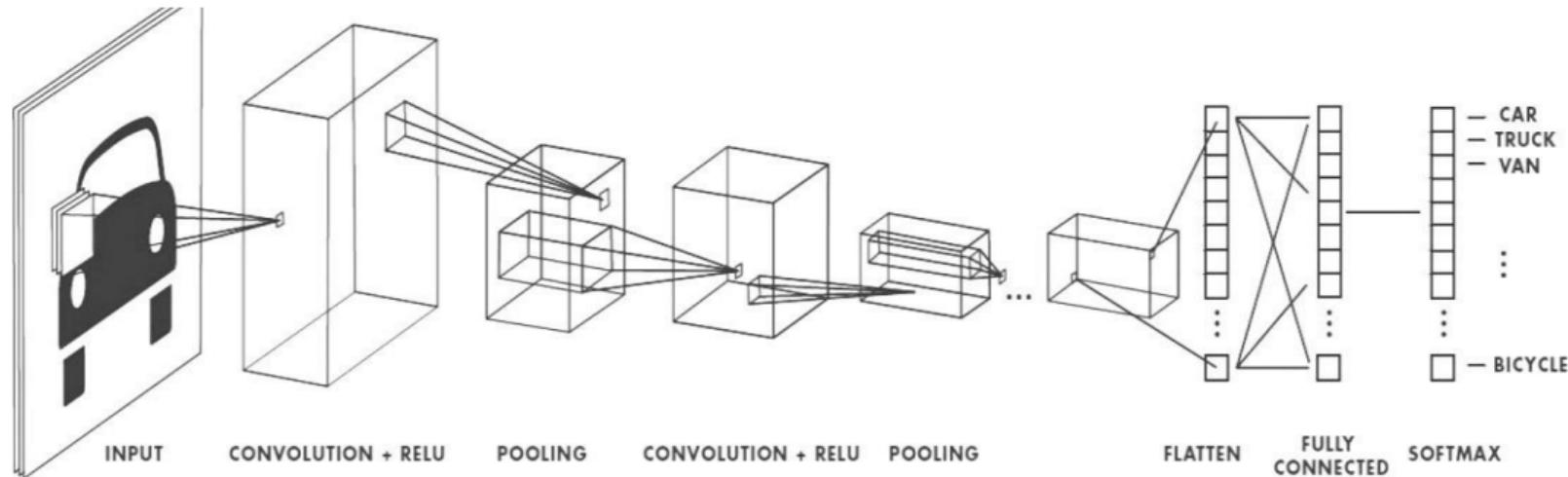
- ▶ ConvNets represent data in 3 dimensions: width, height, depth (= feature maps)

# Convolutional Neural Networks



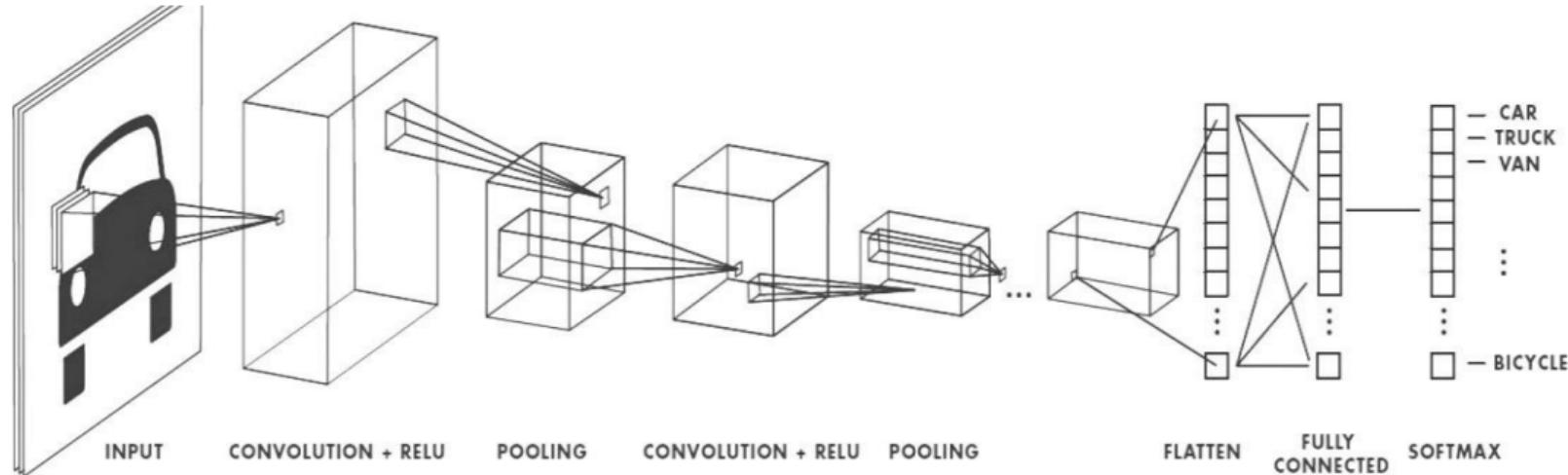
- ▶ ConvNets represent data in 3 dimensions: width, height, depth (= feature maps)
- ▶ ConvNets interleave discrete convolutions, non-linearities and pooling

# Convolutional Neural Networks



- ▶ ConvNets represent data in 3 dimensions: width, height, depth (= feature maps)
- ▶ ConvNets interleave discrete convolutions, non-linearities and pooling
- ▶ The learned weights now correspond to the convolution kernel parameters

# Convolutional Neural Networks

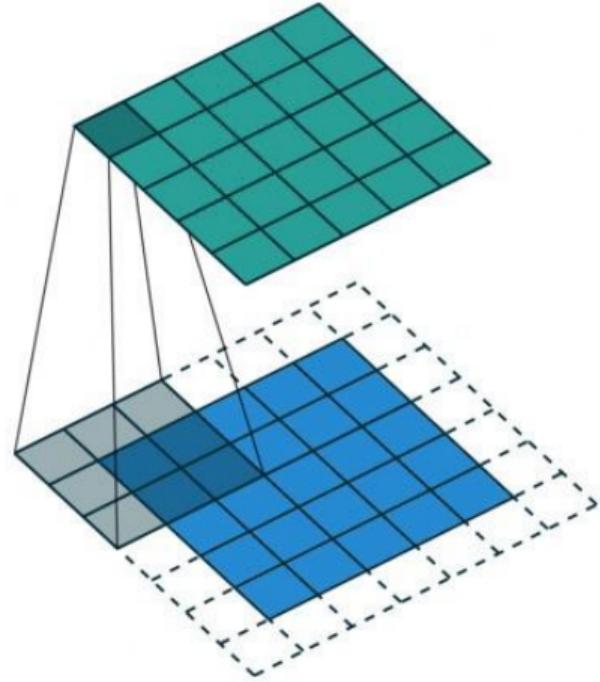


- ▶ ConvNets represent data in 3 dimensions: width, height, depth (= feature maps)
- ▶ ConvNets interleave discrete convolutions, non-linearities and pooling
- ▶ The learned weights now correspond to the convolution kernel parameters
- ▶ Biological example: simple cells and complex cells in primary visual cortex (V1)

## Building Block: Convolution Layer

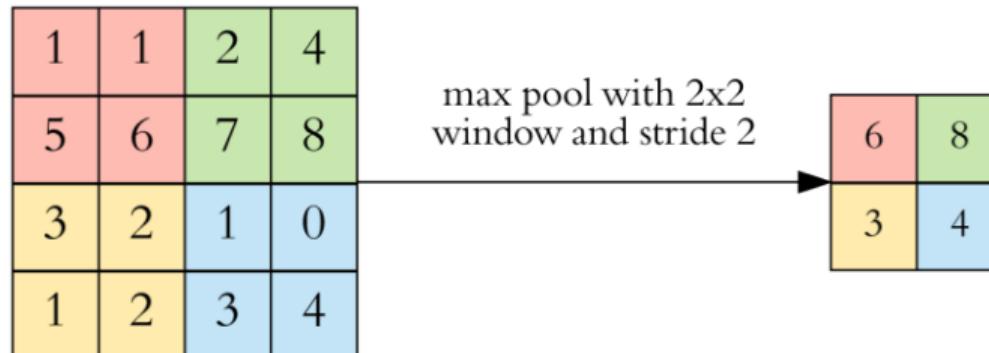
$$x_j(u, v, \cdot) = \sum_{u', v', d} k(u', v', d) y_i(u + u', v + v', d)$$

- ▶ With pixel  $(u, v)$ , channel  $d$ , kernel  $k(u, v, d)$
- ▶ Should be called “cross-correlation”
- ▶ Followed by non-linearity (e.g., ReLU)
- ▶ Right: Example for convolution with stride 1
  - ▶ Here: one feature map as input but in practice multiple feature maps



Total Parameters Per Layer:  
(Kernel Size)\*(Output Channel Size)  
e.g.,  $(3 \times 3 \times 3) * (1)$

# Building Block: Pooling Layer



- Combine multiple inputs locally into one number
- Most common pooling operations: maximum, minimum, average
- Stride 2 increases receptive field size by factor 2 in each dimension
- How many trainable parameters/weights?

# LeNet-5

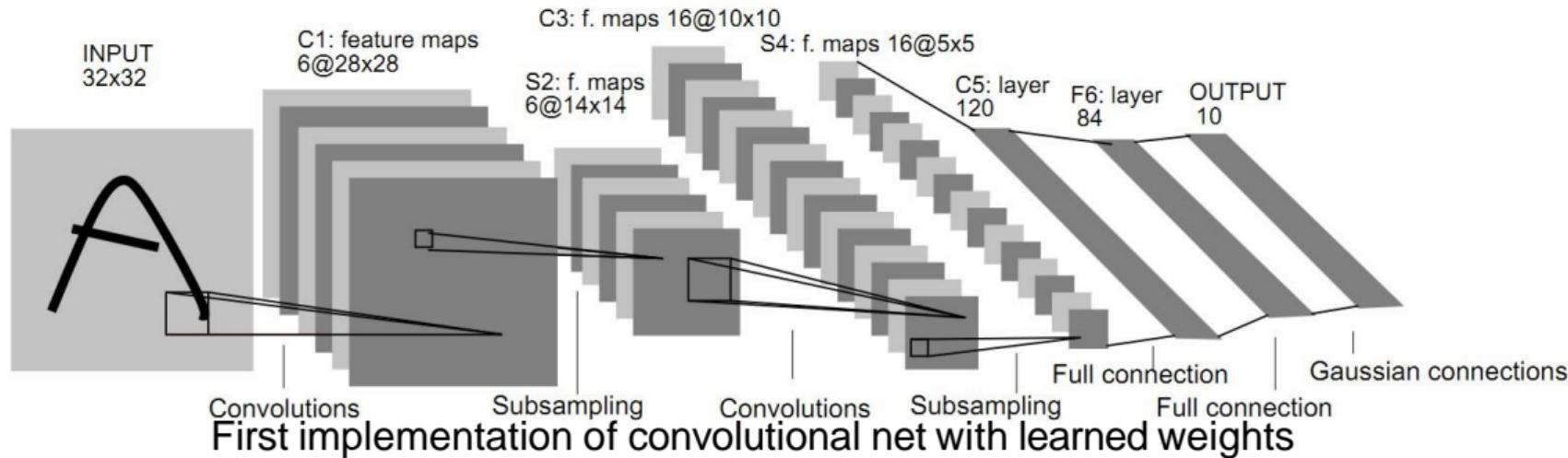


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are convolved to produce the output. **2 convolution layers ( $5 \times 5$ ), 2 subsampling layers (stride 2), 2 fully connected layers**

# VGG-19

- Evaluating networks of increasing depth (today: 100-1000 layers)
- Demonstrated significant improvement by increasing the depth to 16-19 weight layers
- Key to keep number of parameters tractable: Small  $3 \times 3$  filters in all convolutional layers (stride 1)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 LRN	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

# Multilayer Perceptron - Learning

## Learning:

- ▶ Define error function over all data points  $\{(\mathbf{x}_n, \mathbf{t}_n)\}_{n=1}^N$ , e.g.:

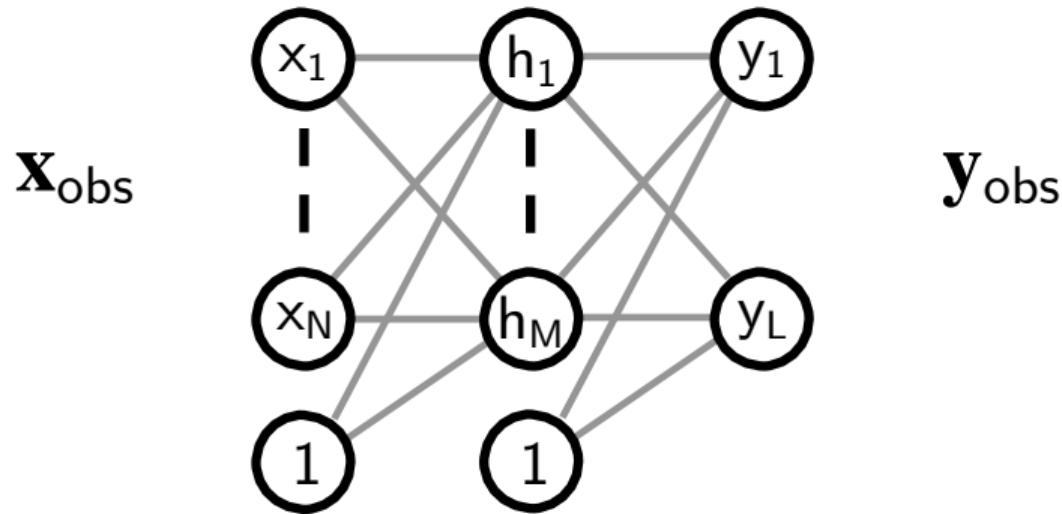
$$E = \frac{1}{2} \sum_n \sum_j (y_{jn} - t_{jn})^2$$

- ▶ Here  $y_{jn}$  is an element of the last layer  $j$  of the network
- ▶ Calculate  $\nabla_{\mathbf{W}} E = \sum_n \nabla_{\mathbf{W}} E_n$  with  $E_n = \frac{1}{2} \sum_j (y_{jn} - t_{jn})^2$
- ▶ Gradient descent:  $\mathbf{W}^{t+1} = \mathbf{W}^t - \eta \nabla_{\mathbf{W}} E|_{\mathbf{W}^t}$  ( $\eta$  = "learning rate")

## Problem?

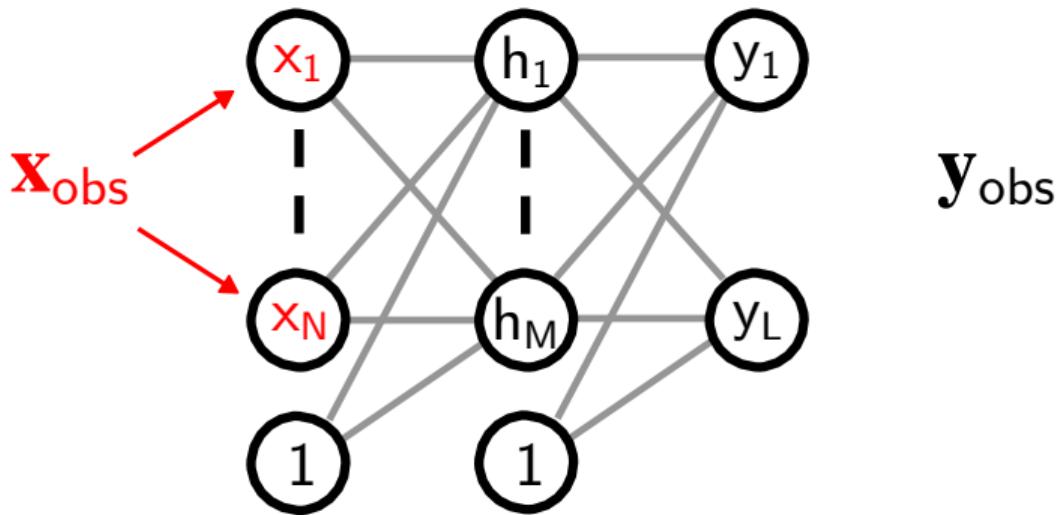
- ▶ Gradient needs to be computed for all data points (problematic when  $N$  is large)
  - ▶ Solution: Perform Stochastic Gradient Descent (SGD) using mini batches:
  - ▶ Approximate gradient with  $M \ll N$  randomly picked data points at each iteration (typically  $M = 16, \dots, 128$  as gradients too noisy for  $M = 1$ )

## Illustration: Forward Pass



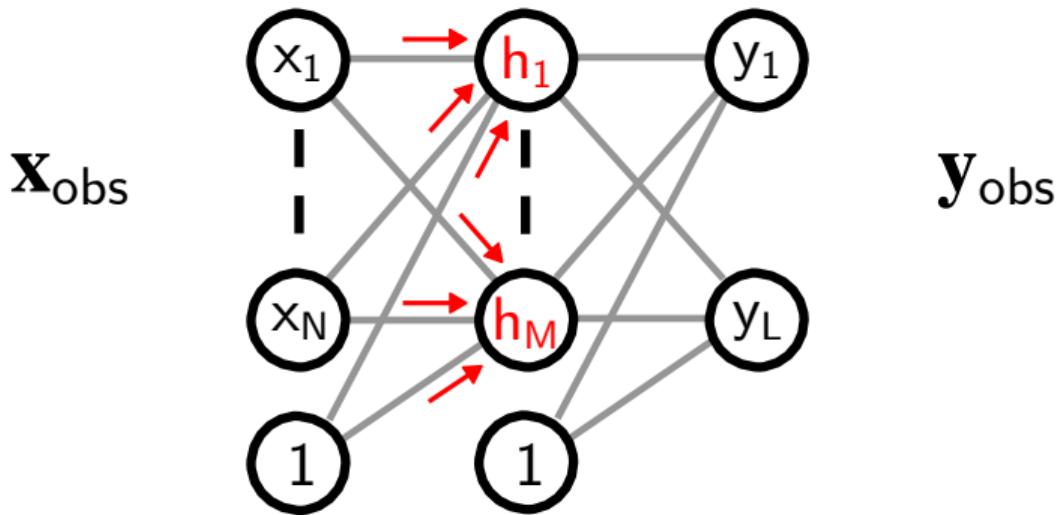
- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

## Illustration: Forward Pass



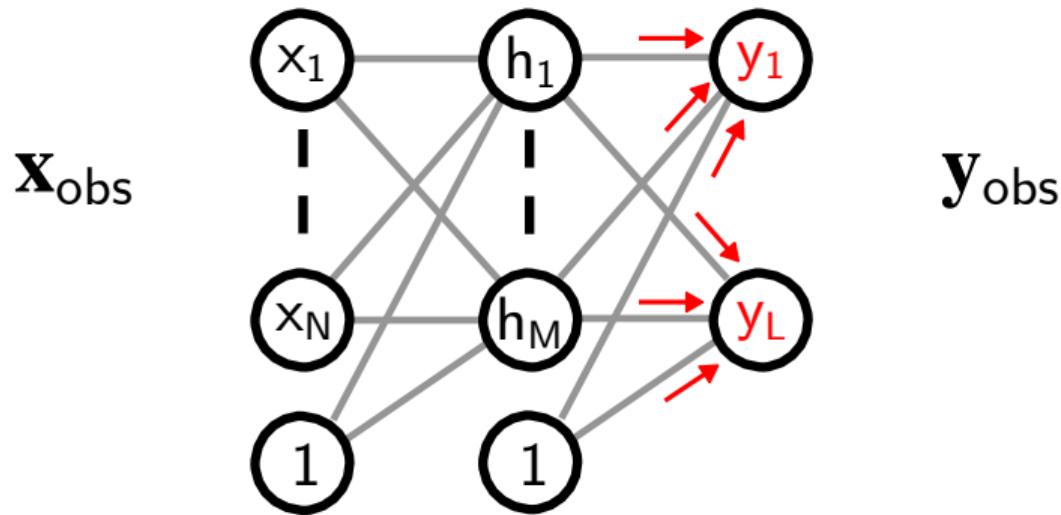
- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

## Illustration: Forward Pass



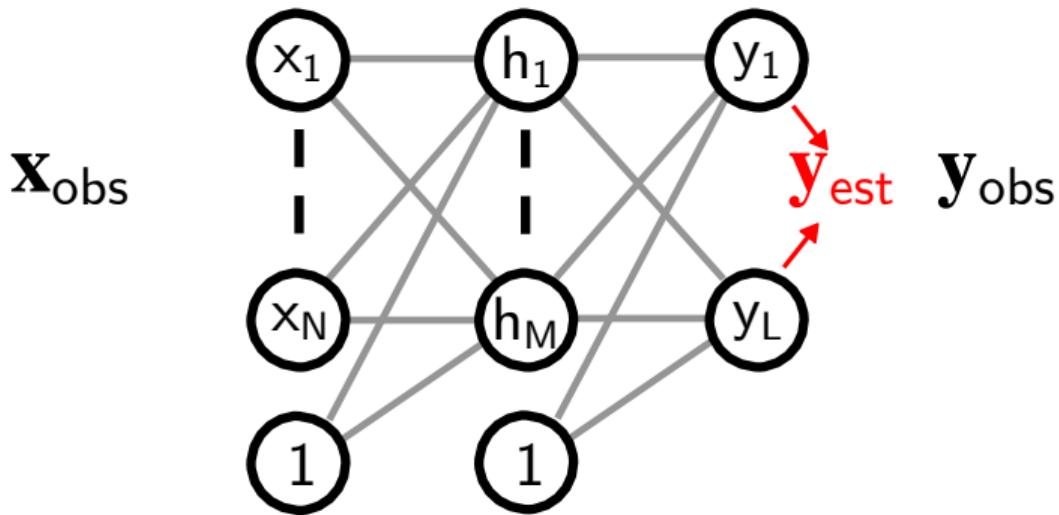
- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

## Illustration: Forward Pass



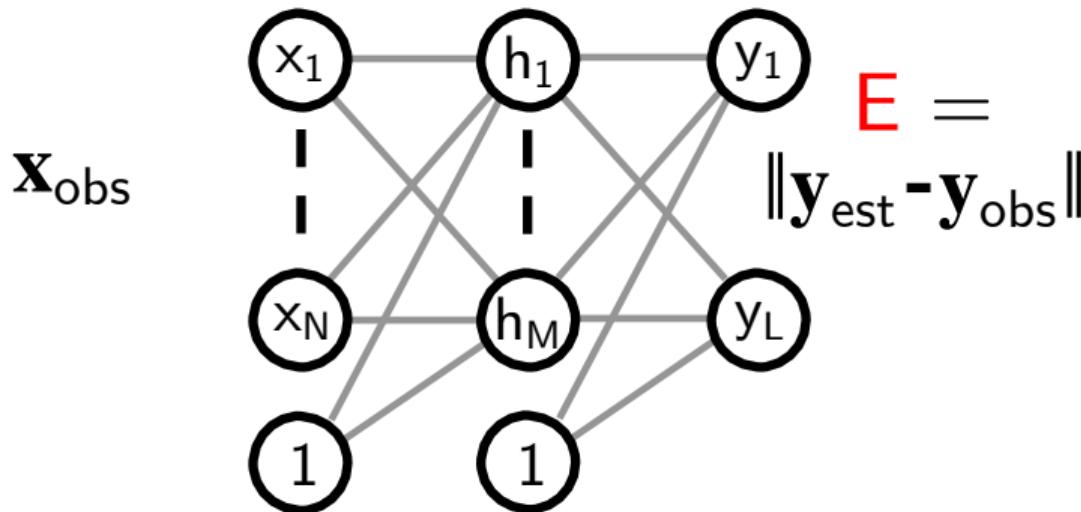
- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

## Illustration: Forward Pass



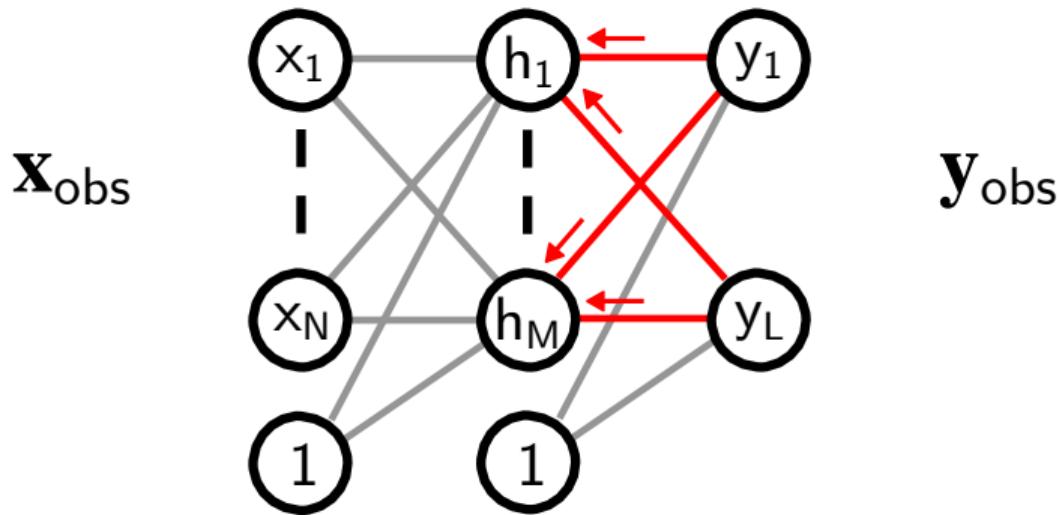
- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

## Illustration: Error Calculation



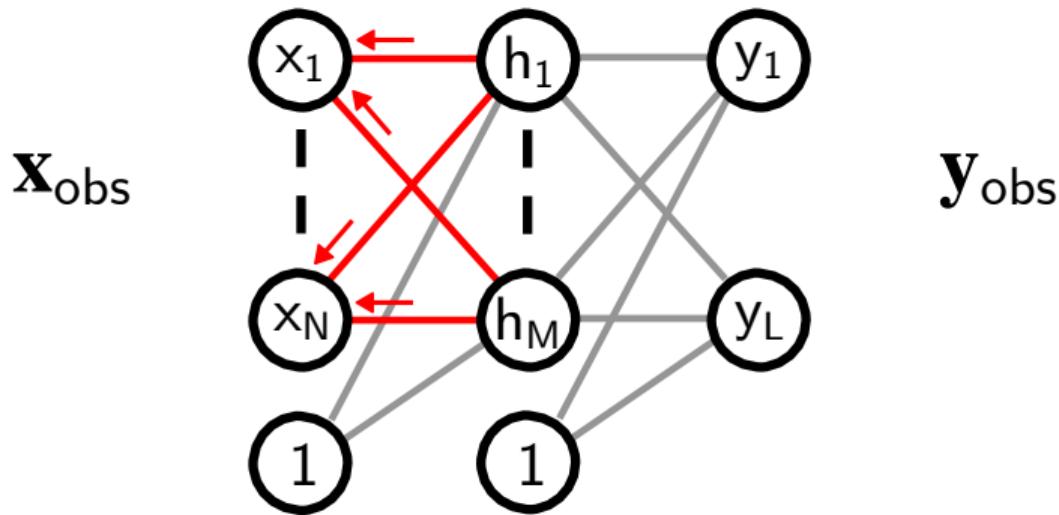
- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

## Illustration: Error Backpropagation



- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

## Illustration: Error Backpropagation

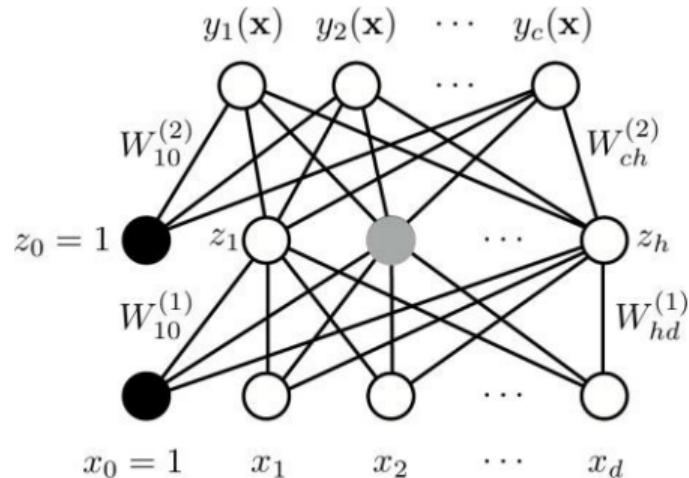


- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

# Dropout

## Idea:

- During forward pass: set every hidden unit to zero with probability 0.5
- Backward pass as before
- Avoids co-adaptation of neurons, so each is forced to independently learn useful weights

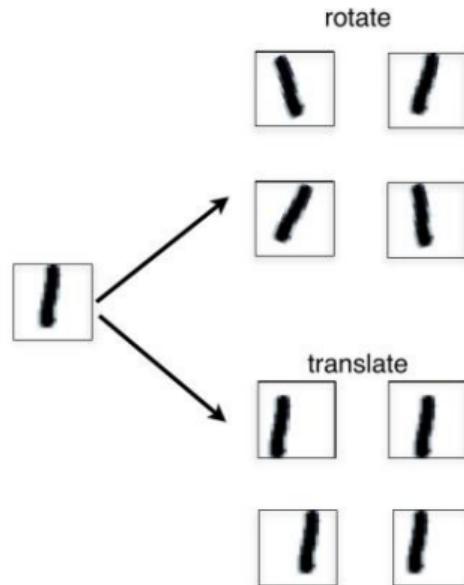


# Data Augmentation

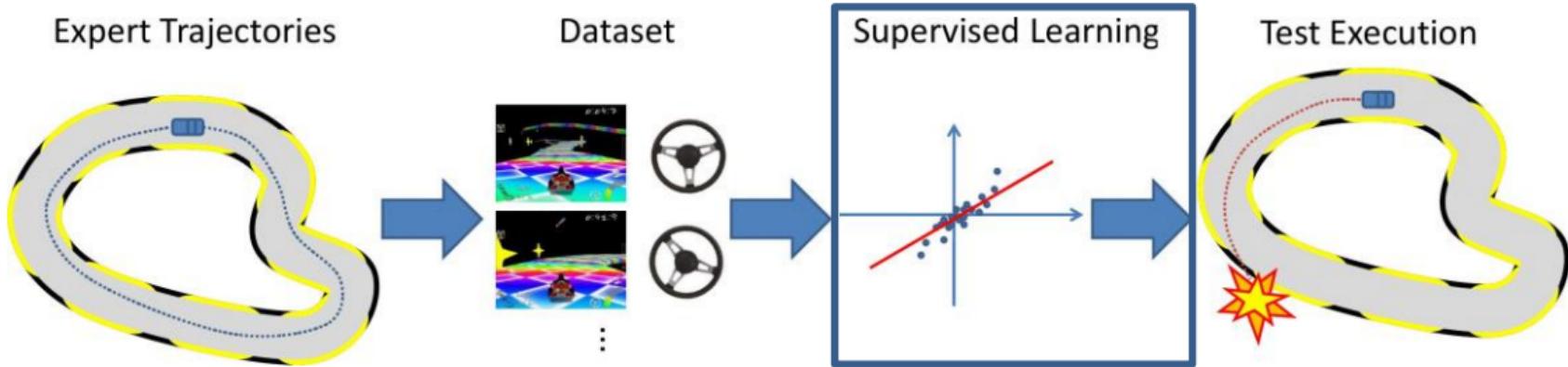
Idea:

- Augment training set by “jittering”
- Transformation must be label preserving

1 1 5 4 3  
7 5 3 5 3  
5 5 9 0 6  
3 5 2 0 0

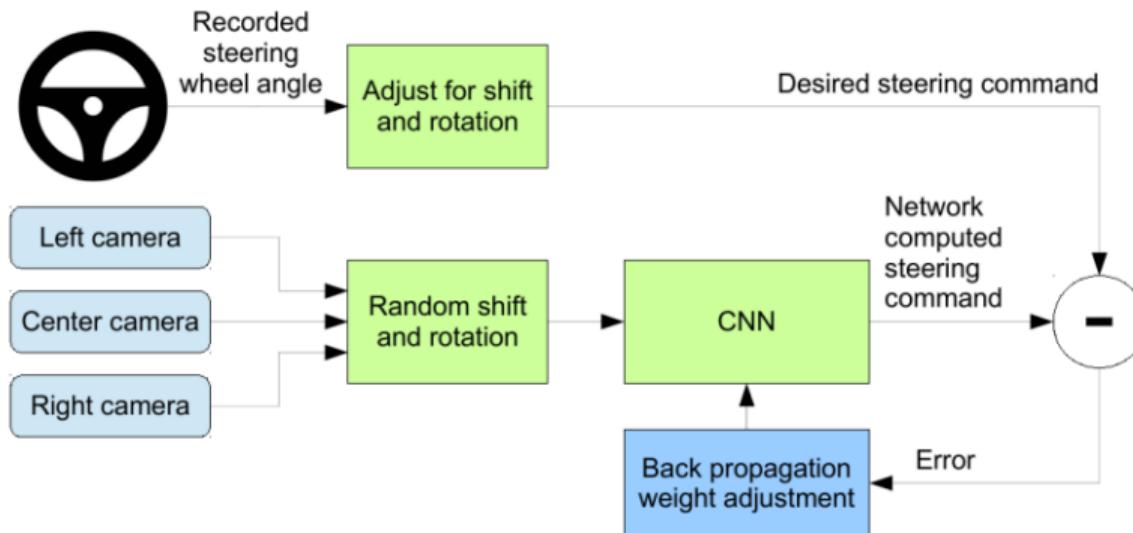


# Behavior Cloning in a Nutshell



- Assumes actions in the expert trajectories are i.i.d (Independent and identically distributed)
- Train a classifier or regressor to map observations to actions at each time step of the trajectory

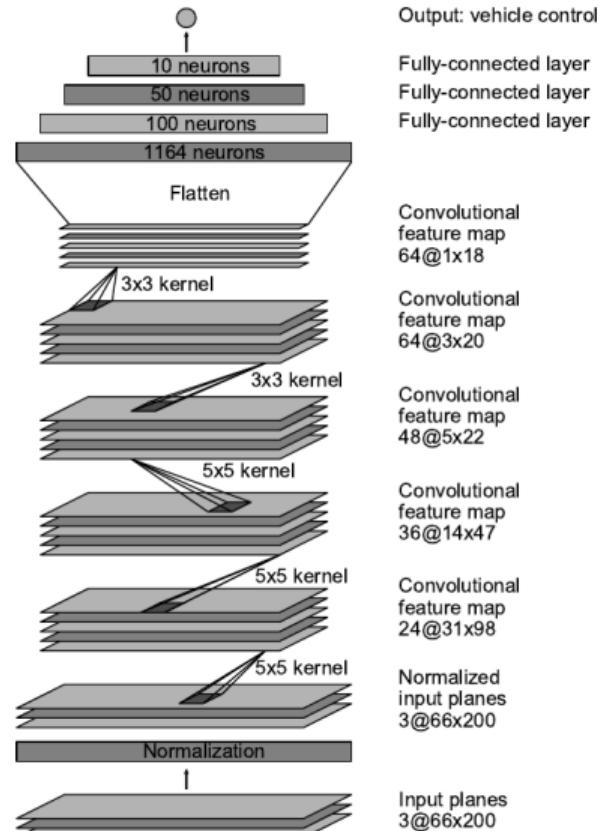
# PilotNet: System Overview



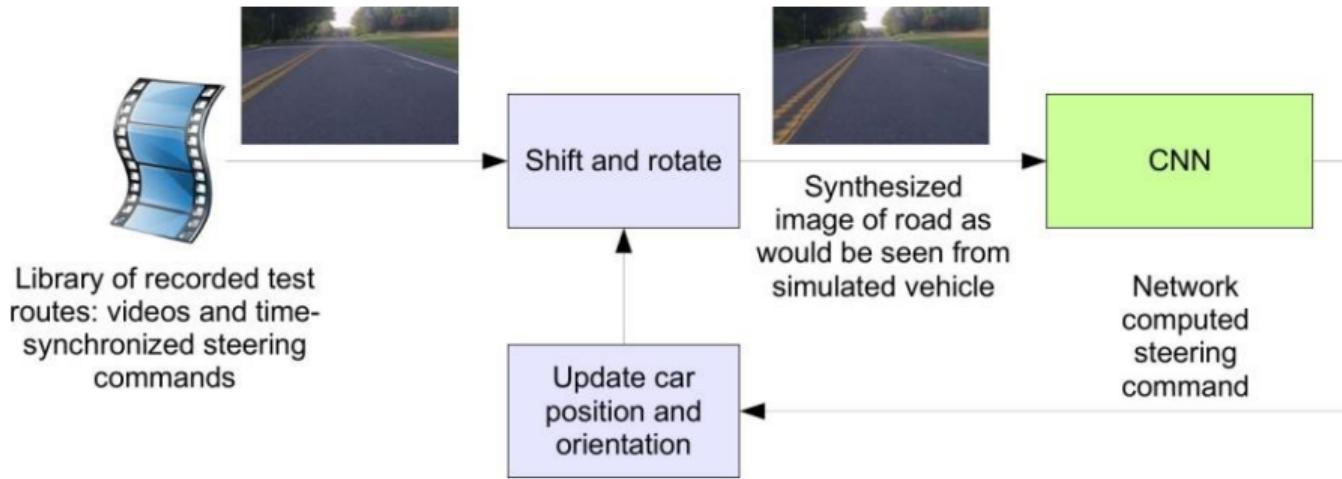
- ▶ Steering command represented as  $1/r$  where  $r$  = turning radius in meters
- ▶ Data augmentation by 3 cameras and virtually shifted / rotated images assuming the world is flat (homography), adjusting the steering angle appropriately

# PilotNet: Architecture

- Convolutional network (250k param)
- Input: YUV image representation
- 1 Normalization Layer
  - Not learned
- 5 Convolutional Layers
  - 3 strided 5x5
  - 2 non-strided 3x3
- 3 Fully connected Layers
- Output: turning radius
- No lateral control!
- Trained on 72h of driving



# PilotNet: Simulation



- Simulator takes pre-recorded videos and extracts lane center
- Creates novel viewpoints and applies CNN commands to vehicle model
- Updates the next image based on the resulting vehicle pose

# PilotNet: Video

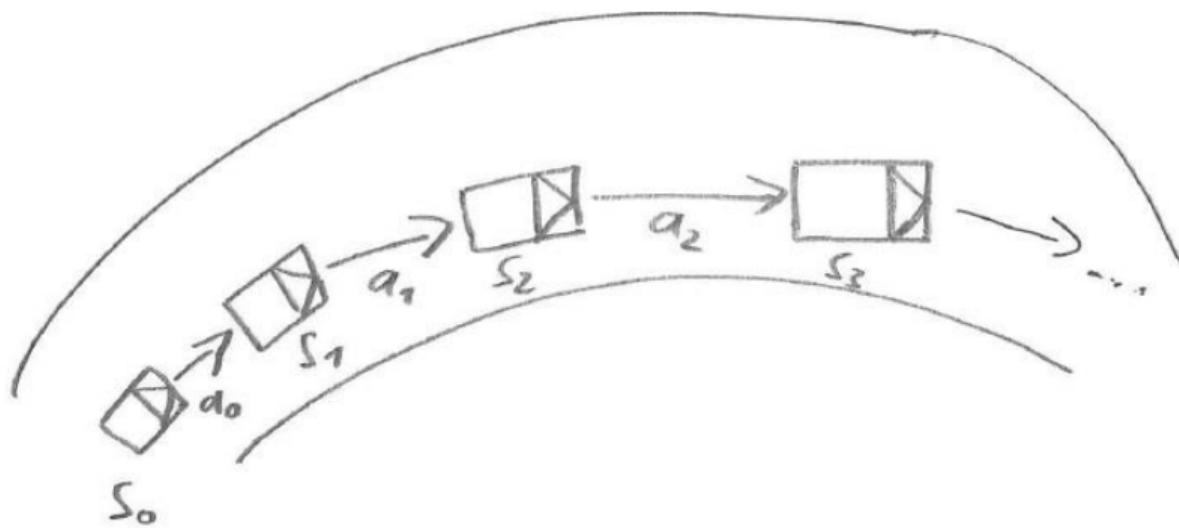


# Issue of Domain/Covariate Shift

- Humans can do things that robots can't do  
(and vice versa)
- Humans understand differently from robots

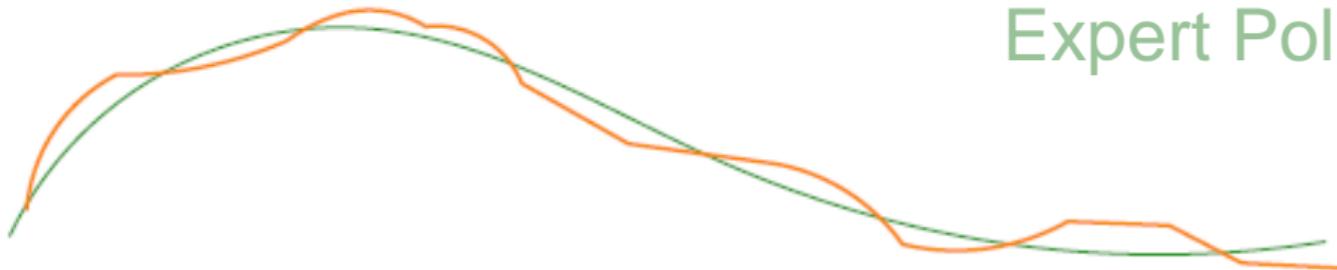
Problem: Actions are interdependent along the trajectory!

But, behavior cloning only learns/predicts one step at a time?  
(that is, trained over pairs of  $(s_t, a_t)$ )



# Errors Independent in Time

Neural Net Policy  
Expert Policy

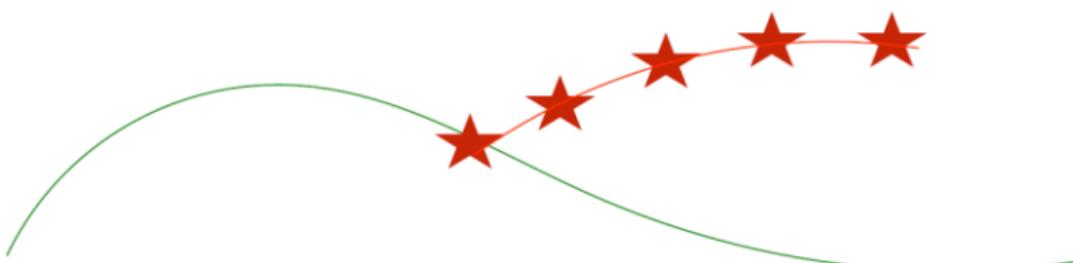


Error at time  $t$  with probability  $\epsilon$

$$E[\text{Total errors}] \lesssim \epsilon T$$

# Compounding Errors

Neural Net Policy  
Expert Policy

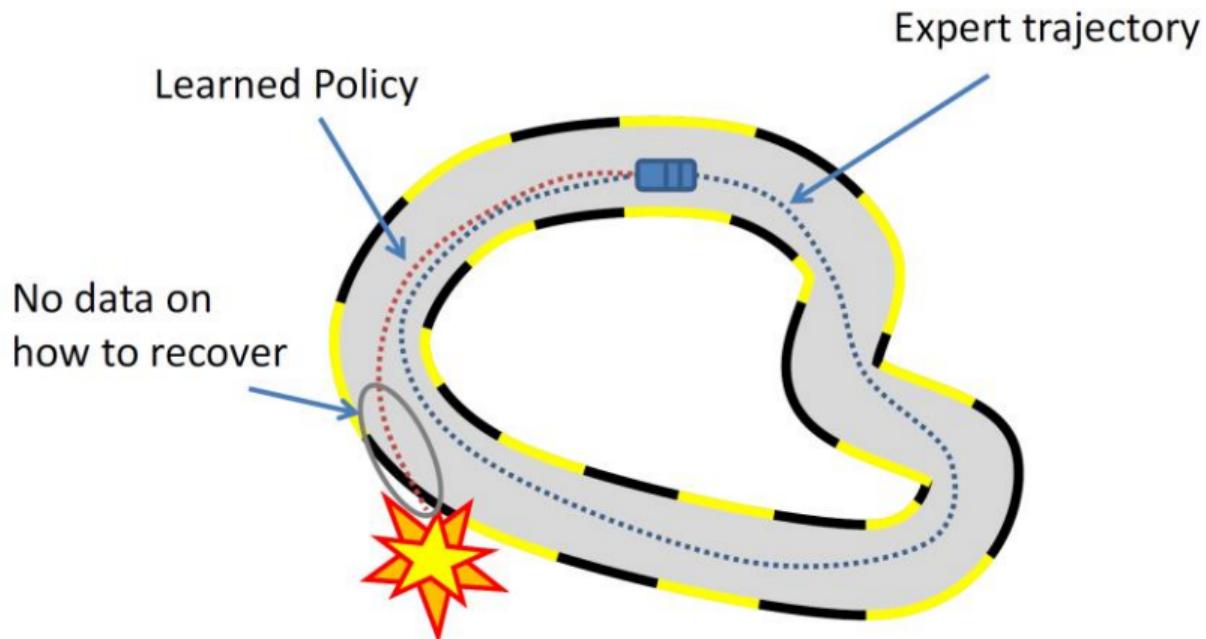


error at time  $t$  with probability  $\varepsilon$

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

# Data Distribution Mismatch

$$P(s|\pi_\theta) \neq P(s|\pi^*)$$



# Formal Definition of Imitation Learning

## General Imitation Learning:

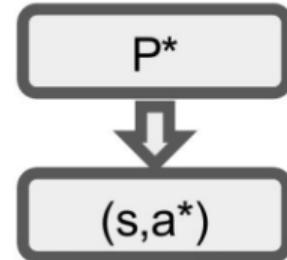
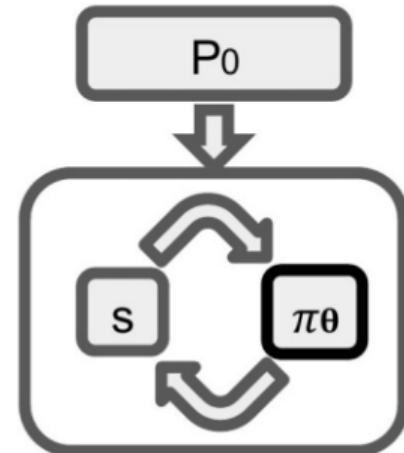
$$\operatorname{argmin}_{\theta} \mathbb{E}_{s \sim P(s|\pi_{\theta})} [\mathcal{L}(\pi^*(s), \pi_{\theta}(s))]$$

- ▶ State distribution  $P(s|\pi_{\theta})$  depends on rollout determined by current policy  $\pi_{\theta}$

## Behavior Cloning:

$$\operatorname{argmin}_{\theta} \underbrace{\mathbb{E}_{(s^*, a^*) \sim P^*} [\mathcal{L}(a^*, \pi_{\theta}(s^*))]}_{= \sum_{i=1}^N \mathcal{L}(a_i^*, \pi_{\theta}(s_i^*))}$$

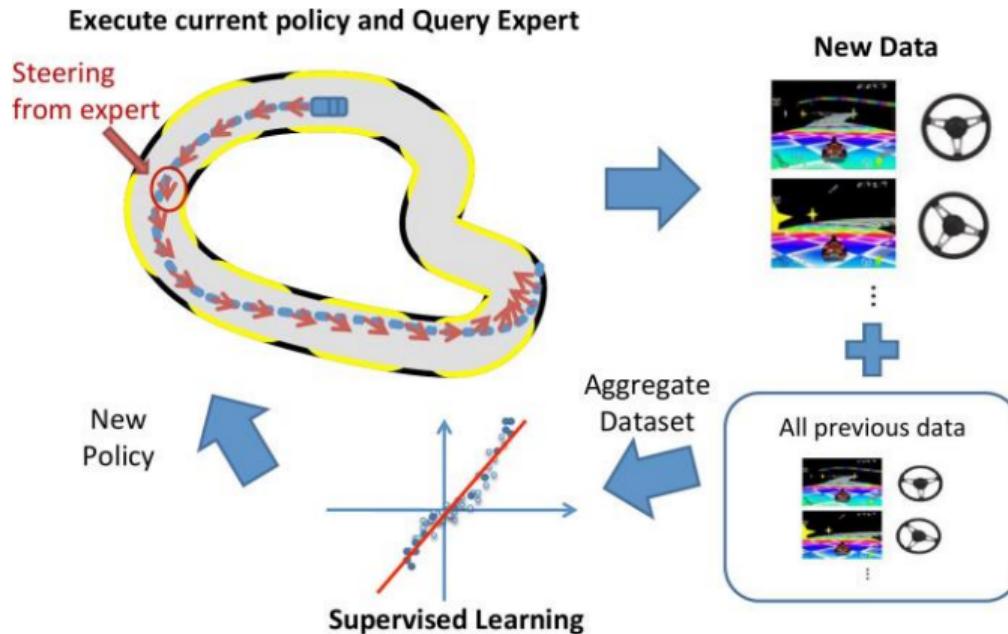
- ▶ State distribution  $P^*$  provided by expert
- ▶ Reduces to supervised learning problem



## Challenges of Behavior Cloning

- ▶ Behavior cloning reasons only about immediate next step
- ▶ Behavior cloning makes IID assumption
  - ▶ Next state is sampled from states observed during expert demonstration
  - ▶ Thus, next state is sampled independently from action predicted by current policy
- ▶ What if  $\pi_\theta$  makes a mistake?
  - ▶ Enters new states that haven't been observed before
  - ▶ New states not sampled from same (expert) distribution anymore
  - ▶ Cannot recover, catastrophic failure in the worst case
- ▶ What can we do to overcome this train/test distribution mismatch?

# DAGGER: Dataset Aggregation



**Idea:** Iteratively build a set of inputs that the final policy is likely to encounter based on previous experience. Query expert for aggregate dataset.

# DAGGER: Dataset Aggregation

```
Initialize  $\mathcal{D} \leftarrow \emptyset$ .
Initialize  $\hat{\pi}_1$  to any policy in  $\Pi$ .
for  $i = 1$  to  $N$  do
    Let  $\pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_i$ .
    Sample  $T$ -step trajectories using  $\pi_i$ .
    Get dataset  $\mathcal{D}_i = \{(s, \pi^*(s))\}$  of visited states by  $\pi_i$ 
    and actions given by expert.
    Aggregate datasets:  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$ .
    Train classifier  $\hat{\pi}_{i+1}$  on  $\mathcal{D}$ .
end for
Return best  $\hat{\pi}_i$  on validation.
```

- ▶ Let  $T$  be the number of rollout time steps and  $\epsilon$  be the probability of a mistake
- ▶ Behavior cloning:  $O(T^2\epsilon)$  regret, DAGGER: no regret (in suitable conditions)
  - ▶ Regret = loss of current policy - loss of perfect policy
- ▶ But requires to flexibly run the environment and query the expert during training

# DAGGER: In Simulation

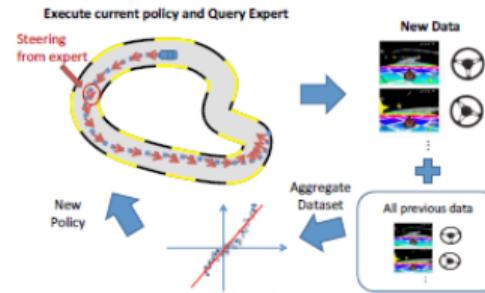
Dataset AGGregation: bring learner's and expert's trajectory distributions closer by labelling additional data points resulting from applying the current policy

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.

Control Notation

**o** - observation

**u** - action



## Problems:

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

DAGGER: In Real Platform

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



Learning monocular reactive UAV control in cluttered natural environments, Ross et al. 2013



Fig. 2. One frame from MAV camera stream. The white line indicates the current yaw commanded by the current DAgger policy  $\pi_{n-1}$  while the red line indicates the expert commanded yaw. In this frame DAgger is wrongly heading for the tree in the middle while the expert is providing the correct yaw command to go to the right instead. These expert controls are recorded for training later iterations but not executed in the current run.

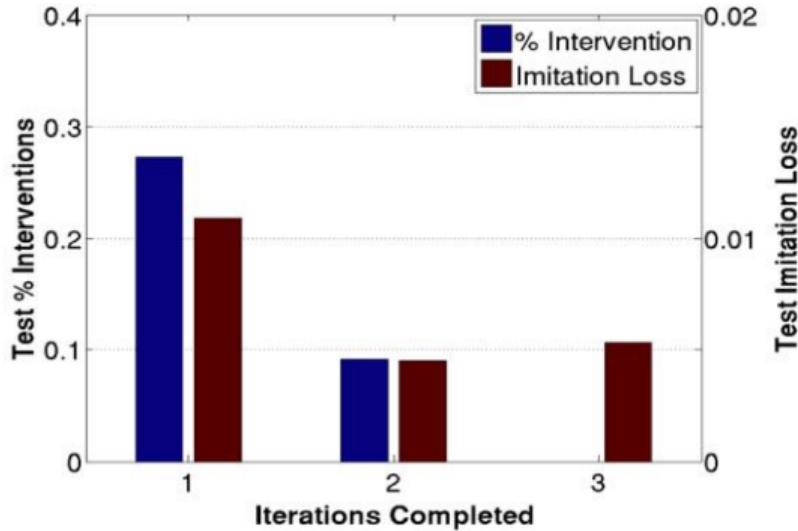
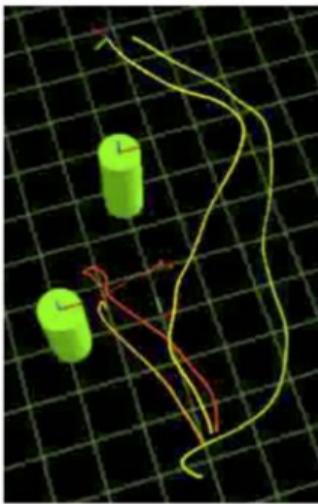


Fig. 4. Left: Improvement of trajectory by DAgger over the iterations. The rightmost green trajectory is the pilot demonstration. The short trajectories in red & orange show the controller learnt in the 1<sup>st</sup> and 2<sup>nd</sup> iterations which fail. The 3<sup>rd</sup> iteration controller successfully avoids both obstacles and is similar to the demonstrated trajectory. Right: Percentage of scenarios the pilot had to intervene and the imitation loss (average squared error in controls of controller to human expert on hold-out data) after each iteration of Dagger. After 3 iterations, there was no need for the pilot to intervene and the UAV could successfully avoid all obstacles

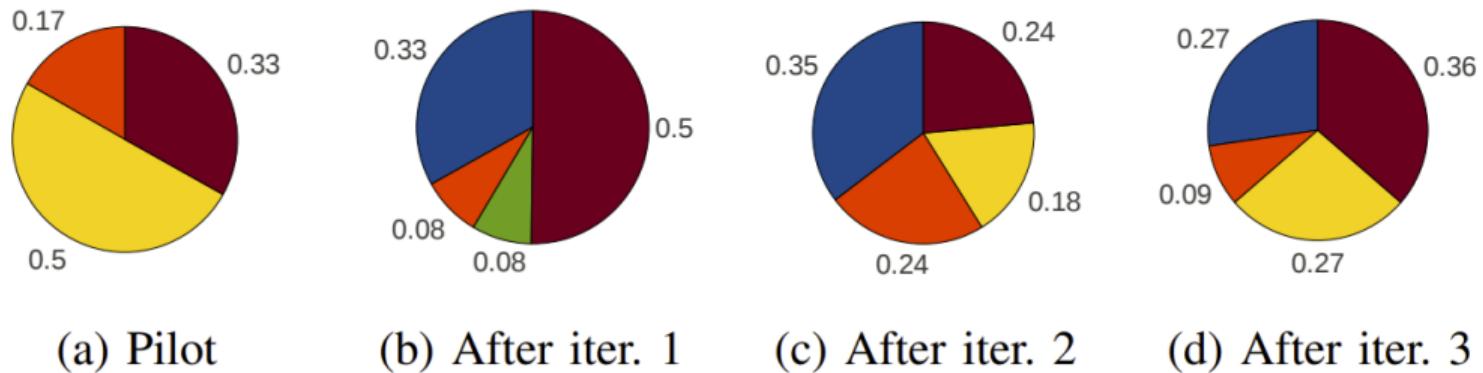
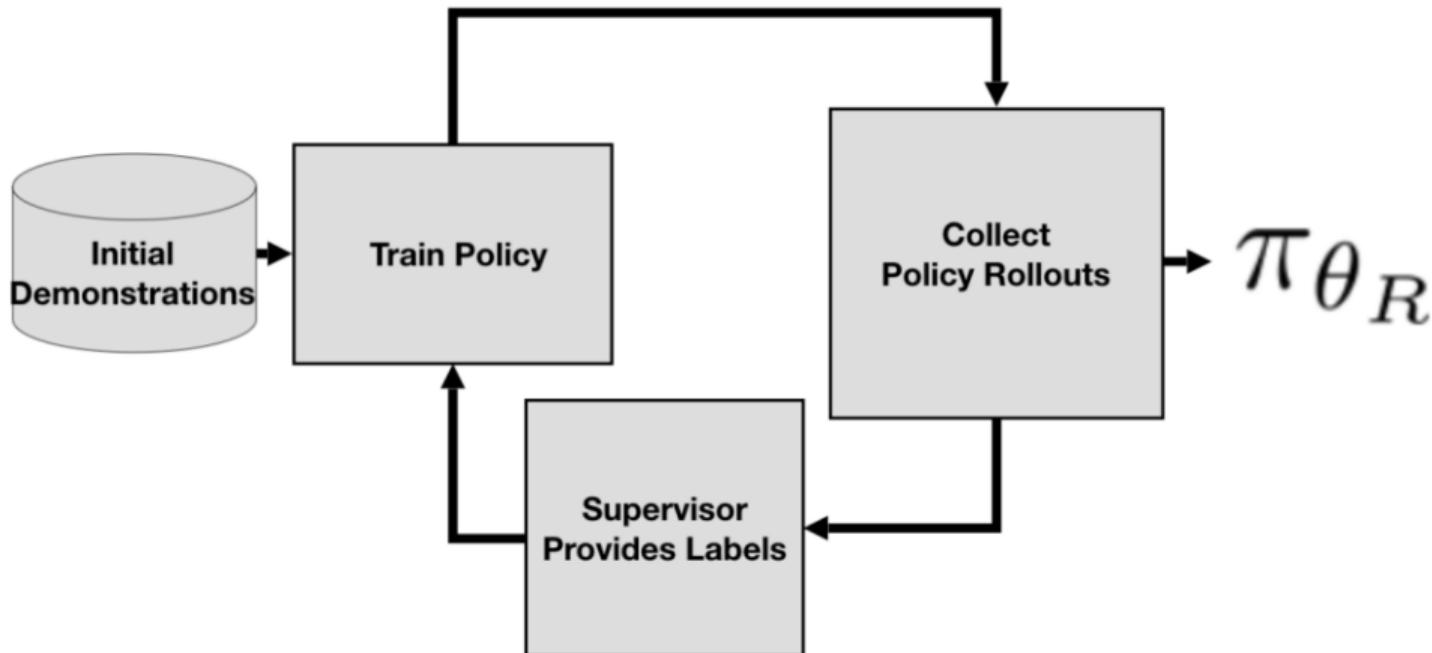


Fig. 8. Percentage of failures of each type for DAgger over the iterations of training in the high-density region. Blue: Large Trees, Orange: Thin Trees, Yellow: Leaves and Branches, Green: Other obstacles (poles, signs, etc.), Red: Too Narrow FOV. Clearly, a majority of the crashes happen due to a too narrow FOV and obstacles which are hard to perceive, such as branches and leaves.

# Reducing Shift with On-Policy Methods



*The DAgger Algorithm.*

DAGGER: In Real Platform

Safe? Executing an imperfect policy.

Difficult for the expert  
(especially for drone application, no feedback)

Expert's reaction time to the drone's behavior is large, this causes imperfect actions to be commanded.

Learning monocular reactive UAV control in cluttered natural environments, Ross et al. 2013

When else does  
behavior cloning fail?

# Conditional Imitation Learning

# Conditional Imitation Learning



(a) Aerial view of test environment

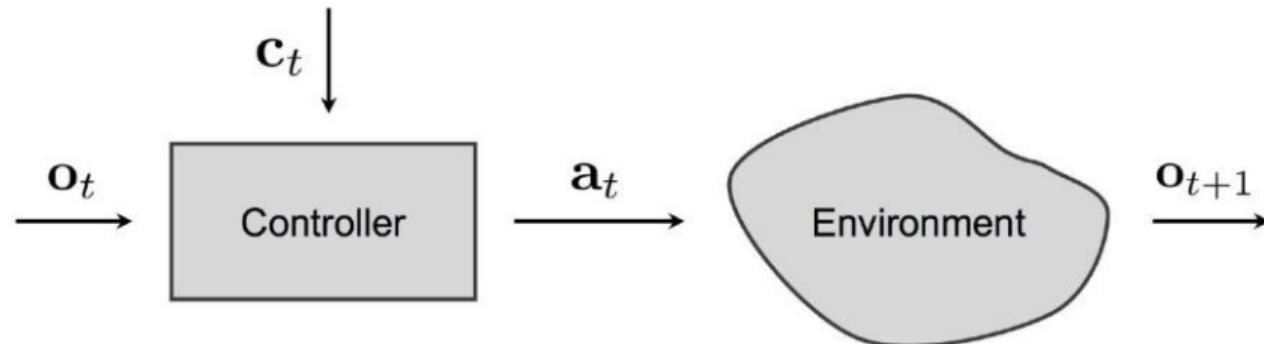


(b) Vision-based driving, view from onboard camera



(c) Side view of vehicle

# Conditional Imitation Learning



## Idea:

- ▶ Condition controller on **navigation command**  $c \in \{left, right, straight\}$
- ▶ High-level navigation command can be provided by consumer GPS, i.e., telling the vehicle to **turn left/right** or go **straight** at the next intersection
- ▶ This removes the task ambiguity induced by the environment (not: noise)
- ▶ Observation  $o_t$ : current image      Action  $a_t$ : steering angle & acceleration

# Comparison to Behavior Cloning

## BC

- ▶ Training Set:

$$\mathcal{D} = \{(a_i^*, s_i^*)_{i=1}^N\}$$

- ▶ Objective:

$$\operatorname{argmin}_{\theta} \sum_{i=1}^N \mathcal{L}(a_i^*, \pi_{\theta}(s_i^*))$$

- ▶ Assumption:

$$\exists f(\cdot) : a_i = f(s_i)$$

Often violated in practice! Why?

## Conditional BC

- ▶ Training Set:

$$\mathcal{D} = \{(a_i^*, s_i^*, c_i^*)_{i=1}^N\}$$

- ▶ Objective:

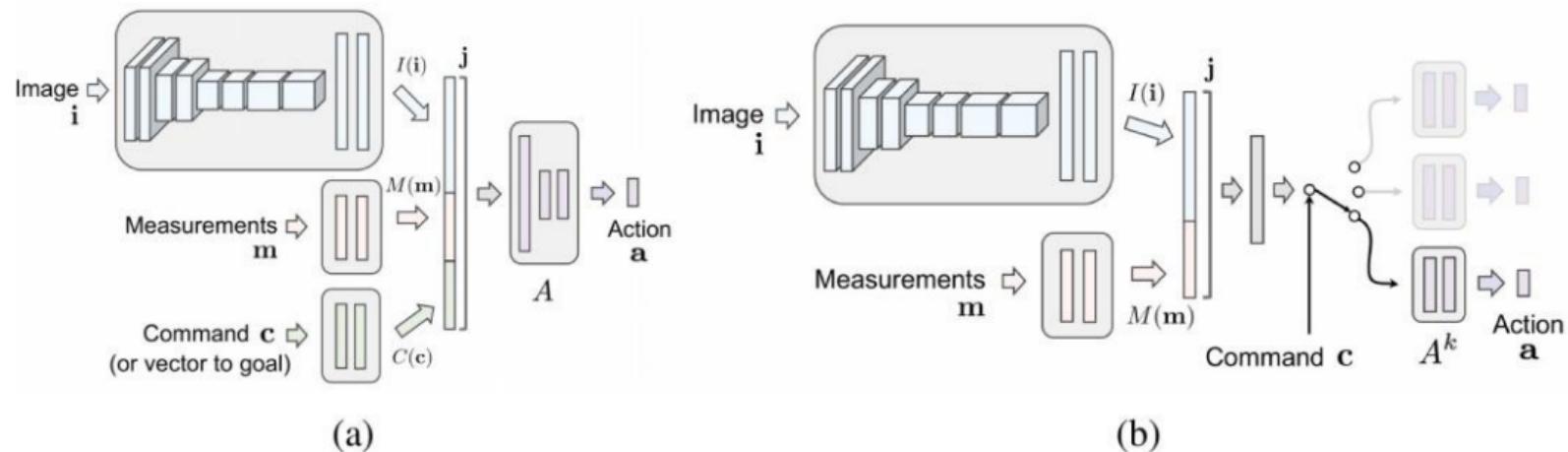
$$\operatorname{argmin}_{\theta} \sum_{i=1}^N \mathcal{L}(a_i^*, \pi_{\theta}(s_i^*, c_i^*))$$

- ▶ Assumption:

$$\exists f(\cdot, \cdot) : a_i = f(s_i, c_i)$$

Better assumption!

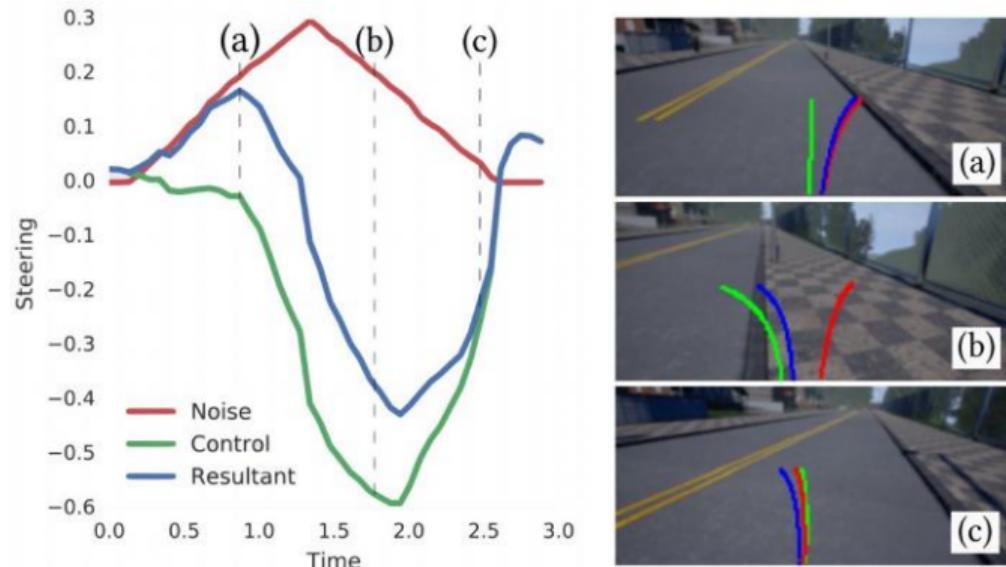
# Conditional Imitation Learning: Network Architecture



- ▶ This paper proposes two network architectures:
  - ▶ (a) Extract features  $C(c)$  and concatenate with image features  $I(i)$
  - ▶ (b) Command  $c$  acts as switch between specialized submodules
- ▶ Measurements  $m$  capture additional information (here: speed of vehicle)

Does CIL also suffer from catastrophic drift?

# Conditional Imitation Learning: Noise Injection



- ▶ Temporally correlated noise injected into trajectories  $\Rightarrow$  drift
- ▶ Record driver's (=expert's) corrective response  $\Rightarrow$  recover from drift

# CARLA Simulator



Town 1 (training)

Town 2 (testing)

<http://www.carla.org>

Codevilla, Muller, Lopez, Koltun and Dosovitskiy: End-to-End Driving Via Conditional Imitation Learning.  
ICRA, 2018.

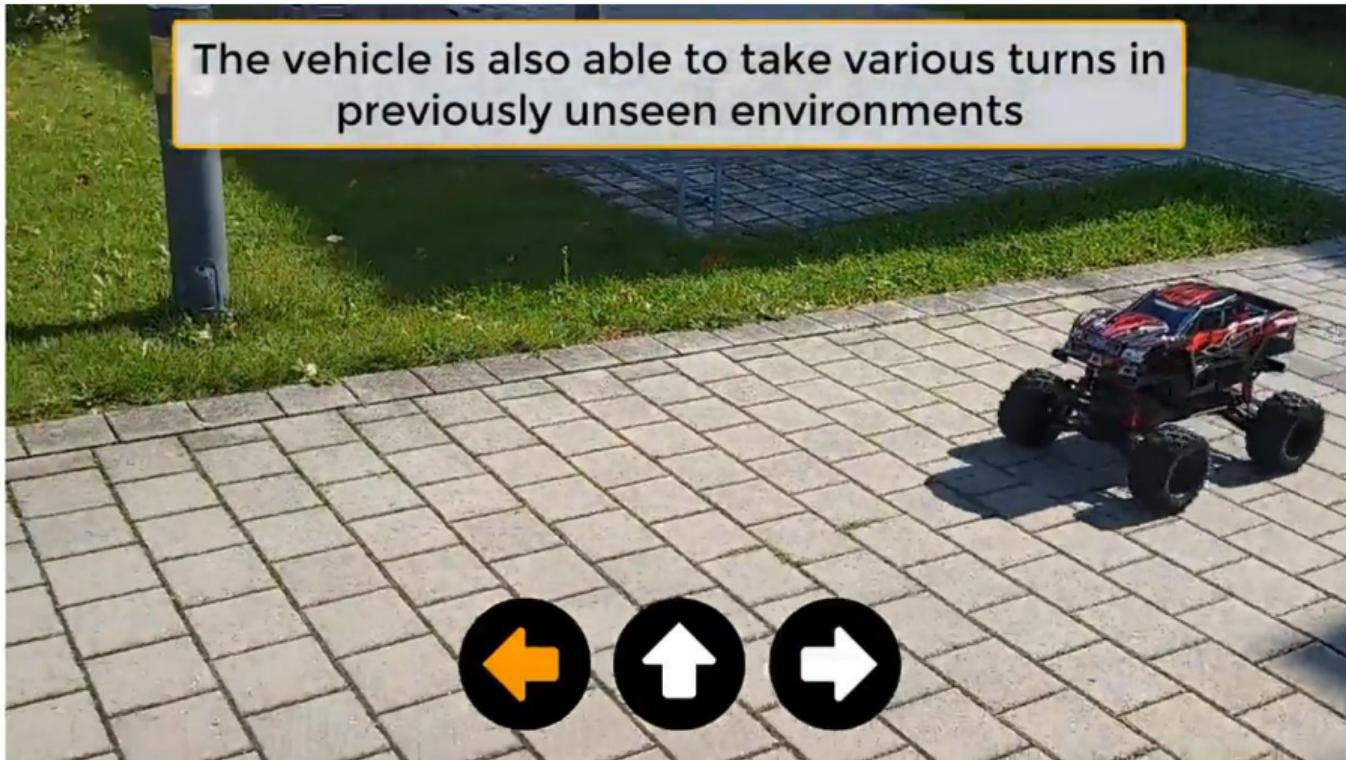
# Conditional Imitation Learning: Results

Model	Success rate		Km per infraction	
	Town 1	Town 2	Town 1	Town 2
Non-conditional	20%	26%	5.76	0.89
Goal-conditional	24%	30%	1.87	1.22
<b>Ours branched</b>	<b>88%</b>	<b>64%</b>	2.34	1.18
Ours cmd. input	78%	52%	3.97	1.30
Ours no noise	56%	22%	1.31	0.54
Ours no aug.	80%	0%	4.03	0.36
Ours shallow net	46%	14%	0.96	0.42

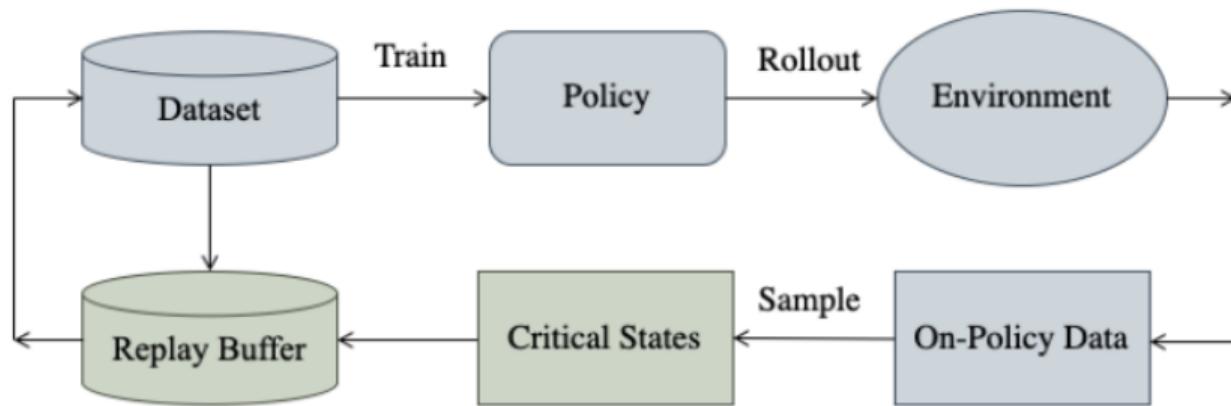
## Methods:

- ▶ Goal-conditional: provided with vector to goal (i.e., compass)
- ▶ Ours branched: Network architecture (b)
- ▶ Ours cmd. input: Network architecture (a)
- ▶ Ours no noise: no noise injected into trajectories (see previous slide)
- ▶ Ours no aug.: no augmentation (contrast, brightness, blur, noise, region dropout)

# Conditional Imitation Learning: Results



# Dagger with Critical States and Replay Buffer



## Key Ideas:

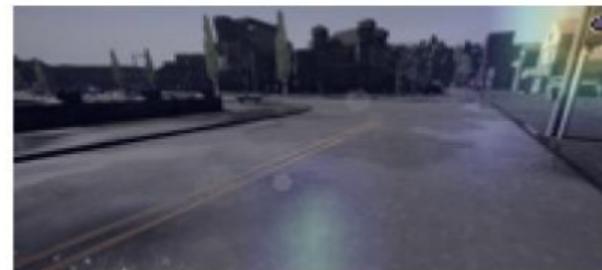
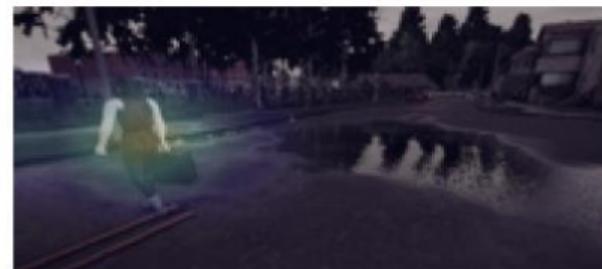
1. Sample **critical states** from the collected on-policy data based on the utility they provide to the learned policy in terms of driving behavior
2. Incorporate a **replay buffer** which progressively focuses on the high uncertainty regions of the policy's state distribution

# Interpretability: GradCAM Attention Maps

CILRS [Codevilla et al. 2019]



Our Approach



## Further Readings

- Ohn-Bar et. al., Learning Situational Driving, CVPR 2020
- Ross, Gordon and Bagnell: A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. AISTATS, 2011.
- Pomerleau: ALVINN: An Autonomous Land Vehicle in a Neural Network. NIPS, 1988.
- Bojarski et al.: End-to-End Learning for Self-Driving Cars. Arxiv, 2016.
- Levine et. al. :End-to-End Training of Deep Visuomotor Policies , JMLR, 2016
- Learning monocular reactive UAV control in cluttered natural environments, Ross et al. 2013