
HE VD
IG

Intelligence Artificielle pour les systèmes autonomes (IAA)

Introduction et principes des systèmes autonomes

Prof. Yann Thoma - Prof. Marina Zapater

Février 2024

Basé sur le cours du Prof. A. Geiger

Outline

Today's lesson

→ **The challenges of self-driving cars today**

→ A bit of history

→ Technical approaches to self-driving

→ Simulators

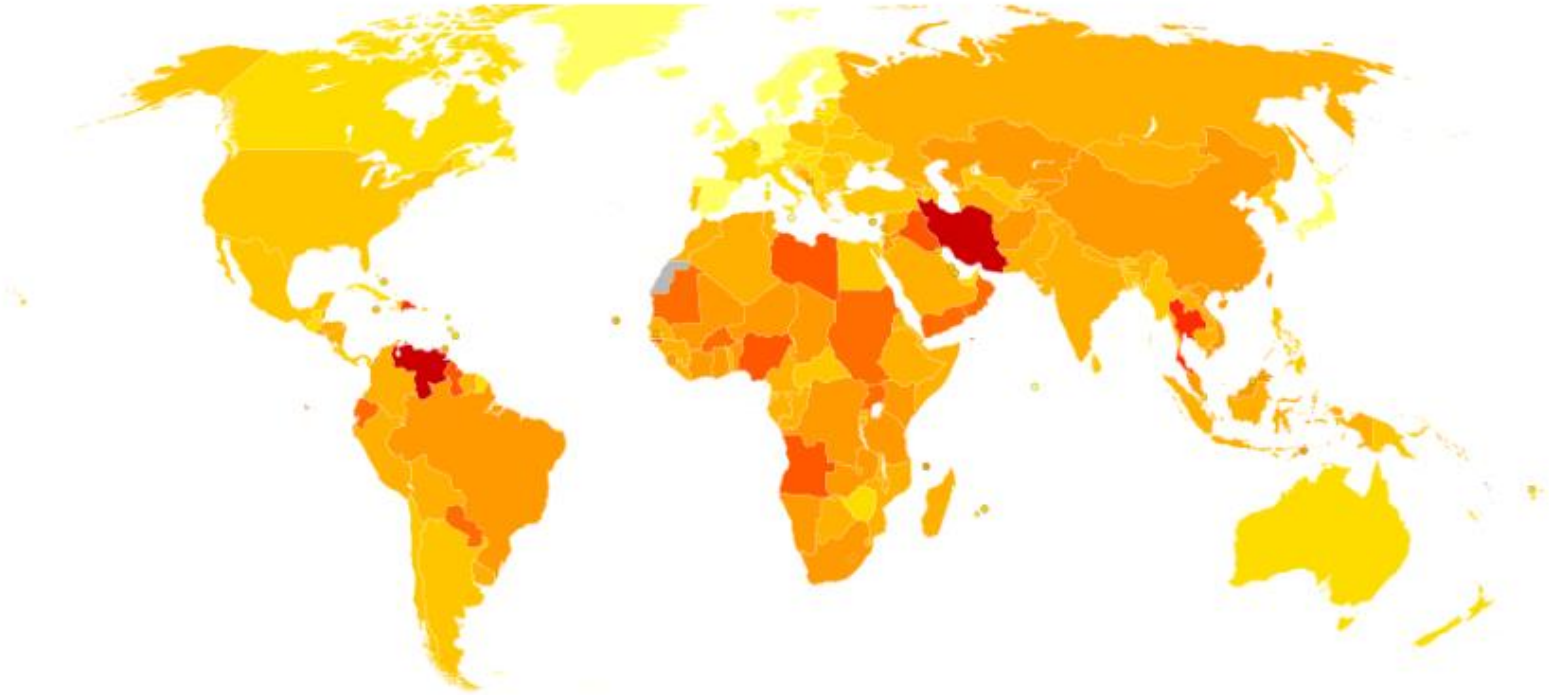
→ Deep Learning Recap

Why self-driving cars?

Road fatalities in 2017

→ USA : 32’700 Germany: 3’300 World: 1’300’000

→ Main factors: speed, intoxication, distraction



Uber Commercial (2018)

Uber self-driving car

https://www.youtube.com/watch?v=OKJK3_XIGD4

Uber Fatal Accident (2018)

First pedestrian killed by a Uber autonomous car

https://youtu.be/JWXqn4rg2WQ

Uber Fatal Accident (2018)

Exterior and interior view

https://www.youtube.com/watch?v=2O5Xu-YeBuc

Self-driving performance

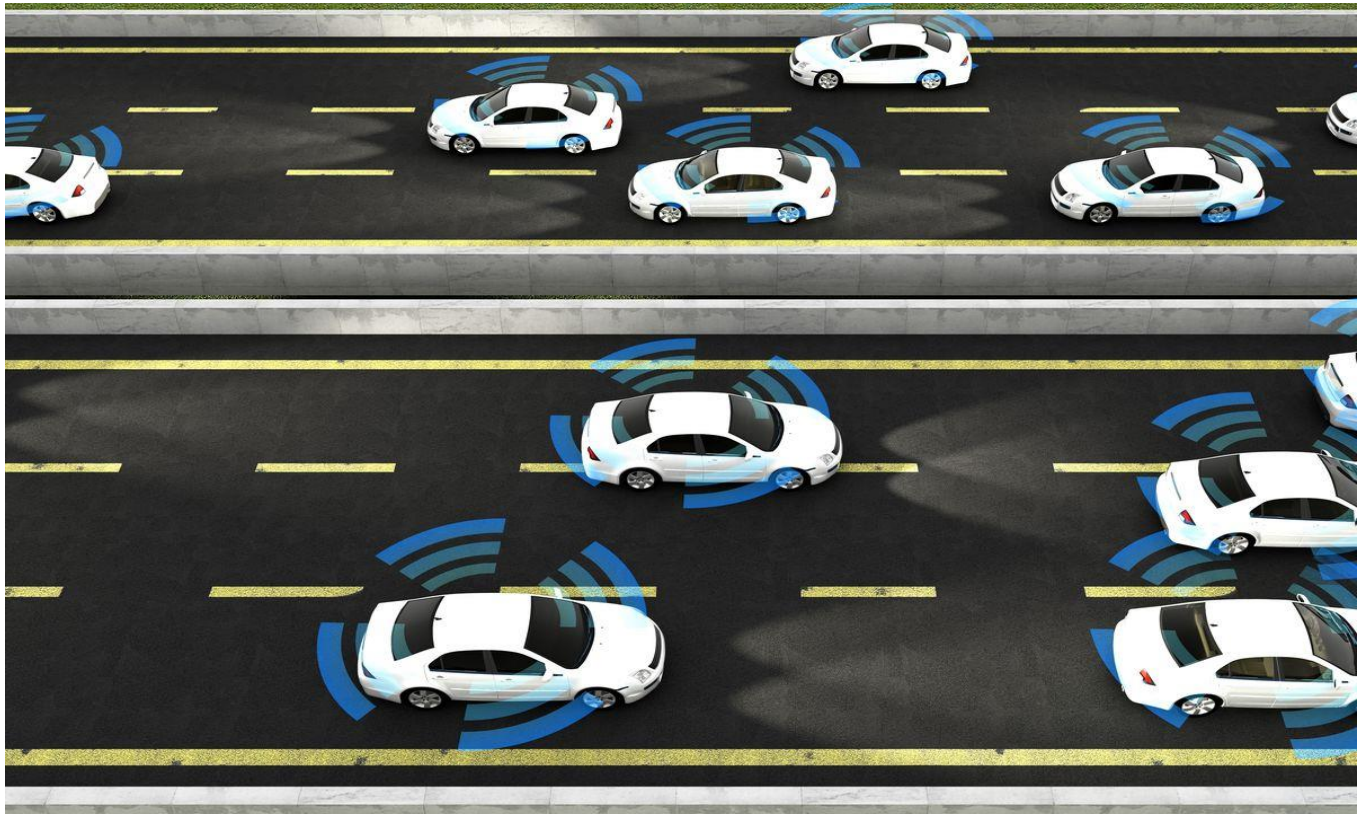
→ Human performance: 1 fatality per 100 million miles
- Error rate: 0.000001%

→ Neural nets… at around 0.1%?

→ Challenges:
- Snow, heavy rain, night
- Unstructured roads, traffic, parking lots
- Erratic behavior, pedestrians, etc.
- Unseen events
- Ethics
- Legal questions

A perfect world (well…)

Fully autonomous cars



Unstructured traffic

Paris, Arc de Triomphe

https://www.youtube.com/watch?v=FXfGZF2-sUU

The trolley problem (1905)

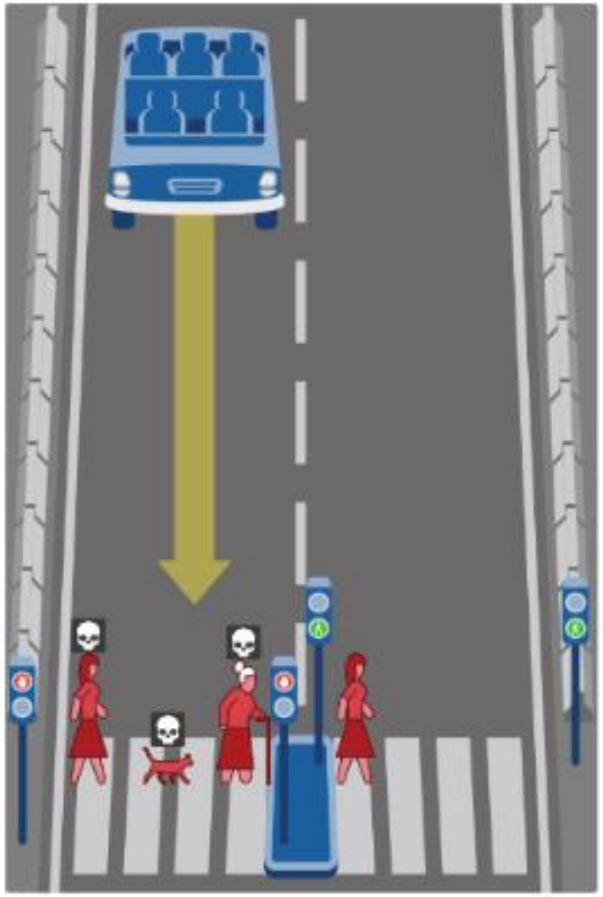
Thought experiment

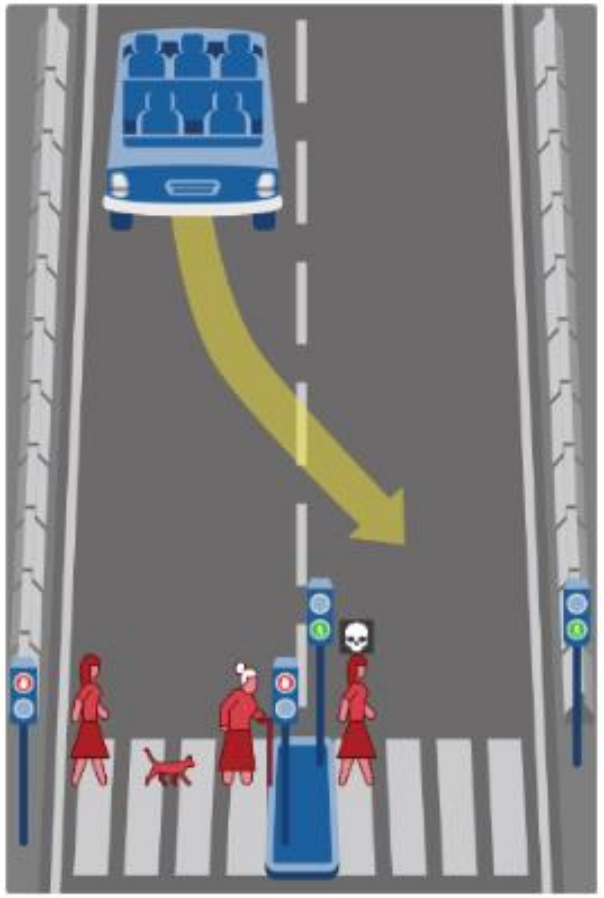
→ You observe a train that will kill 5 people on the rail tracks if it continues
→ You have the option to pull a lever to redirect the train to another track
→ However, the train will kill one (other) person on that alternate track
→ What is your decision? What is the correct/ethical decision?

→ Bonus: What if the five are old, and the one is 4-year old
→ Bonus: What if the five are bad persons, and the one an angel?

The MIT moral machine

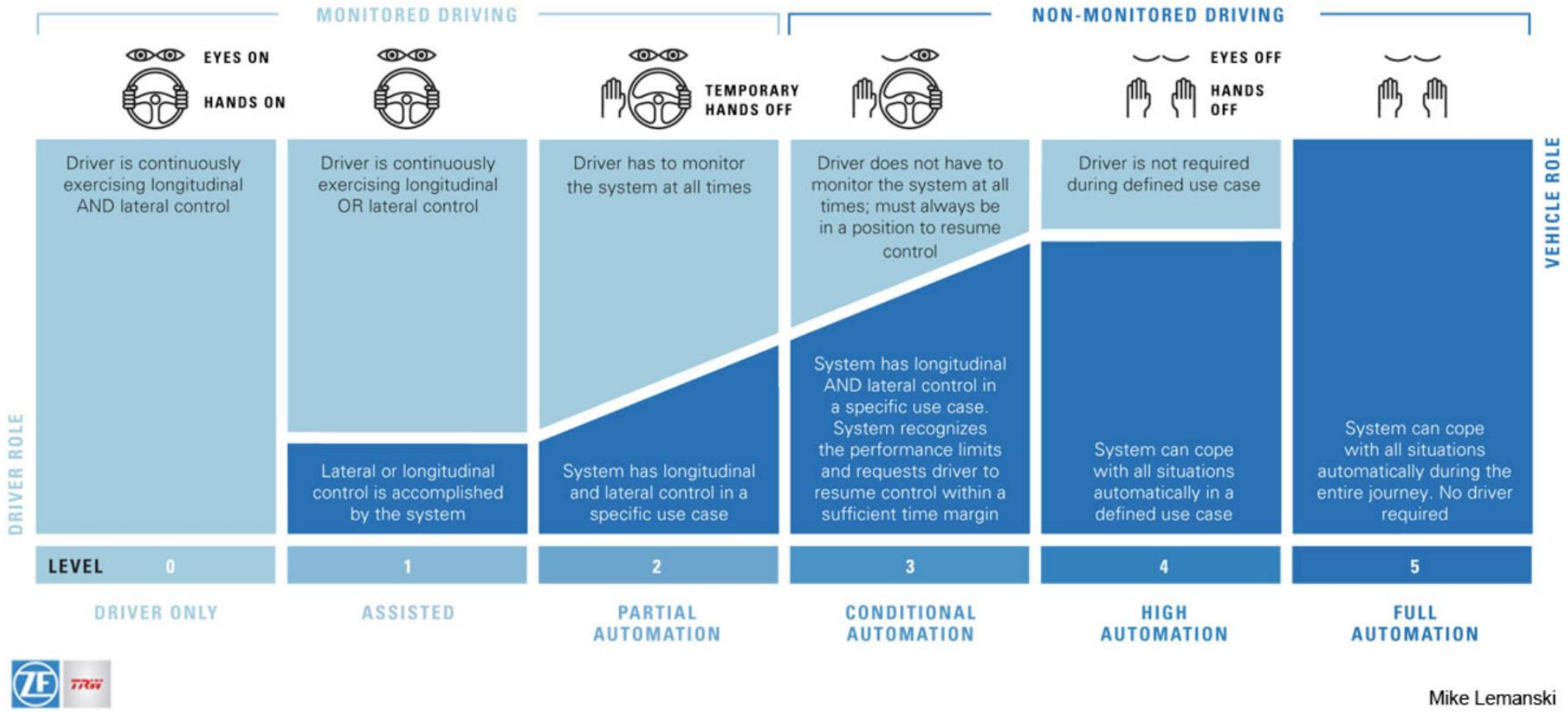
http://moralmachine.mit.edu/





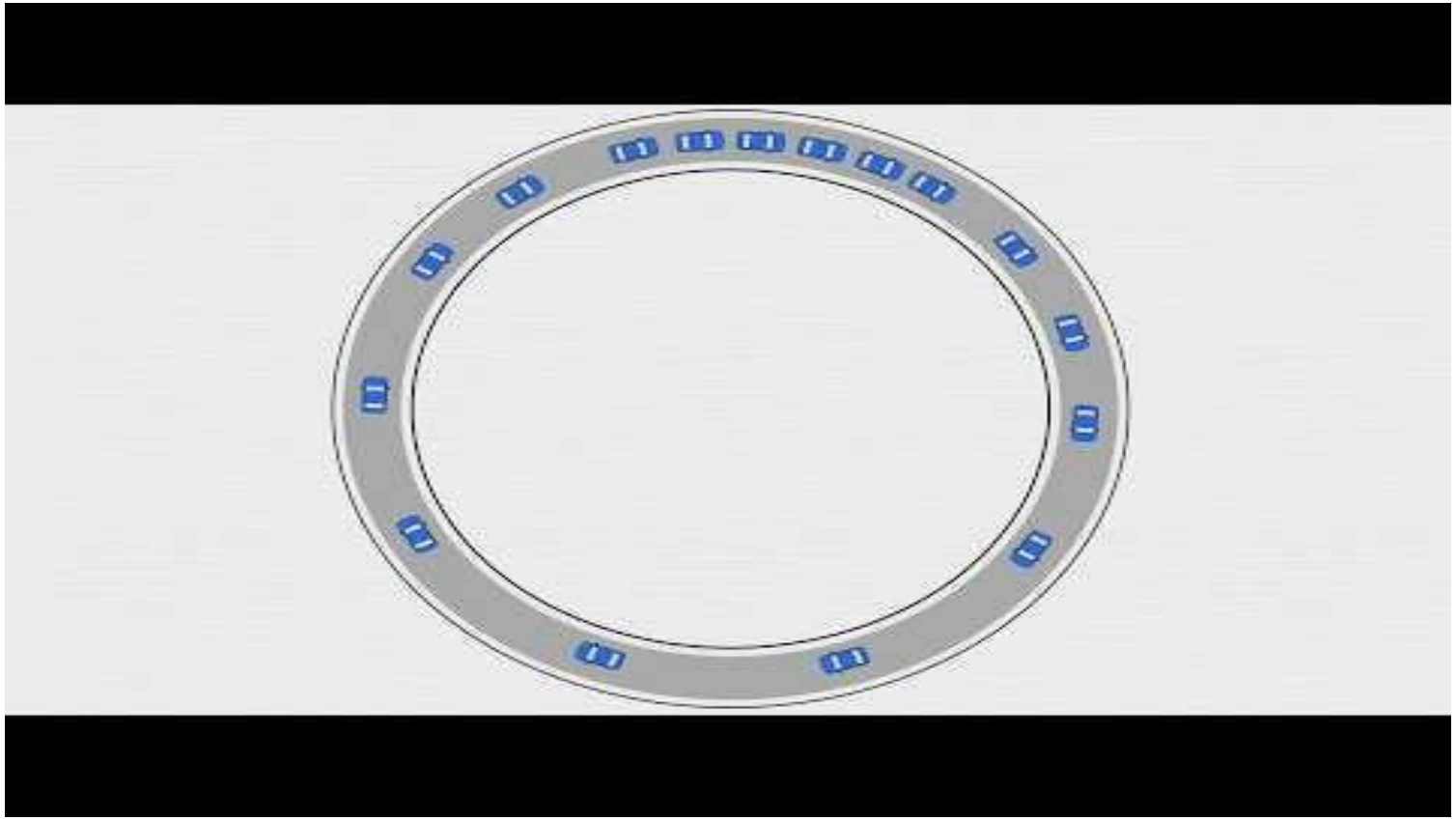
Levels of Autonomy in Self-Driving cars

Society of Automotive Engineers (2014)



Mike Lemanski

About efficiency: autonomous-driven vs human drivers



https://youtu.be/iHzzSao6ypE

Outline

Today's lesson

→ The challenges of self-driving cars today

→ **A bit of history**

→ Technical approaches to self-driving

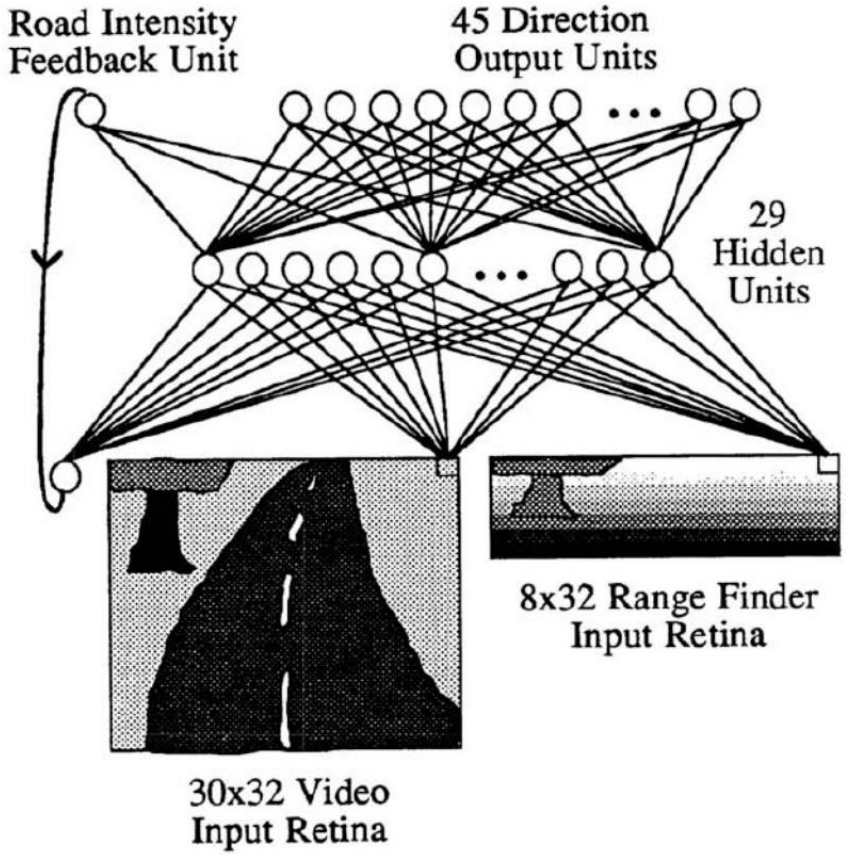
→ Simulators

→ Deep Learning Recap

1988: ALVINN

ALVINN: An autonomous Land Vehicle in a Neural Network

- → Forward-looking, vision based driving
- → Fully connected neural network maps road images to vehicle turn radius
- → Directions discretized (45 bins)
- → Trained on simulated road images
- → Tested on unlined paths, lined city streets and interstate highways
- → 90 consecutive miles at up to 70mph



ALVINN's Neural Net

What's Hidden in the Hidden Layers?

The contents can be easy to find with a geometrical problem, but the hidden layers have yet to give up all their secrets

David S. Touretzky and Dean A. Pomerleau

tions, we fed the network road images taken under a wide variety of viewing angles and lighting conditions. It would be impractical to try to collect thousands of real road images for such a data set. Instead, we developed a synthetic road-image generator that can create as many training examples as we need.

To train the network, 1200 simulated road images are presented 40 times each, while the weights are adjusted using the back-propagation learning algorithm. This takes about 30 minutes on Carnegie Mellon's Warp systolic-array supercomputer. (This machine was designed at Carnegie Mellon and is built by General Electric. It has a peak rate of 100 million floating-point operations per second and can compute weight adjustments for back-propagation networks at a rate of 20 million connections per second.)

Once it is trained, ALVINN can accurately drive the NAVLAB vehicle at about 3½ miles per hour along a path through a wooded area adjoining the Carnegie Mellon campus, under a variety of weather and lighting conditions. This speed is nearly twice as fast as that achieved by non-neural-network algorithms running on the same vehicle. Part of the reason for this is that the forward pass of a back-propagation network can be computed quickly. It takes about 200 milliseconds on the Sun-3/160 workstation installed on the NAVLAB.

The hidden-layer representations ALVINN develops are interesting. When trained on roads of a fixed width, the network chooses a representation in which hidden units act as detectors for complete roads at various positions and orientations. When trained on roads of variable

continued

Photo 1: *The NAVLAB autonomous navigation test-bed vehicle and the road used for trial runs.*

ALVINN

By Carnegie Mellon University

https://youtu.be/2KMAAmkz9go

1995: Invention of Adaptive Cruise Control (ACC)

→ 1992: Lidar-based distance control by Mitsubishi (distance warning)

→ 1997: Laser adaptive cruise control by Toyota (throttle control & downshift)

→ 1999: Distronic radar-assisted ACC by Mercedes-Benz (S-Class), level 1 autonomy

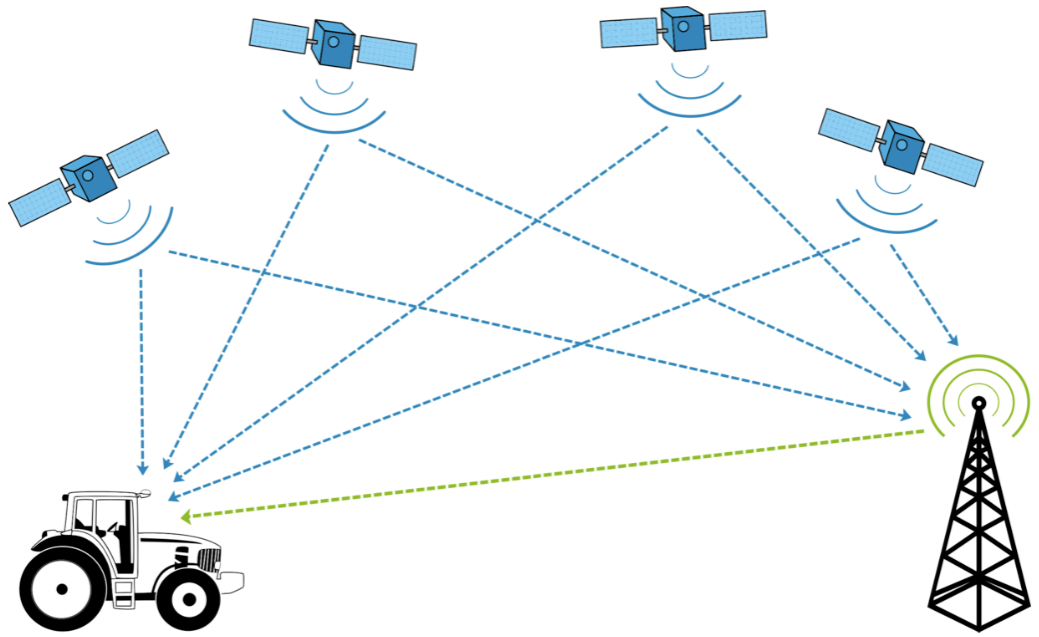
2000: GPS & Maps

“First technological revolution”

→ NAVSTAR GPS available with 1 meter accuracy, IMUs improve up to 5 cm

→ Navigation systems and road maps available

→ Accurate self-localization and ego-motion estimation algorithms



GPS: Global Positioning System
IMU: Inertial Measurement Unit

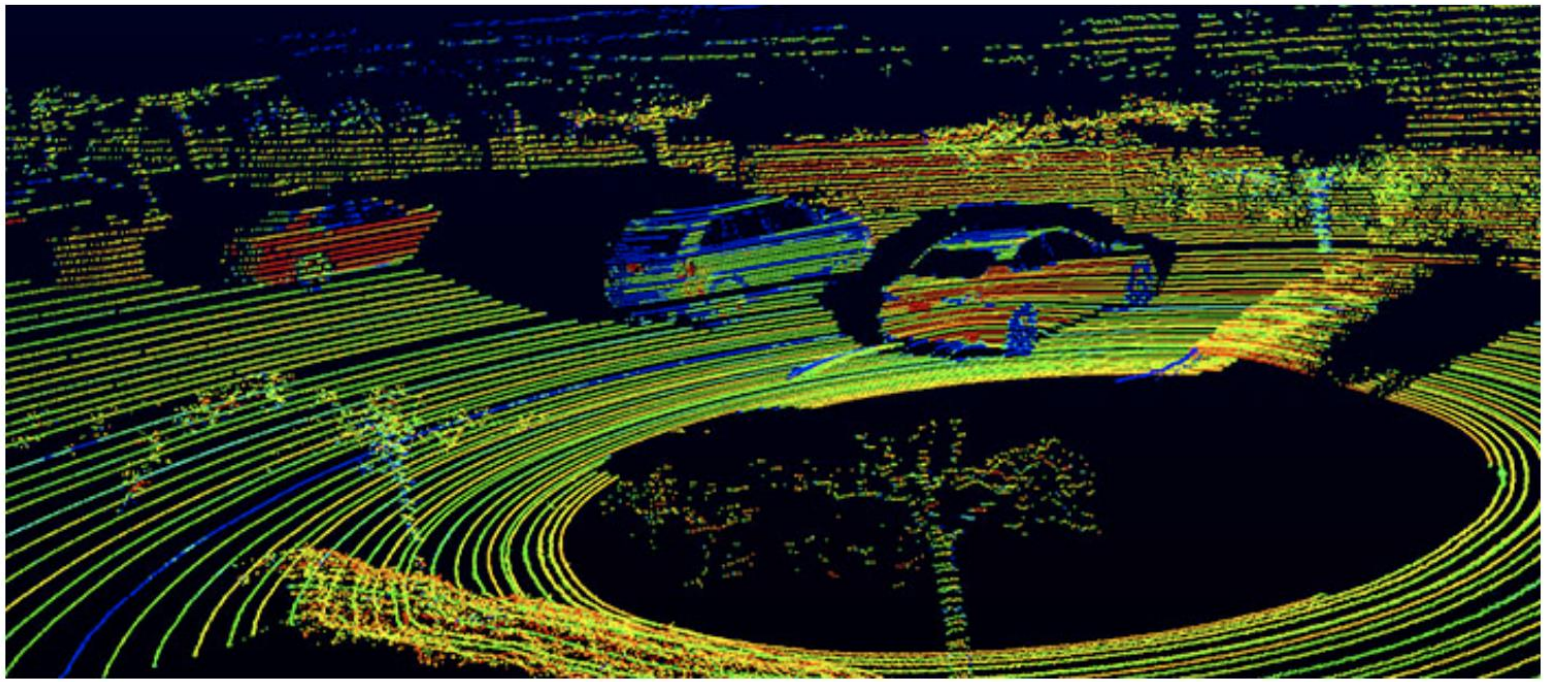
2006: Lidars and high-resolution sensors

“Second technological revolution”

→ High-resolution Lidar (Light Detection and Ranging)

→ Camera systems with increasing resolution

→ Accurate 3D reconstruction, 3D detection & 3D localization



2012: Deep Learning and new benchmarks

“Third technological revolution”

2014: Mercedes S class

First Level 2 Autonomy

→ Autonomous steering, lane keeping, acceleration/braking, collision avoidance

→ Driver fatigue monitoring in city traffic and highway speeds up to 200 km/h

2015: Tesla Model S Autopilot

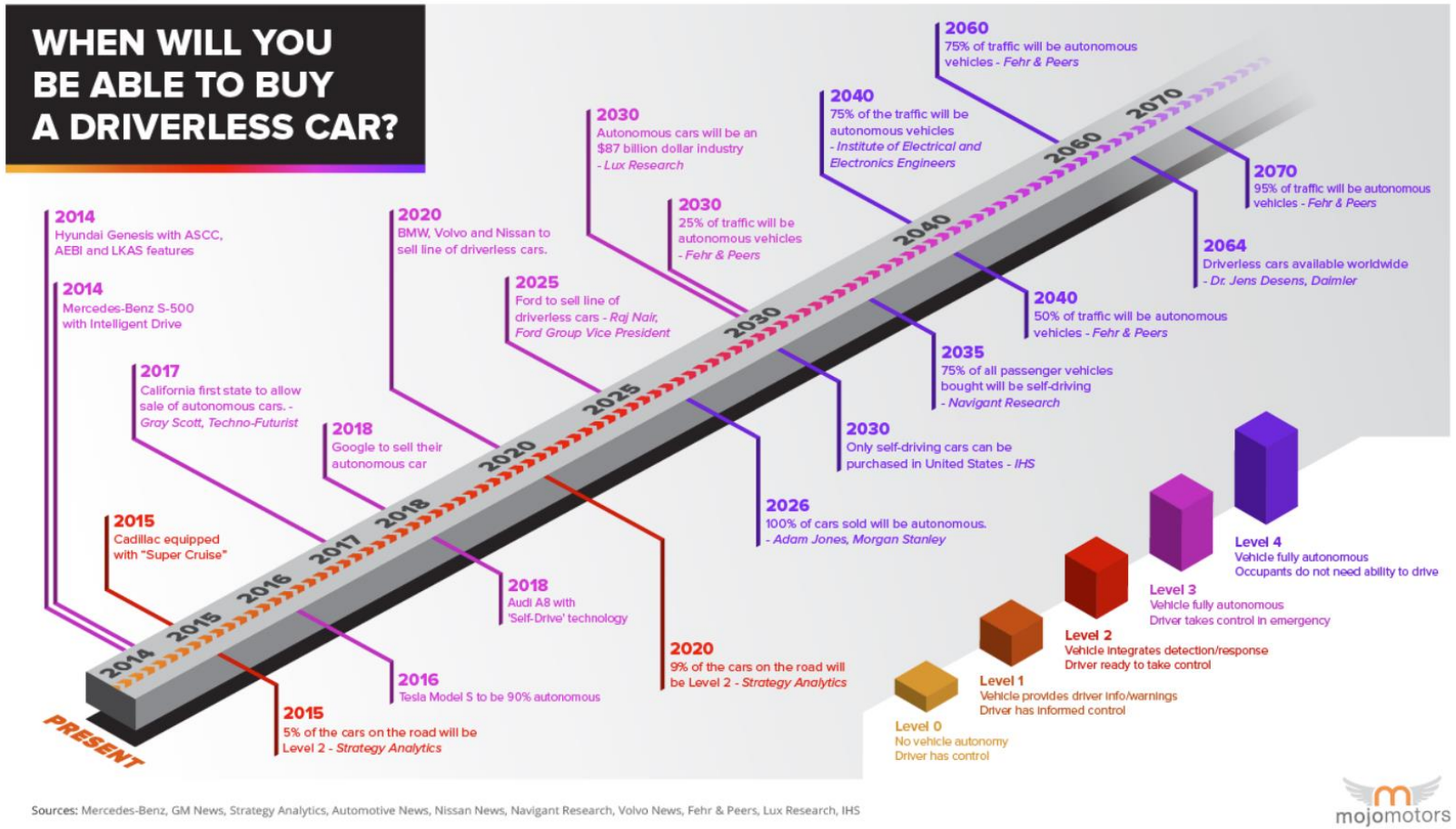
Level 2 Autonomy

→ Lane keeping for limited-access highways (hands off time: 30-120 seconds)

→ Doesn’t read traffic signals, traffic signs or detect pedestrians/cyclists

Wild predictions about the future of self-driving

Prediction from 2014



In any case…

Two business models

→ The risky one: Autonomous or nothing (Google, Apple, Uber)

- Very risky, only few companies
- Long-term goals

→ Introduce technology gradually (all car companies)

- Car industry is very conservative
- Advanced Driver Assistance System (ADAS) as an intermediate goal
  - Alerts and warnings
  - Crash mitigation
  - Driving task assistance
  - Visual and environmental monitoring
- How to maintain driver engaged?

Outline

Today's lesson

→ The challenges of self-driving cars today

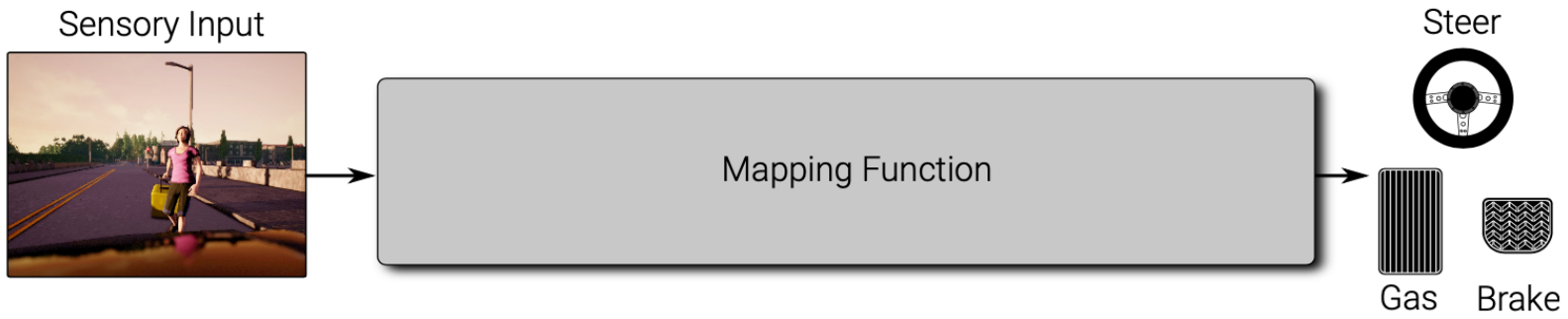
→ A bit of history

→ **Technical approaches to self-driving**

→ Simulators

→ Deep Learning Recap

Technical approaches to self-driving



→ Modular pipelines

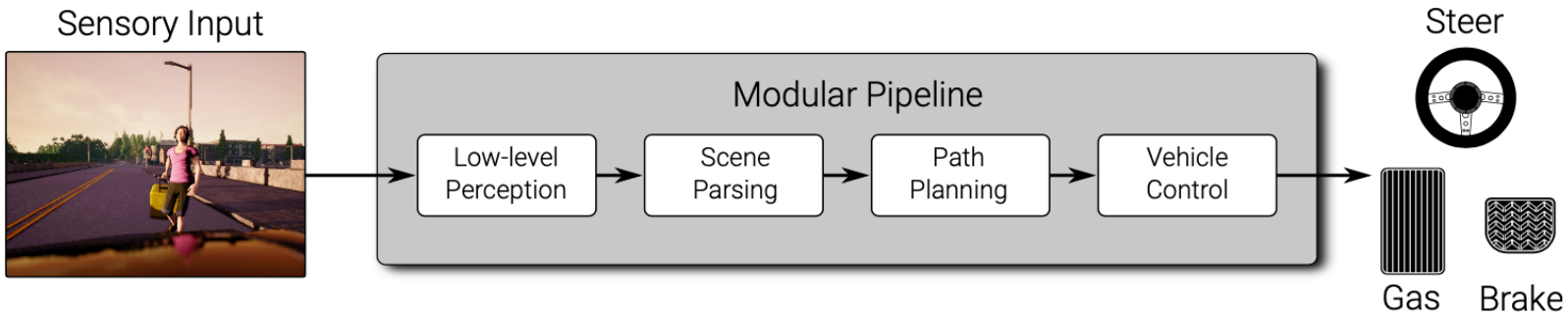
→ End-to-end learning (imitation learning, reinforcement learning)

→ Direct perception

To be studied in-depth during this course

Autonomous driving: The modular pipeline

(First approach → and the one finally used)



→ Small components, easy to develop in parallel

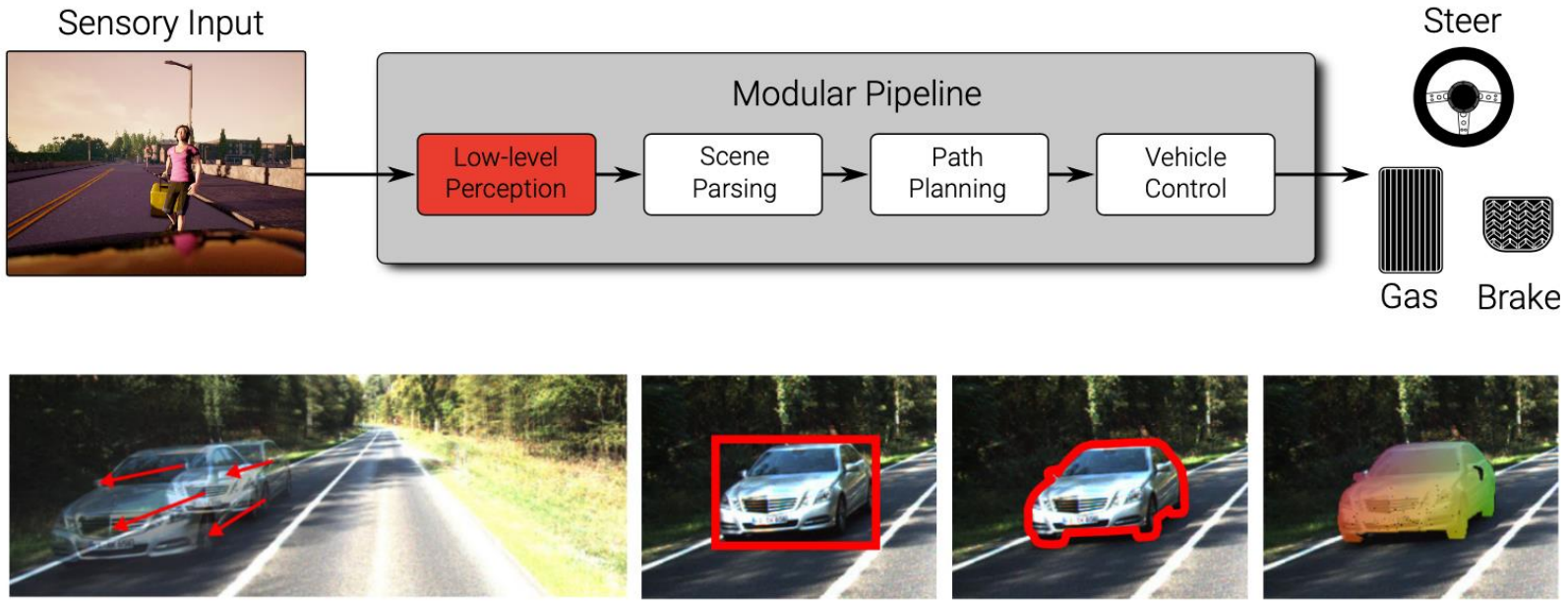
→ Interpretability

→ Piece-wise training (not jointly)

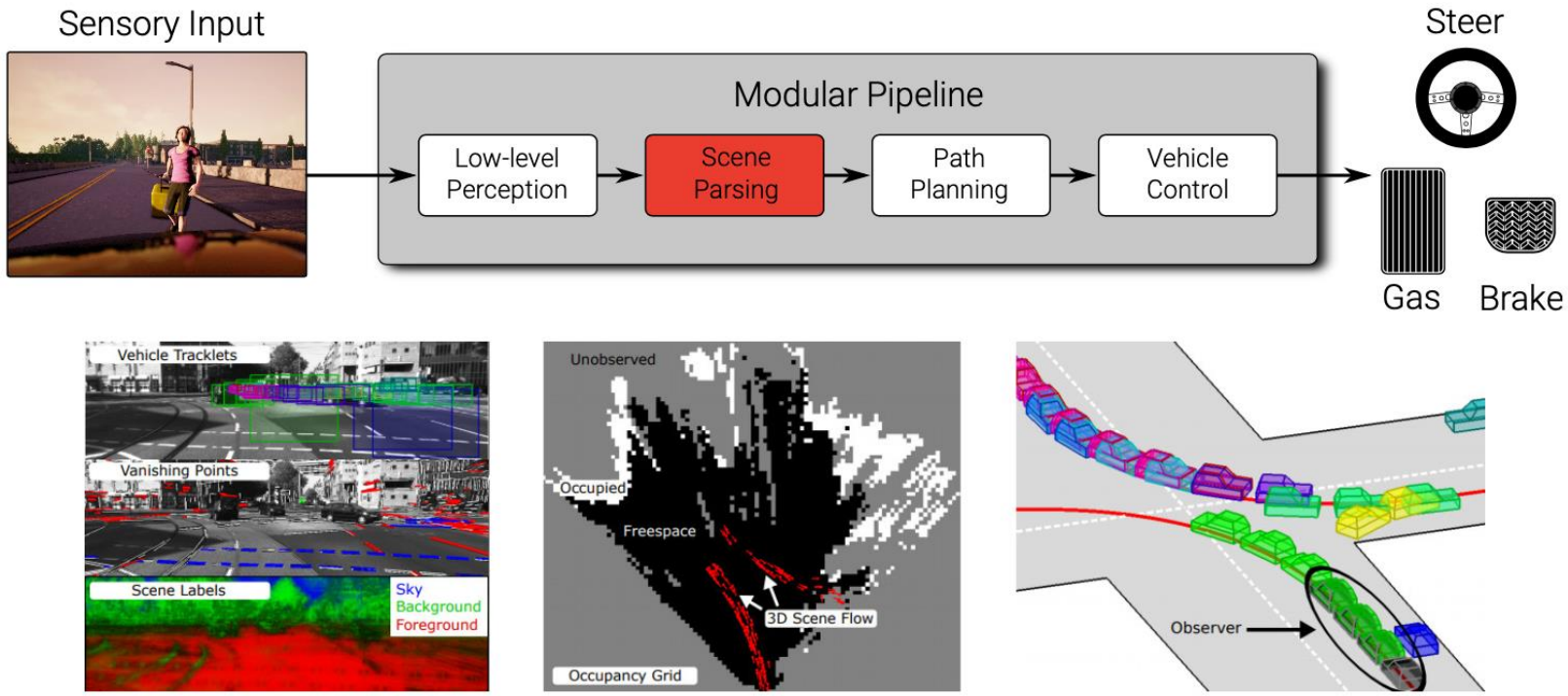
→ Path-planning (and localisation) relies on HD maps

- HD maps: Centimeter precision lanes, markings, traffic lights/signs, human annotated

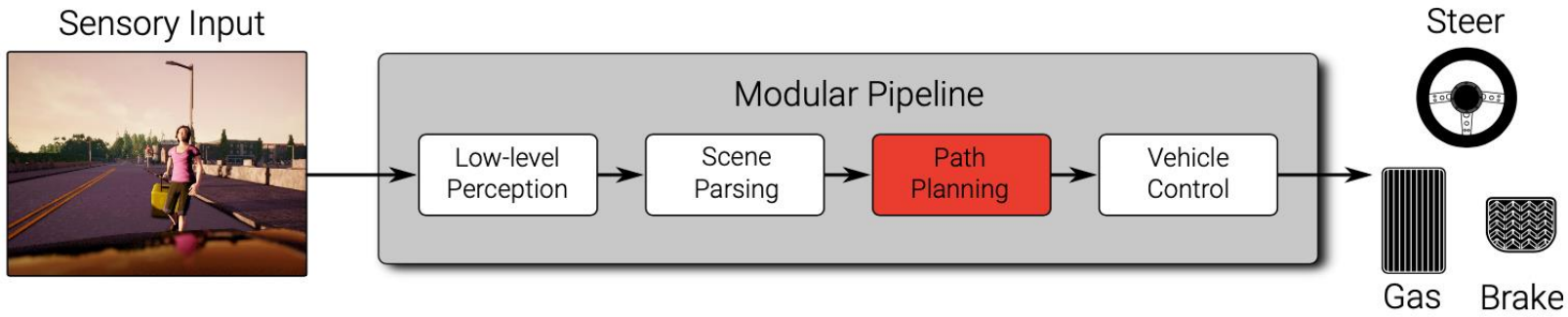
Autonomous driving: The modular pipeline

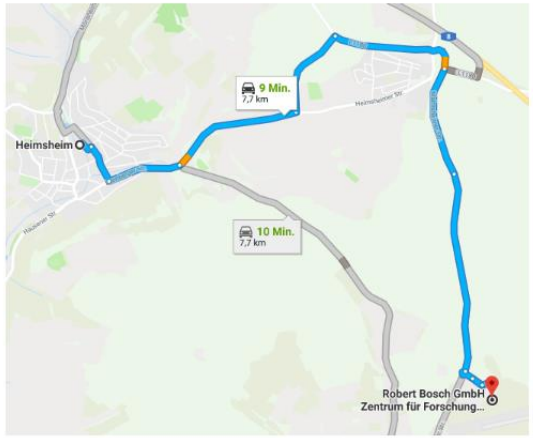


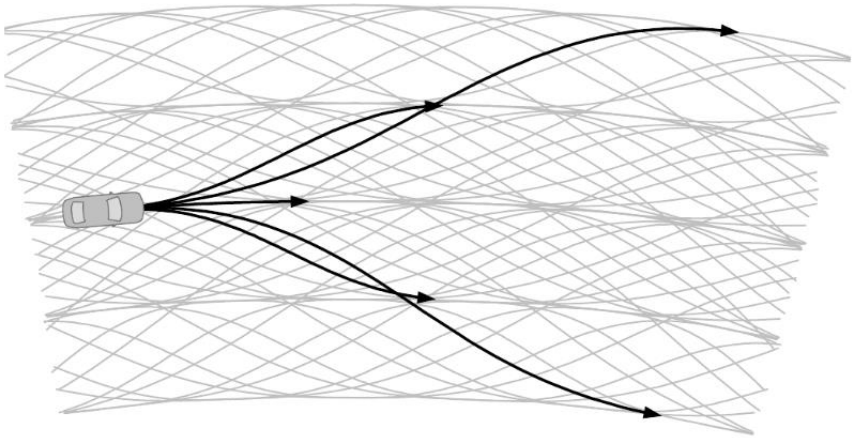
Autonomous driving: The modular pipeline



Autonomous driving: The modular pipeline

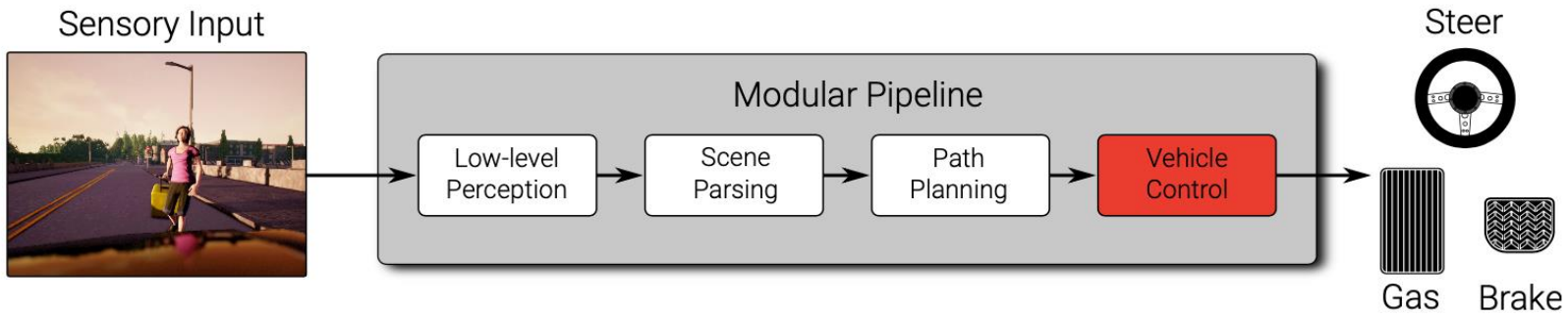


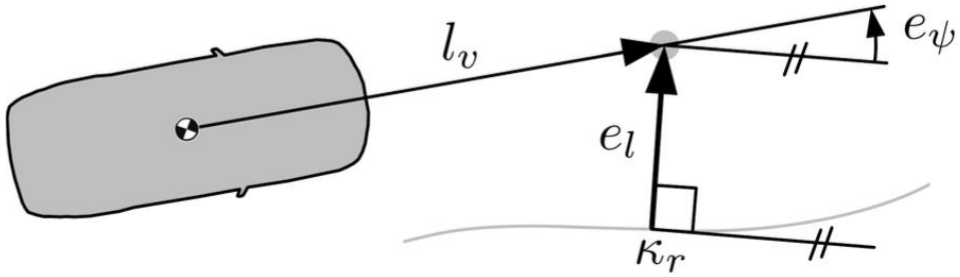




→ https://www.geospatialworld.net/article/hd-maps-autonomous-vehicles/

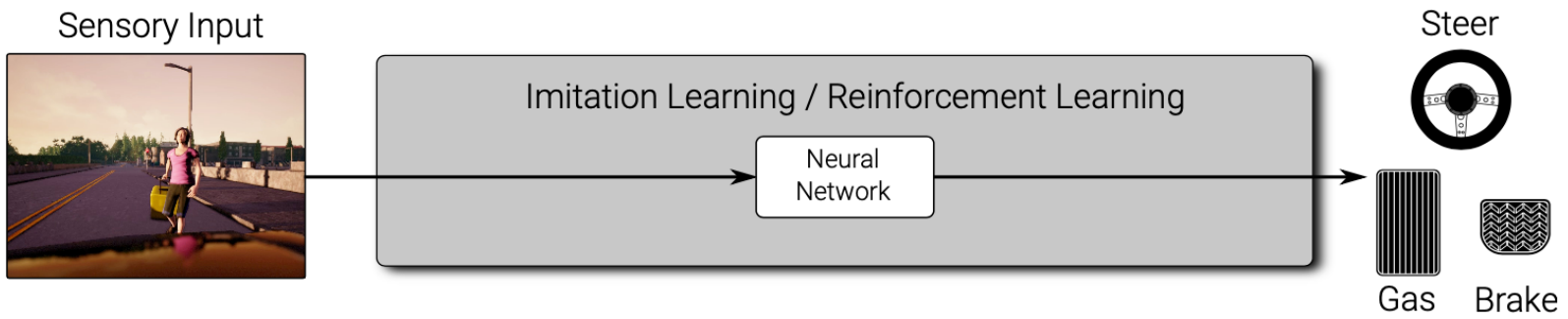
Autonomous driving: The modular pipeline





Autonomous driving: End-to-end learning

(Second approach)



→ End-to-end requires cheap(er) annotations

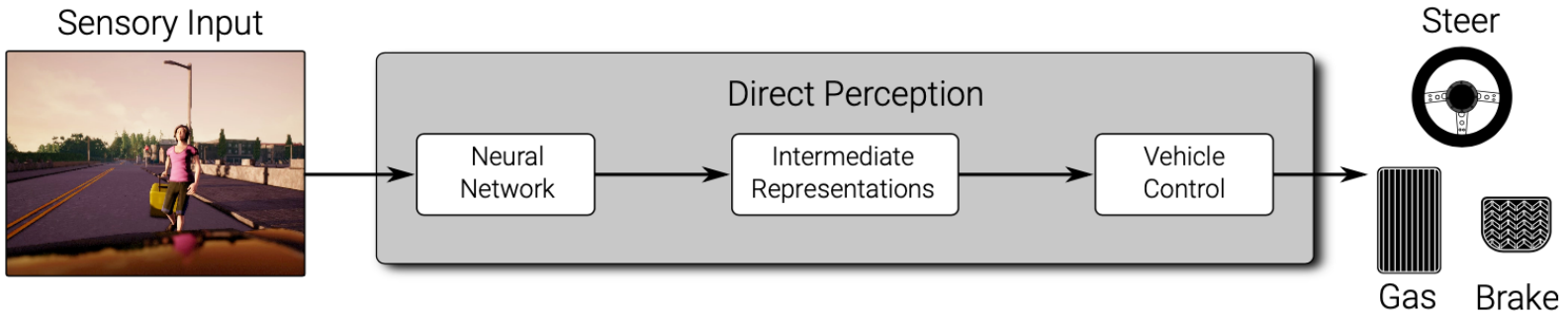
→ Issues with training/generalization

→ Lack of interpretability…

→ Certification issues

Autonomous driving: Direct Perception

(Third approach)



→ Middle ground between previous two approaches

→ Compact representation and interpretability

→ Control is not learnt jointly

→ Representations can be difficult to choose

Outline

Today's lesson

→ The challenges of self-driving cars today

→ A bit of history

→ Technical approaches to self-driving

→ **Simulators**

→ Deep Learning Recap

How do we create and train/test self-driving models?

Simulators

→ Toy simulators for our course (and beyond)

- Gymnasium : https://gymnasium.farama.org/environments/box2d/car_racing/
- TORCS : The Open Racing Car Simulator

→ Carla Simulator

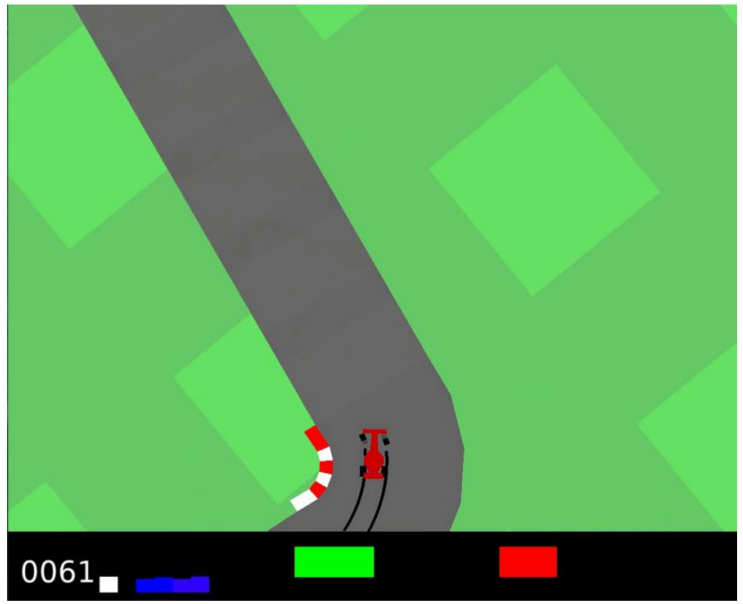
How do we create and train/test self-driving models?

Simulators

→ Gymnasium by OpenAI

- https://gymnasium.farama.org/environments/box2d/car_racing/
- https://towardsdatascience.com/getting-started-with-openai-gym-d2ac911f5cbc

Gymnasium



Car Racing

This environment is part of the Box2D environments.Please read that page first for general information.

Action Space	Box([-1. 0. 0.], 1.0, (3,), float32)
Observation Shape	(96, 96, 3)
Observation High	255
Observation Low	0
Import	`gymnasium.make("CarRacing-v2")`

Description

The easiest control task to learn from pixels - a top-down racing environment. The generated track is random every episode.

Some indicators are shown at the bottom of the window along with the state RGB buffer. From left to right: true speed, four ABS sensors, steering wheel position, and gyroscope. To play yourself (it's rather fast for humans), type:

```
python gymnasium/envs/box2d/car_racing.py
```

Gymnasium by OpenAI

Car Racing

https://www.youtube.com/watch?v=Ev0wpVB7OEs

How do we create and train/test self-driving models?

Simulators

→ TORCS : The Open Racing Car Simulator

- https://sourceforge.net/projects/torcs/

https://www.youtube.com/watch?v=8rlExkFYvNE

How do we create and train/test self-driving models?

Simulators

→ CARLA Simulator

https://www.youtube.com/watch?v=AZhzZy57XeU

How does Carla work?

Fundamentals

→ https://www.youtube.com/watch?v=pONr1R1dy88

→ C++/Python interaction with Carla

Outline

Today's lesson

→ The challenges of self-driving cars today

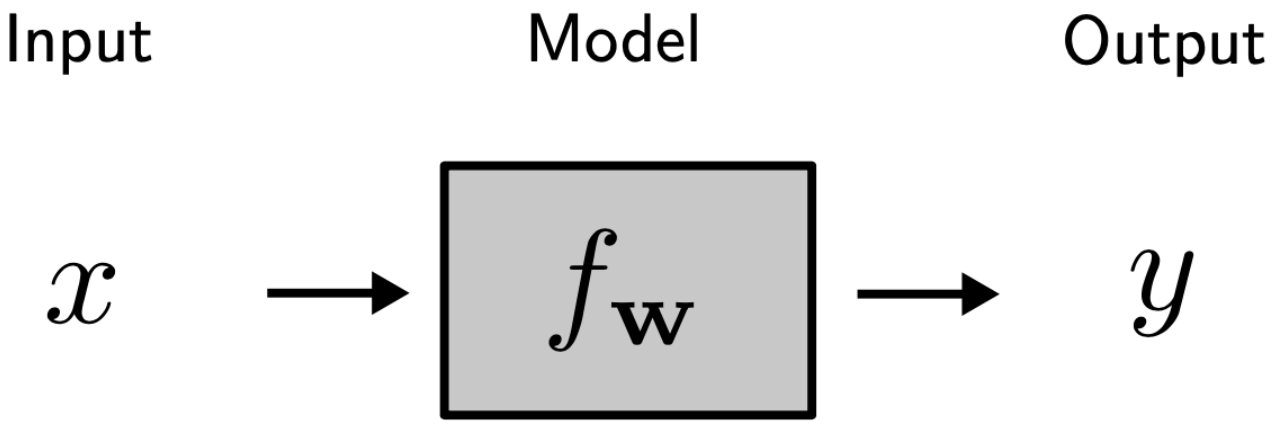
→ A bit of history

→ Technical approaches to self-driving

→ Simulators

→ **Deep Learning Recap**

Supervised Learning



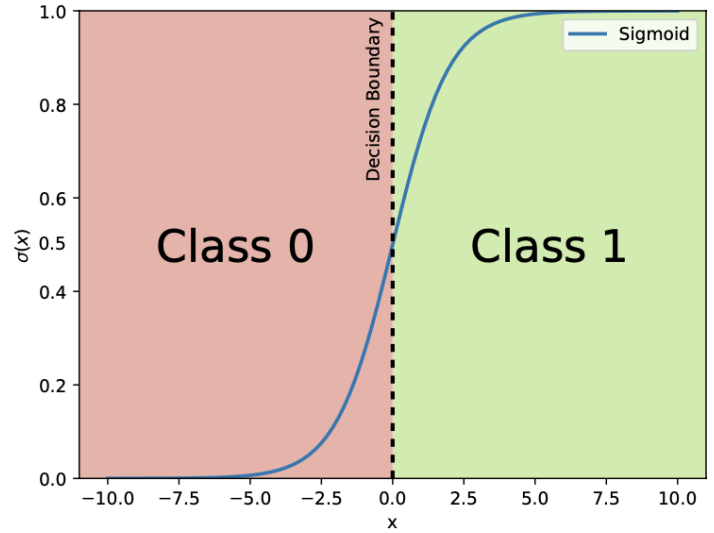
- **Learning:** Estimate parameters $\mathbf{w}$ from training data $\{(x_i, y_i)\}_{i=1}^{N}$
- **Inference:** Make novel predictions: $y = f_{\mathbf{w}}(x)$

Logistic regression

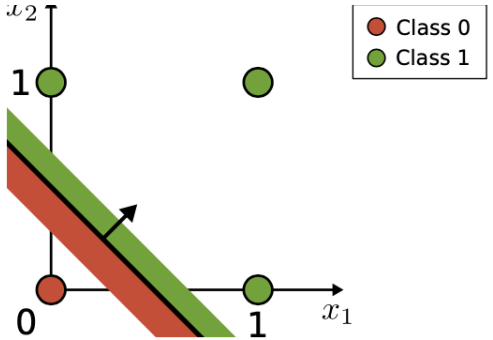
Applied to linear classification

$$\hat{y} = \sigma(\mathbf{w}^\top \mathbf{x} + w_0) \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$

- Let $\mathbf{x} \in \mathbb{R}^2$
- Decision boundary: $\mathbf{w}^\top \mathbf{x} + w_0 = 0$
- Decide for class $1 \Leftrightarrow \mathbf{w}^\top \mathbf{x} > -w_0$
- Decide for class $0 \Leftrightarrow \mathbf{w}^\top \mathbf{x} < -w_0$
- Which problems can we solve?



$$\underbrace{\begin{pmatrix} 1 & 1 \end{pmatrix}}_{\mathbf{w}^\top} \underbrace{\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}}_{\mathbf{x}} > \underbrace{0.5}_{-w_0}$$



Non-linear classification

Multi-layer perceptrons

→ **MLPs** are feedforward networks (no feedback connections)

→ They compose several non-linear functions: $f(x) = \hat{y}\left(h_3\left(h_2\left(h_1(x)\right)\right)\right)$ where $\mathrm{h}_i(.)$ are called **hidden layers** and $\hat{y}(.)$ is **the output layer**

→ The data specifies only the behavior of the output layers (not the hidden)

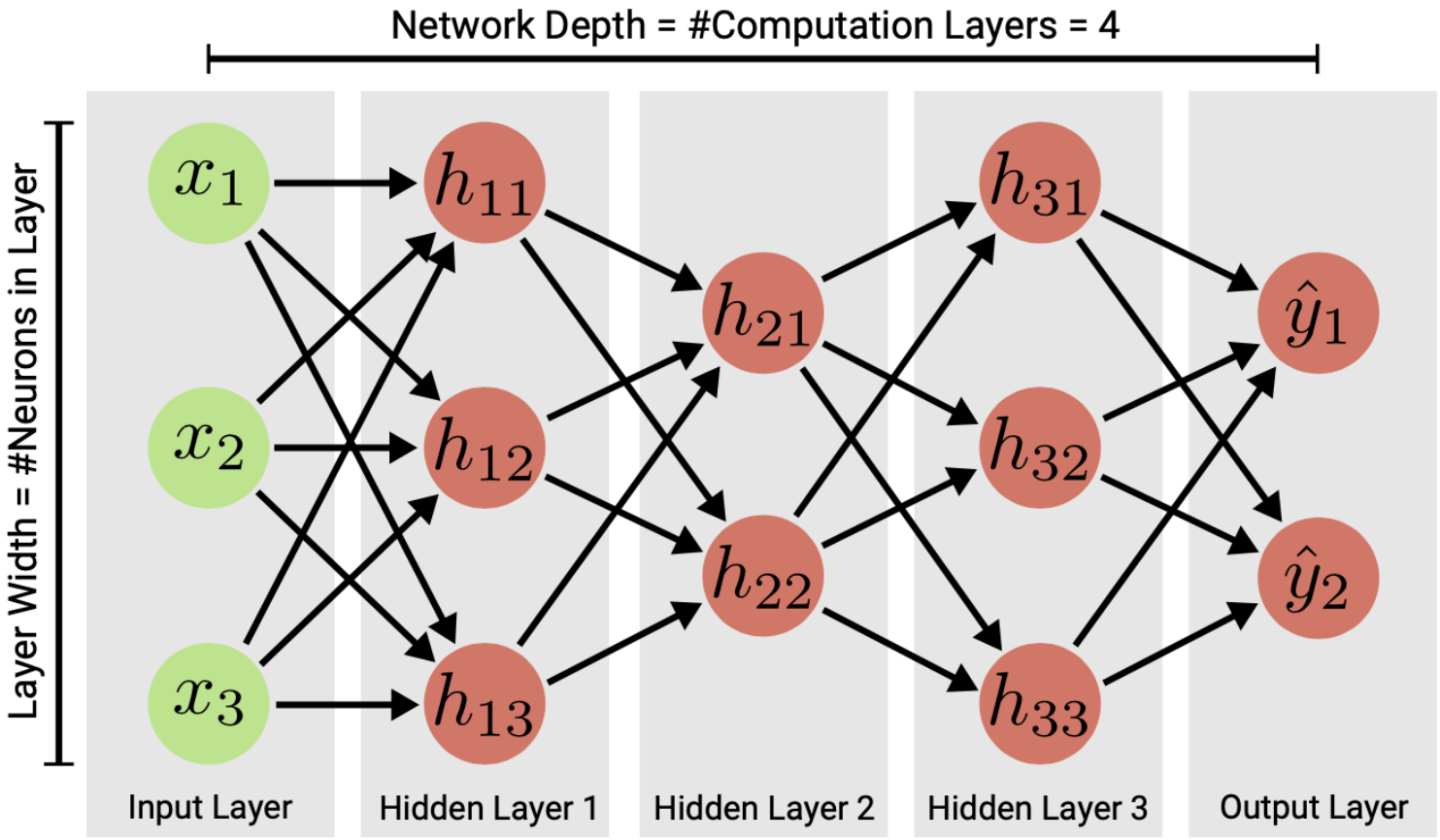
→ Each layer i comprises multiple neurons j which are implemented as **affine transformations** $(a^T x + b)$ followed by a non-linear activation (g):

$$h_{ij} = g(a_{ij}{}^T h_{i\_1\_} bi_j)$$

→ Each neuron is **fully connected** to all neurons of the previous layer

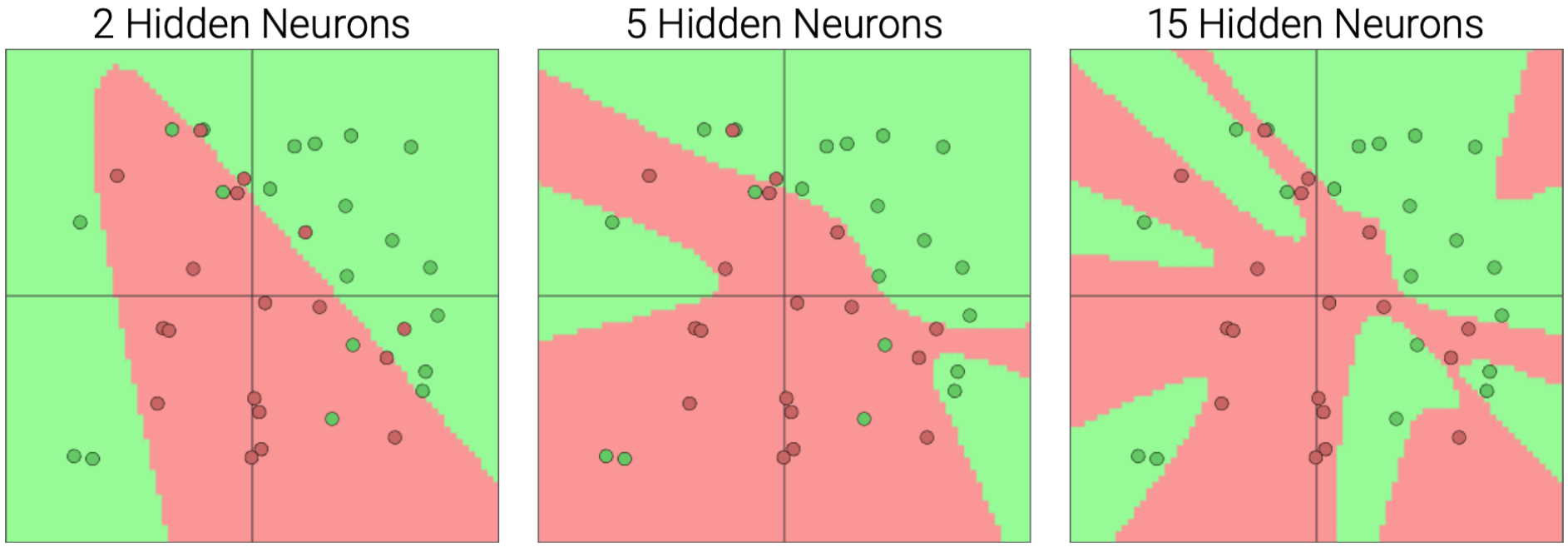
→ The overall length of the chain is the **depth** of the model (“Deep Learning”)

MLP network architecture



Deeper Models allow for more complex classification

Give it a try!



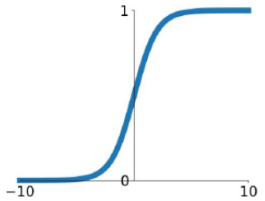
→ https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Activation functions

Adding a non-linearity

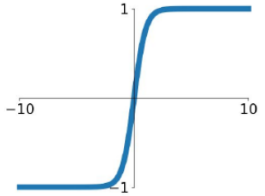
Sigmoid

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



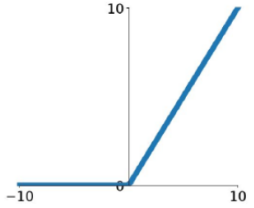
tanh

$$\tanh(x)$$



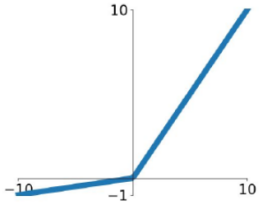
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

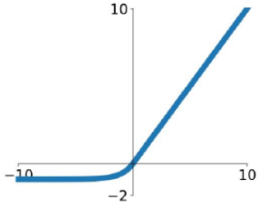


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

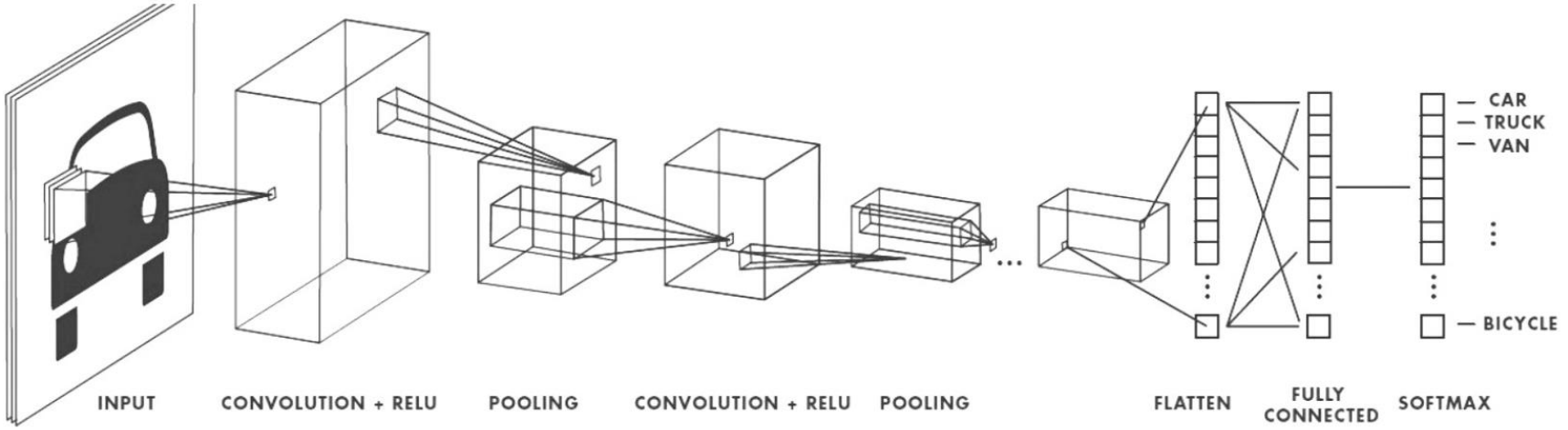
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



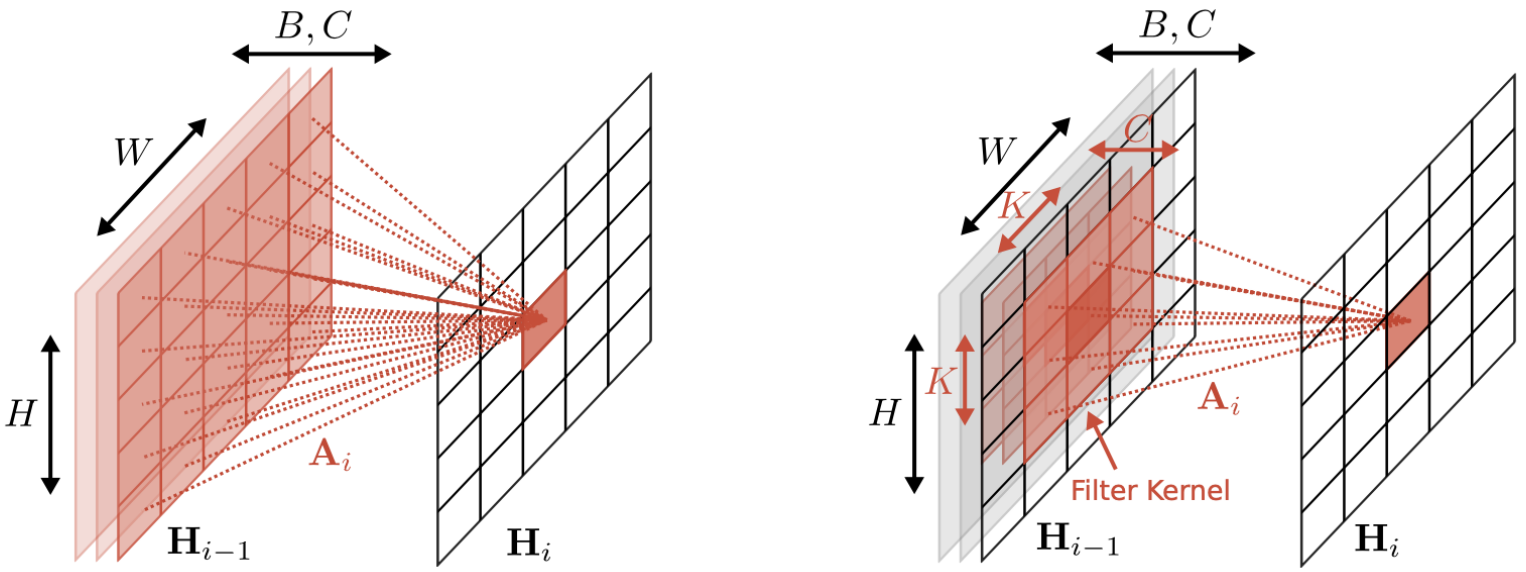
Deep Learning recap

Convolutional Neural Networks

→ CNNs often have fully-connected layers at the end



Fully connected vs Convolutional layers



- **Fully connected layer:** #Weights $= W \times H \times C_{out} \times (W \times H \times C_{in} + 1)$
- **Convolutional layer:** #Weights $= C_{out} \times (K \times K \times C_{in} + 1)$ ("weight sharing")
- With C_{in} input and C_{out} output channels, layer size $W \times H$ and kernel size $K \times K$
- Convolutions are followed by non-lineary activation functions (e.g., ReLU)

How does training work?

Optimization → Gradient descent

→ The function to optimize is not convex… no global minima

→ We use gradient descent

→ Good news! Multiple local minima **but** good local minima

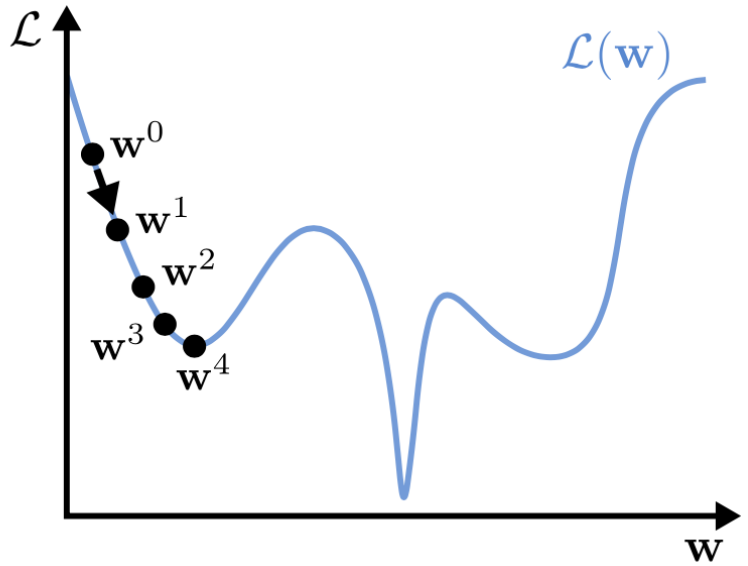
Optimization Problem: (dataset $\mathcal{X}$)

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\ \mathcal{L}(\mathcal{X}, \mathbf{w})$$

Gradient Descent:

$$\mathbf{w}^0 = \mathbf{w}^{\text{init}}$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} \mathcal{L}(\mathcal{X}, \mathbf{w}^t)$$



Backpropagation

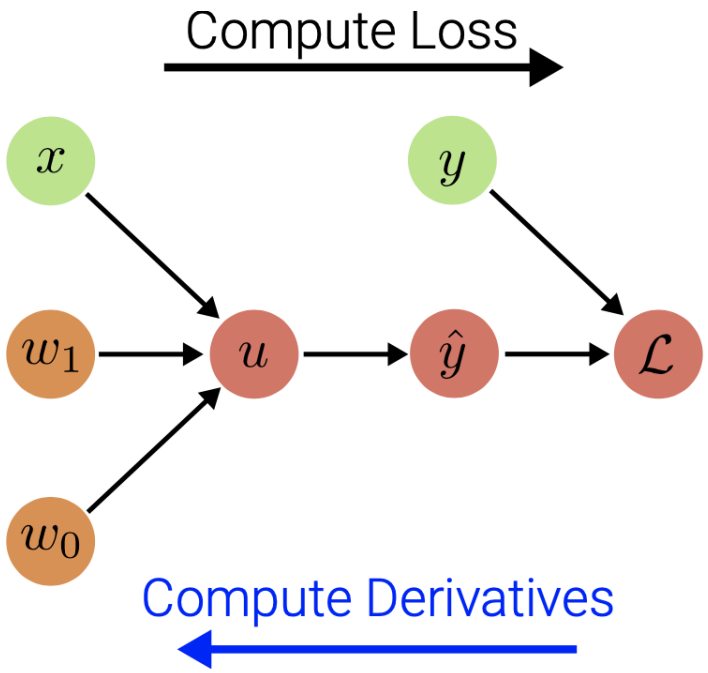
Computing gradients (derivatives) backwards

→ Values are computed forward, gradients backward

→ Modular! Each node only “knows” how to compute gradients with respect to itself (its own arguments)

→ Very simply said:

- How much am I responsible of the total loss?



Issues of gradient descent

And benefits of stochastic gradient descent

→ Gradient descent is very expensive to compute
- There are millions of training points !

→ Therefore, we use “stochastic gradient descent”
- Or said otherwise, we don’t compute it for the whole dataset, but only for a subset of the data (=minibatch)
- And we pick minibatches randomly as long as error decreases

→ Many many variants to this optimization!
- Adam is often the method of choice due to its robustness

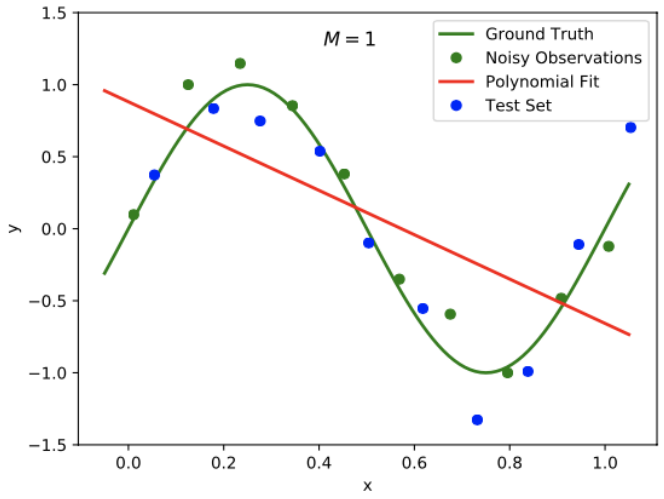
Overfitting and underfitting

How closely (and realistically) we match the right behavior

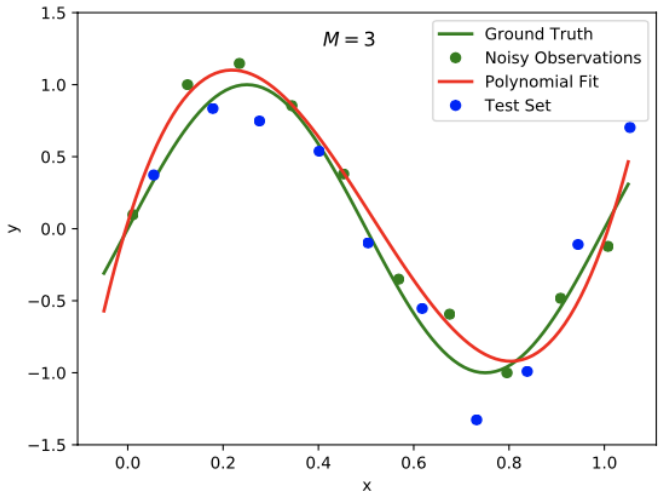
→ Underfitting: model too simple, does not achieve low error on training set

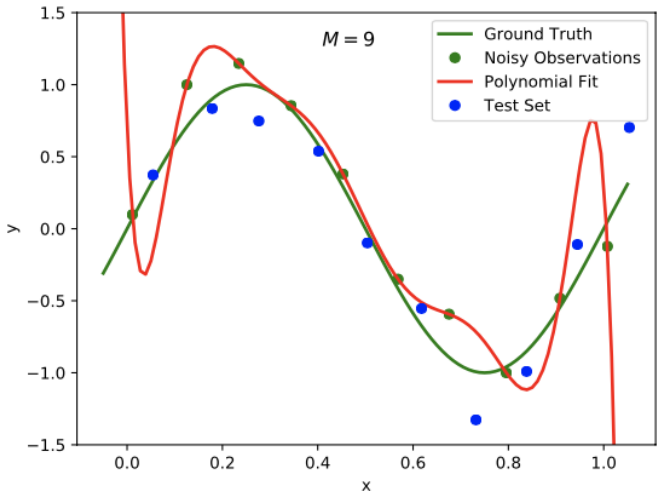
→ Overfitting: training error small, but test error (=generalization error) large

→ Regularization: moving from the righmost model (overfitted) to the one on the middle



Underfitting





Overfitting

TODO's for today

Exercises

1. Try the moral machine!
   - http://moralmachine.mit.edu/
2. Don't hesitate to spend some time watching videos
   - On the slides, and the AI Day Tesla videos linked on Cyberlearn
3. Install Gymnasium and run the car
   - Useful for next lectures
4. Check out the Carla simulator and how it works
5. Getting acquainted to our Drone for the lab (if time allows)

Gymnasium car racing

Installation steps (in a Python venv in the virtual machine)

- sudo apt install swig
- python3 -m venv IAAcarRacing
- source IAAcarRacing/bin/activate

- In the environment:
- pip install wheel
- pip install gymnasium[box2d]
- python IAAcarRacing/lib64/python3.8/site-packages/gymnasium/envs/box2d/car_racing.py

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale