Self-Driving Cars

Lecture 2 - Imitation Learning

Prof. Dr.-Ing. Andreas Geiger

Autonomous Vision Group
MPI-IS / University of Tübingen

October 25, 2018

University of Tübingen
MPI for Intelligent Systems

Autonomous Vision Group

Agenda

Date	Lecture (Thursday)	Date	Exercise (Friday)
18.10.	01 - Introduction to Self-Driving Cars	19.10.	00 - Introduction Pytorch & OpenAI Gym
25.10.	02 - DNNs, ConvNets, Imitation Learning	26.10.	01 - Intro: Imitation Learning
1.11.	none (Allerheiligen)	2.11.	
8.11.	03 - Direct Perception	9.11.	01 - Q&A
15.11.	none (CVPR Deadline)	16.11.	
22.11.	04 - Reinforcement Learning	23.11.	01 - Discussion & 02 - Intro: Reinforcement Learning
29.11.	05 - Vehicle Dynamics & Control	30.11.	
6.12.	06 - Localization & Visual Odometry	7.12.	02 - Q&A
13.12.	07 - Simultaneous Localization and Mapping (J. Stückler)	14.12.	
20.12.	08 - Road and Lane Detection	21.12.	02 - Discussion & 03 - Intro: Modular Pipeline
10.1.	09 - Reconstruction and Motion Estimation	11.1.	
17.1.	10 - Object Detection & Tracking	18.1.	
24.1.	11 - Scene Understanding	25.1.	03 - Q&A
31.1.	12 - Planning	1.2.	03 - Discussion & Announcement of Winners
7.2.	13 - Winner's Presentations and Exam Q&A	8.2.	

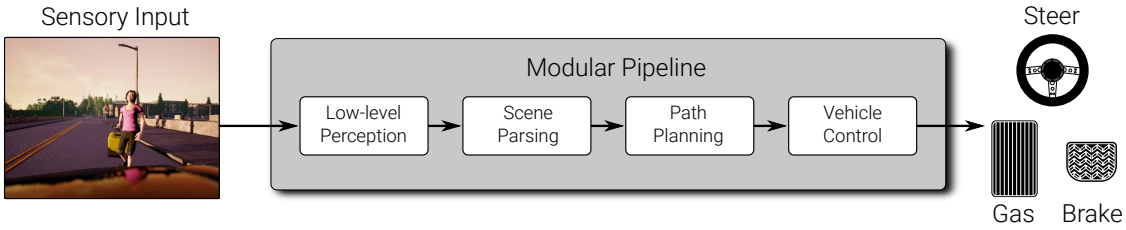
Approaches to Self-Driving

Autonomous Driving

Dominating Paradigms:

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

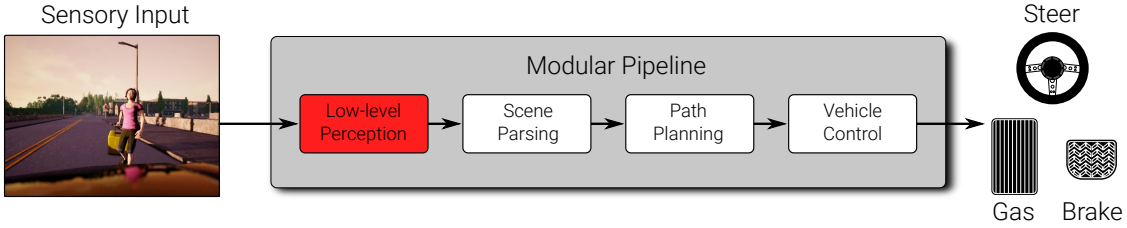
Autonomous Driving: Modular Pipeline

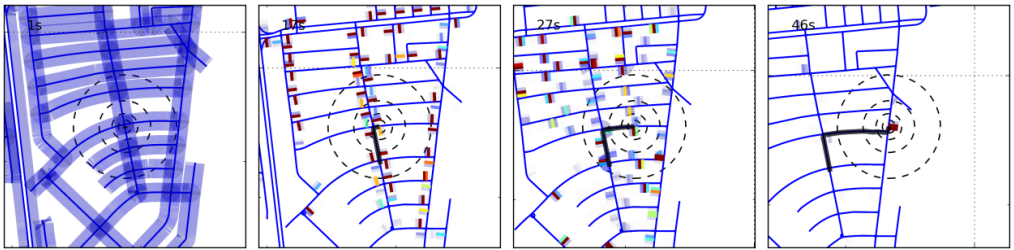


Examples:

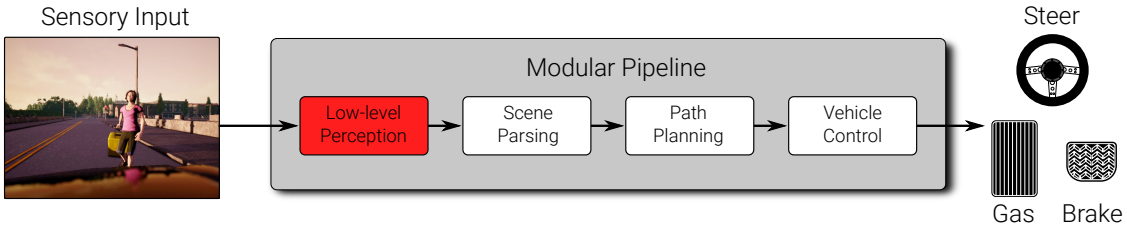
- [Montemerlo et al., JFR 2008]
- [Urmson et al., JFR 2008]
- Waymo, Uber, Tesla, Zoox, …

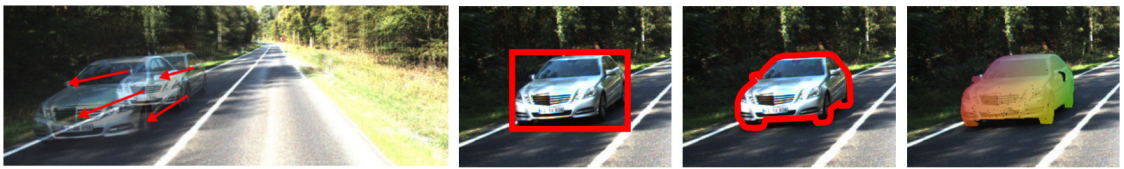
Autonomous Driving: Modular Pipeline



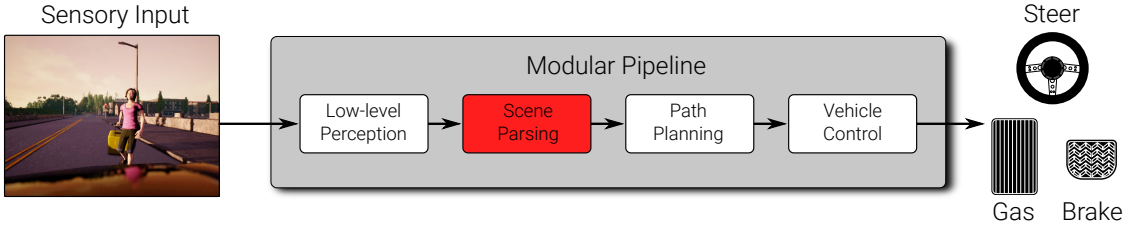


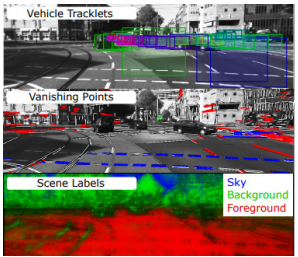
Autonomous Driving: Modular Pipeline

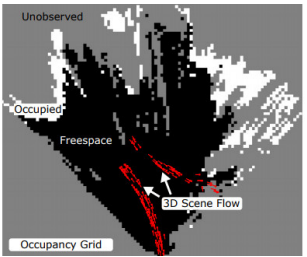


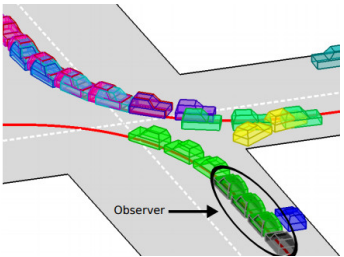


Autonomous Driving: Modular Pipeline

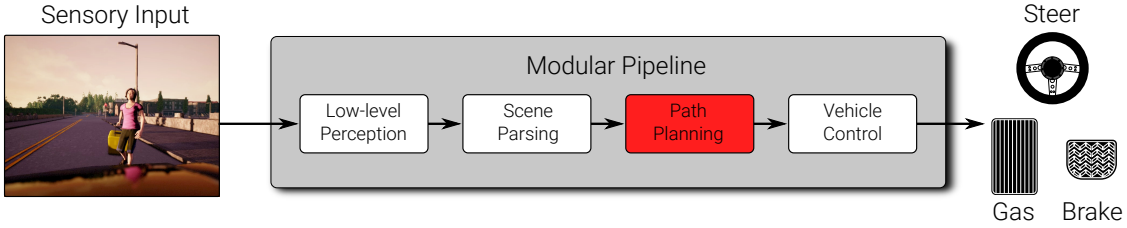


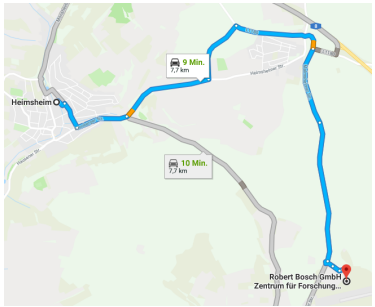


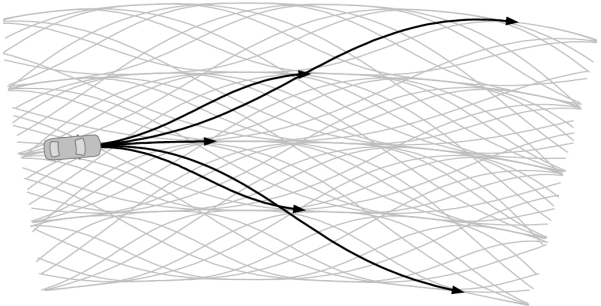




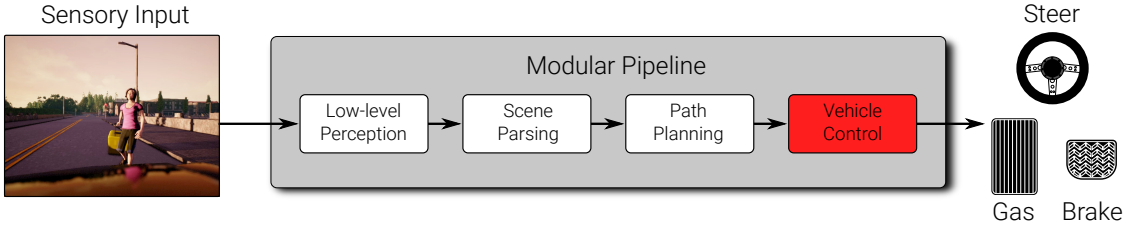
Autonomous Driving: Modular Pipeline

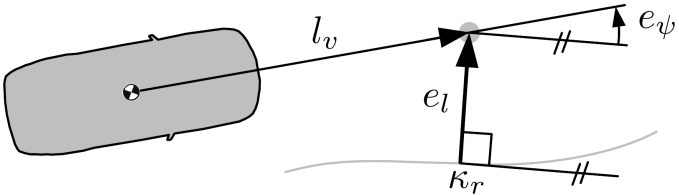




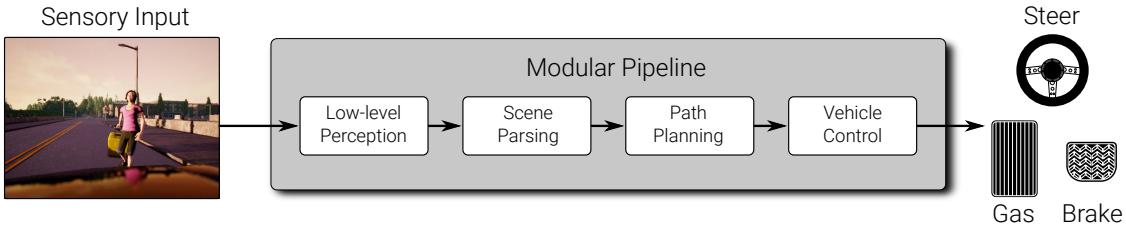


Autonomous Driving: Modular Pipeline





Autonomous Driving: Modular Pipeline



Pros:

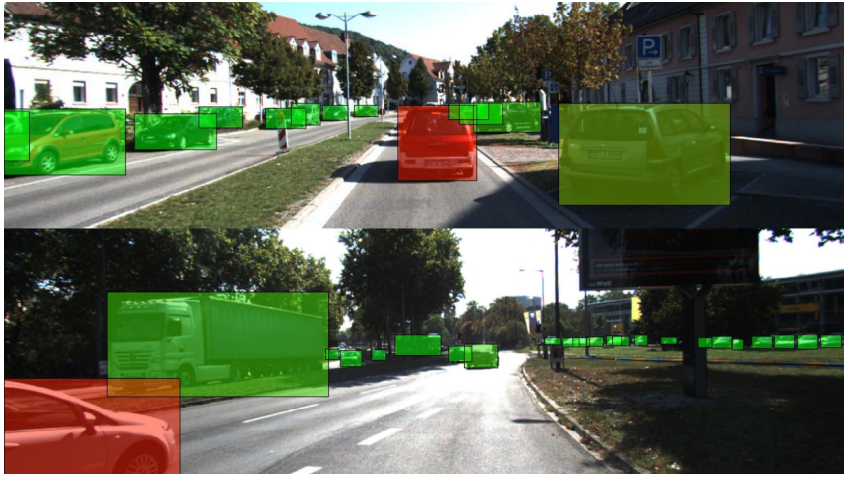
- Small components, easy to develop in parallel
- Interpretability

Cons:

- Piece-wise training (not jointly)
- Localization and planning heavily relies on HD maps

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

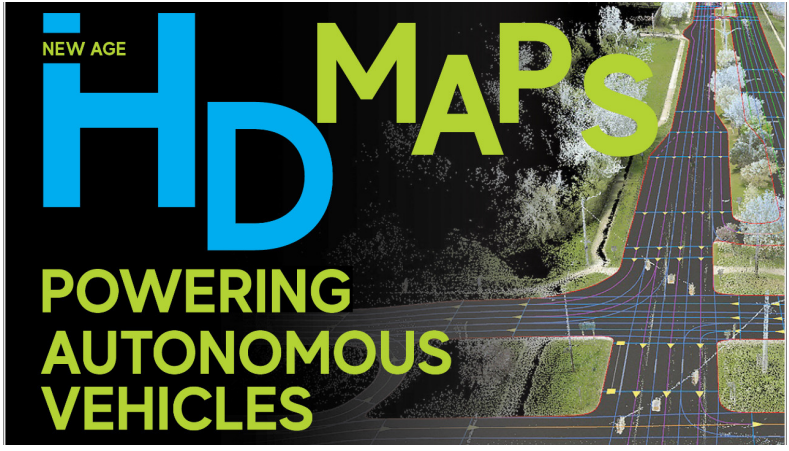
Autonomous Driving: Modular Pipeline



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

Ohn-Bar and Trivedi: Are All Objects Equal? Deep Spatio-Temporal Importance Prediction in Driving Videos. PR, 2017.

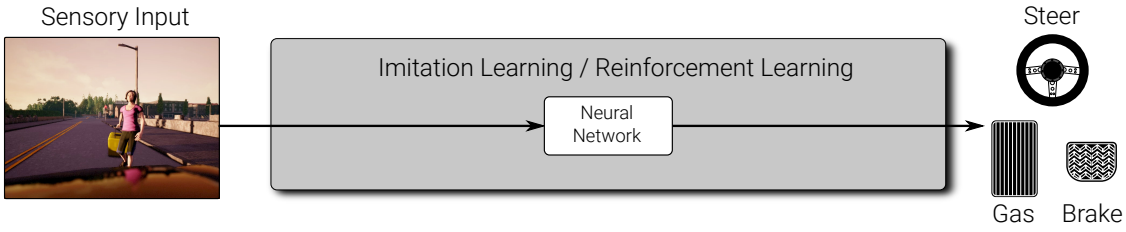
Autonomous Driving: Modular Pipeline



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

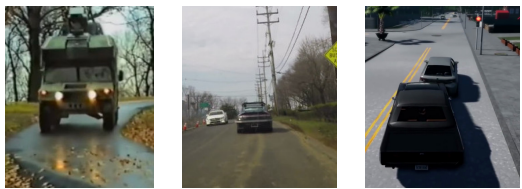
- HD Maps are expensive to create (data collection & annotation effort)

Autonomous Driving: End-to-End Learning

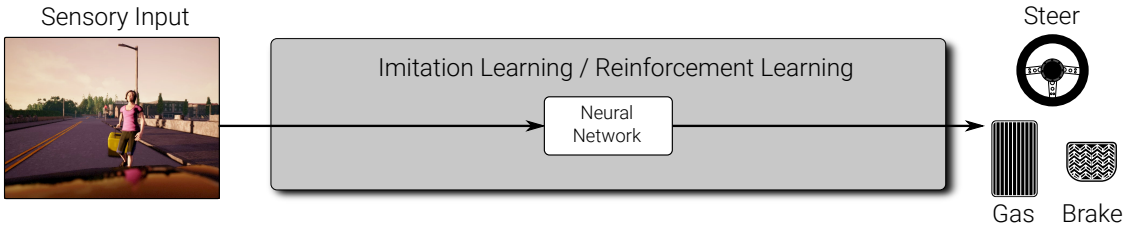


Examples:

- [Pomerleau, NIPS 1989]
- [Bojarski, Arxiv 2016]
- [Codevilla et al., ICRA 2018]



Autonomous Driving: End-to-End Learning



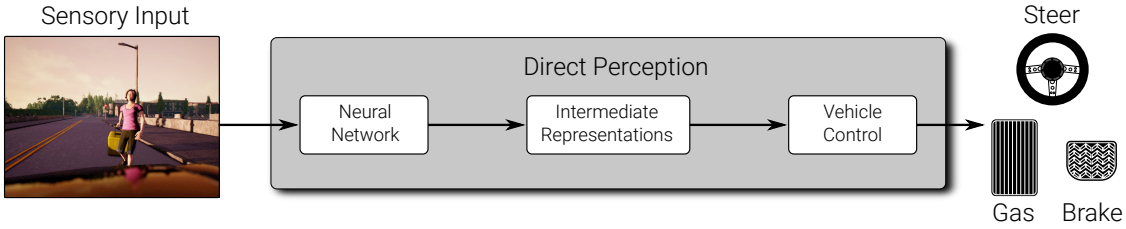
Pros:

- End-to-end training
- Cheap annotations

Cons:

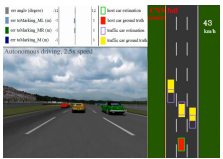
- Training / Generalization
- Interpretability

Autonomous Driving: Direct Perception

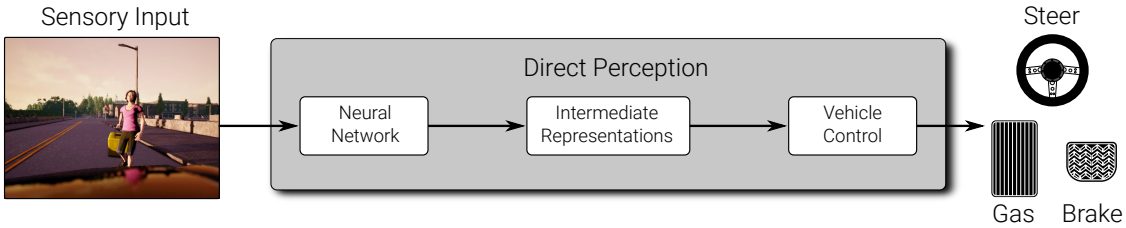


Examples:

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



Autonomous Driving: Direct Perception



Pros:

- Compact Representation
- Interpretability

Cons:

- Control typically not learned jointly
- How to choose representations?

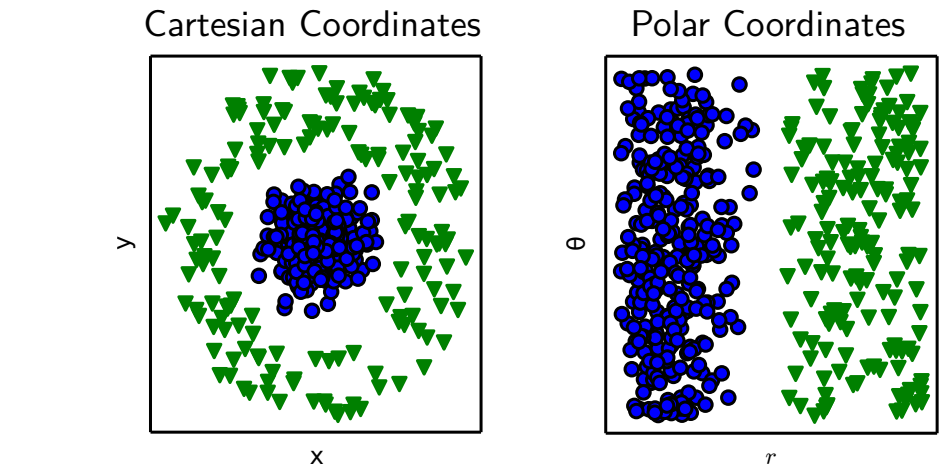
Topic for Today:

Imitation Learning

But before a little recap on ...

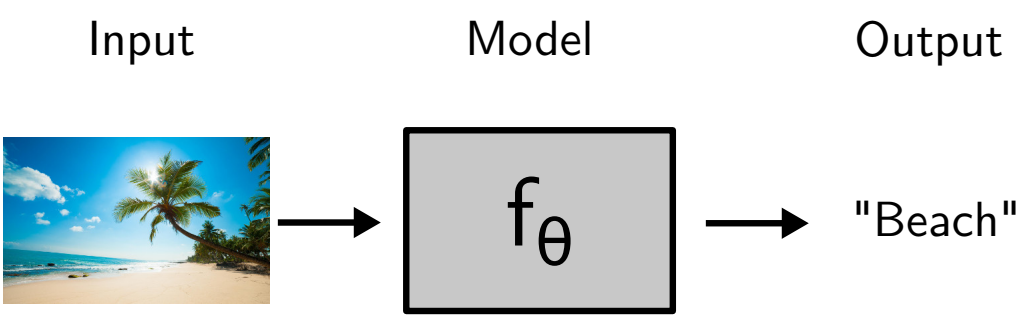

Deep Learning

Motivation: Representation Matters



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

Example: Multi-Class Classification



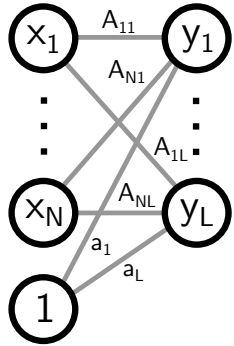
- ~~$f_\theta : \mathbf{x} \in \mathbb{R}^{W \times H} \mapsto y \in \{1, \ldots, L\}$~~
- $f_\theta : \mathbf{x} \in \mathbb{R}^{W \times H} \mapsto \mathbf{y} = [0, \ldots, 1, \ldots, 0] \in \{0, 1\}^L$

Linear Regression

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

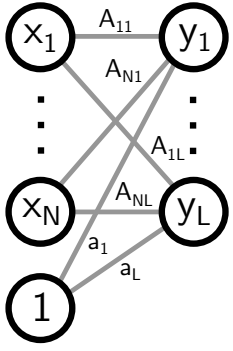
Linear Regression

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



Linear Regression

- Mapping:
  $\mathbf{x} =$ Image
  $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{a}$
- Classification:
  $L^* = \text{argmax}_{\,l}\ y_l$



Linear Regression

- 3 Layers:

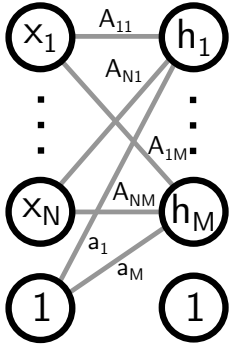
  $\mathbf{x} =$ Image

Linear Regression

- 3 Layers:

  $\mathbf{x} =$ Image

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



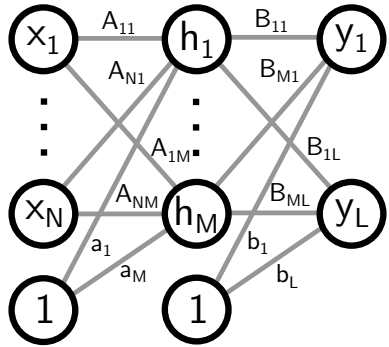
Linear Regression

- 3 Layers:

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

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

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$



Linear Regression

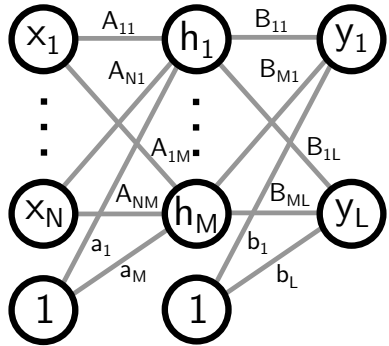
- 3 Layers:

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

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

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$

- Is this model better?



Linear Regression

- 3 Layers:

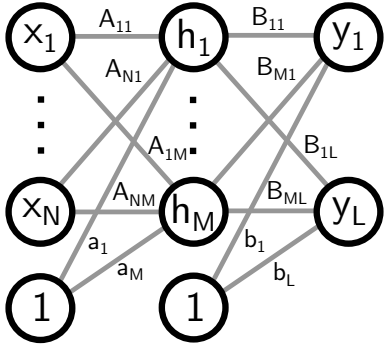
  $\mathbf{x} =$ Image

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

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$

- Is this model better?

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$



Linear Regression

- 3 Layers:

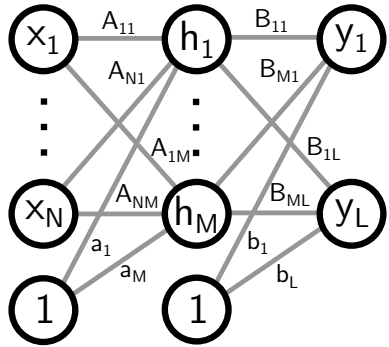
  $\mathbf{x} = \text{Image}$

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

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$

- Is this model better?

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



Linear Regression

- 3 Layers:

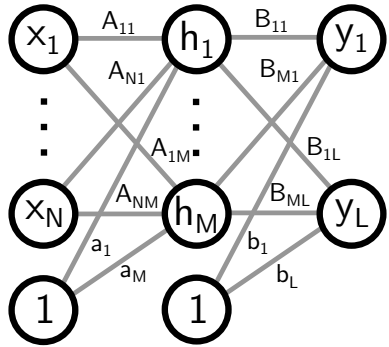
  $\mathbf{x} =$ Image

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

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$

- Is this model better?

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



Linear Regression

- 3 Layers:

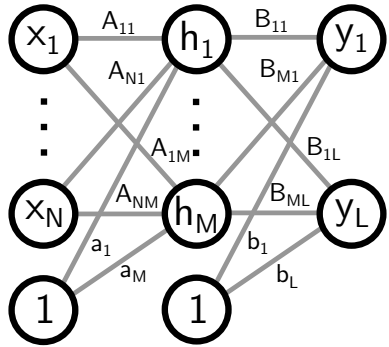
  $\mathbf{x} = \text{Image}$

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

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$

- Is this model better?

  $\mathbf{y} = \underbrace{\mathbf{B}\mathbf{A}}_{=\mathbf{C}}\mathbf{x} + \underbrace{\mathbf{B}\mathbf{a} + \mathbf{b}}_{=\mathbf{c}}$



Linear Regression

- 3 Layers:

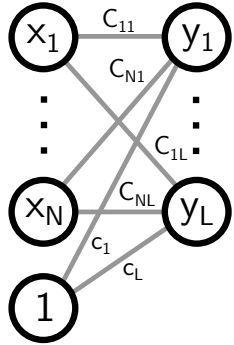
  $\mathbf{x} =$ Image

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

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$

- Is this model better?

  $\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{c}$

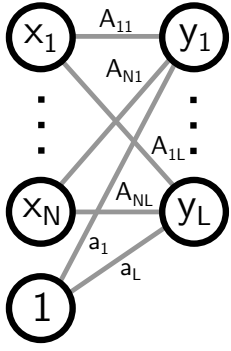


Logistic Regression

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

Logistic Regression

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



Logistic Regression

- Mapping:

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

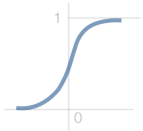
  $\mathbf{y} = \sigma(\mathbf{A}\mathbf{x} + \mathbf{a})$

- With (elementwise):

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

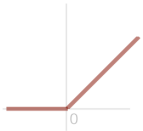
(a) Sigmoid

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



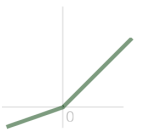
(b) ReLU

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



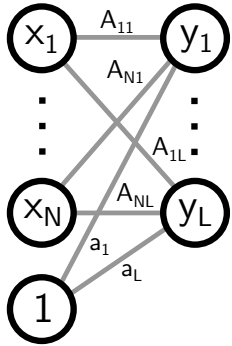
(c) PReLU

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



Logistic Regression

- Mapping:
  $\mathbf{x} = \text{Image}$
  $\mathbf{y} = \sigma(\mathbf{A}\mathbf{x} + \mathbf{a})$
- With (elementwise):
  $\sigma(x) = \frac{1}{1+\exp(-x)}$
- Classification:
  $L^* = \text{argmax}_l \; y_l$



Multilayer Perceptron

- 3 Layers:

  $\mathbf{x}$ = Image

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

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

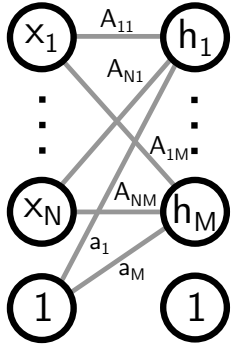
Multilayer Perceptron

- 3 Layers:

  $\mathbf{x} =$ Image

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

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



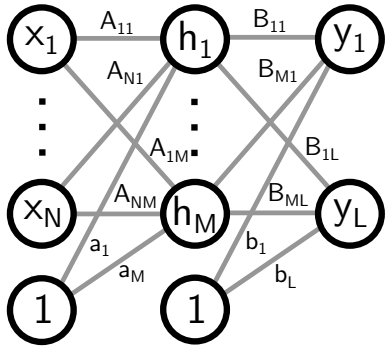
Multilayer Perceptron

- 3 Layers:

  $\mathbf{x}$ = Image

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

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



Multilayer Perceptron

- 3 Layers:

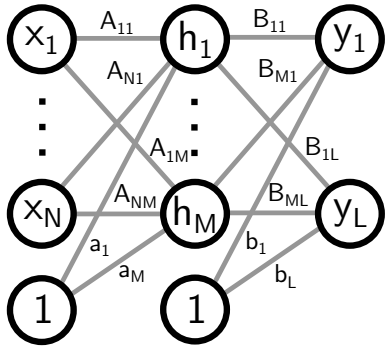
  $\mathbf{x} = \text{Image}$

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

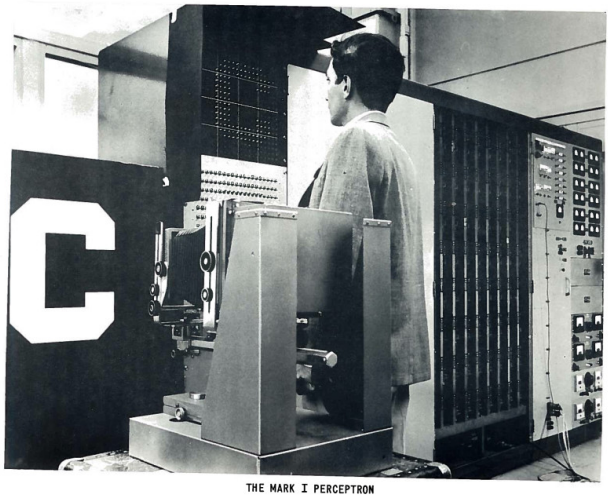
  $\mathbf{y} = \sigma(\mathbf{B}\mathbf{h} + \mathbf{b})$

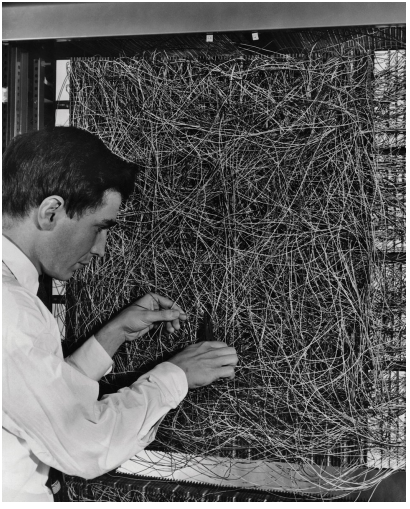
- Now:

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



Perceptron: First Hardware Implementation (2 Layers)

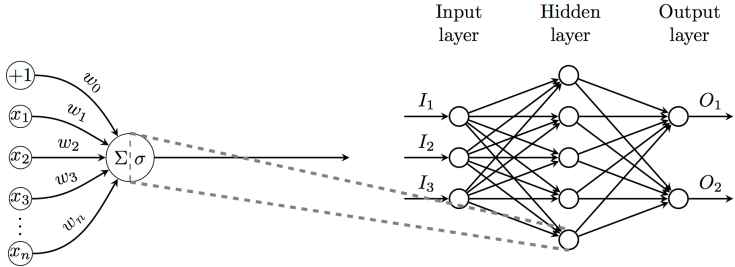




Mark 1 Perceptron, Frank Rosenblatt, 1958

Rosenblatt: The Perceptron—a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

Summary: Multilayer Perceptron in a Nutshell

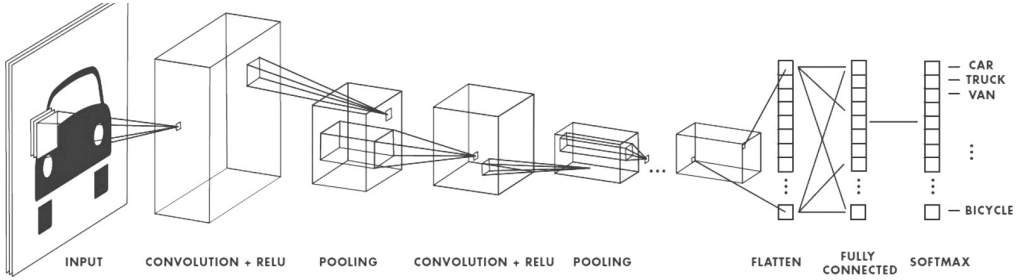


Layer j in a Multilayer Perceptron:

$$y_j = \sigma(x_j) = \underbrace{\frac{1}{1+\exp\{-x_j\}}}_{\text{activation function}} \qquad x_j = \underbrace{\sum_i w_{ji} y_i}_{\text{weighted sum}}$$

- Linearly combines *all* outputs y_i of previous layer i into *one* weighted sum x_j
  - Thus the layers in a MLP are also called “fully connected layers”
- Applies non-linear activiation function $\sigma(x_j)$, yielding new output y_j

Convolutional Neural Networks



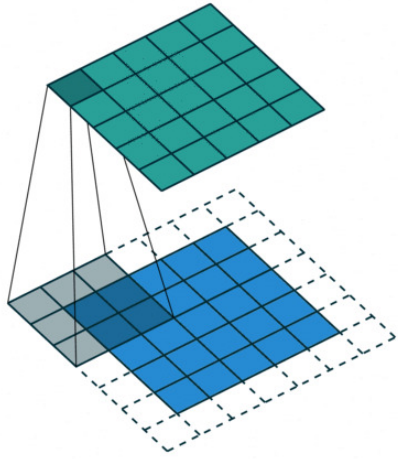
- Images are typically very high-dimensional. What is the problem with MLPs?
- ConvNets represent data in 3 dimensions: width, height, depth (= feature maps)
- ConvNets interleave discrete convolutions, non-linearities and pooling
- The learned weights now correspond to the convolution kernel parameters
- Key ideas: sparse interactions, parameter sharing, equivariant representation
- Biological example: simple cells and complex cells in primary visual cortex (V1)

Building Block: Convolution Layer

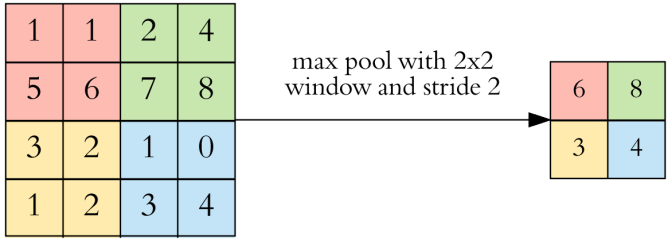
Convolution Operation:

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

- 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
- Let’s do a little quiz!



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?
- Convolutions can also be strided!

LeNet-5

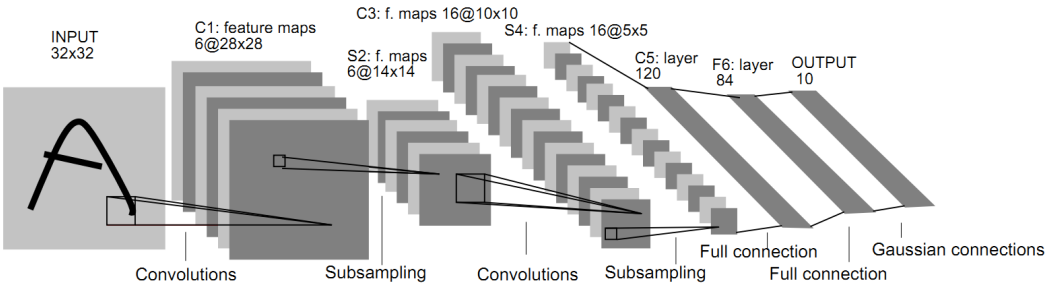


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- First implementation of convolutional net with learned weights
- 2 convolution layers (5x5), 2 subsampling layers (stride 2), 2 fully connected layers

LeCun, Bottou, Bengio and Haffner: Gradient-based learning applied to document recognition. IEEE, 1998.

MNIST Digit Classification

LeCun, Bottou, Bengio and Haffner: Gradient-based learning applied to document recognition. IEEE, 1998.

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 3×3 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 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	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 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					


Parameter Learning

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 $M << N$ randomly picked data points at each iteration (typically $M = 16, \ldots, 128$ as gradients too noisy for $M = 1$)
- In other words: $\nabla_{\mathbf{W}} E \approx \sum_m \nabla_{\mathbf{W}} E_m$. But how to calculate $\nabla_{\mathbf{W}} E_m$?

Backpropagation

[Rumelhart et al., Nature 1986]

Backpropagation on two Slides

Neural Network Layer:

$$y_j = \frac{1}{1+\exp\{-x_j\}} \qquad x_j = \sum_i w_{ji} y_i$$

Goal: Calculate gradient $\nabla_{\mathbf{W}} E_m = \nabla_{\mathbf{W}} \frac{1}{2} \sum_j (y_{jm} - t_{jm})^2$ (dropping m for simplicity)

$$\frac{\partial E}{\partial y_j} = y_j - t_j$$

$$\underbrace{\frac{\partial E}{\partial x_j}}_{=:\delta_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_j} = (y_j - t_j) y_j (1 - y_j)$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial w_{ji}} = \delta_j y_i$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial y_i} = \sum_j \delta_j w_{ji}$$

$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial x_i} = \underbrace{\boxed{\delta_i := y_i(1-y_i)\sum_j \delta_j w_{ji}}}_{\text{Backpropagation formula}}$$

Rumelhart, Hinton and Williams: Learning representations by back-propagating errors. Nature, 1986.

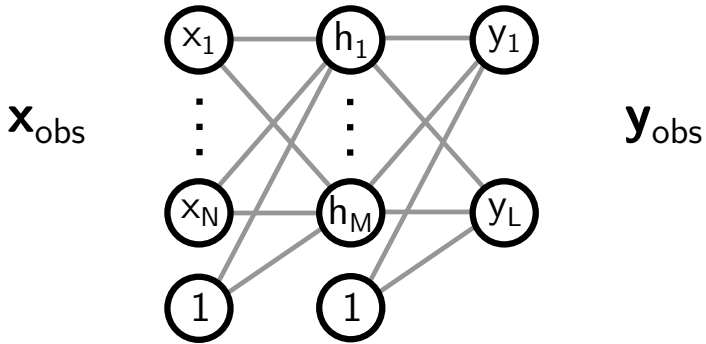
Backpropagation on two Slides

Algorithm for training a neural network using SGD:

1. Randomly initialize $\mathbf{W}$
2. Sample mini-batch $\mathcal{X}_{\text{batch}} \subset \mathcal{X}$
3. For each data point $\mathbf{x}_m \in \mathcal{X}_{\text{batch}}$ do:
   - 3.1 **Forward propagate** $\mathbf{x}_m$ through network: $y_j = \sigma\left(\sum_i w_{ji} y_i\right)$ (here: $\{y_0\} = \mathbf{x}_m$)
   - 3.2 **Calculate errors** δ_j at output nodes: $\delta_j = (y_j - t_j) y_j (1 - y_j)$
   - 3.3 **Backpropagate errors** δ_i through network: $\delta_i = y_i (1 - y_i) \sum_j \delta_j w_{ji}$
   - 3.4 **Calculate derivatives** $\frac{\partial E_m}{\partial w_{ji}} = \delta_j y_i$
4. Update gradients $\mathbf{W}^{t+1} = \mathbf{W}^t - \eta \sum_{m=1}^{M} \nabla_{\mathbf{W}} E_m |_{\mathbf{W}^t}$
5. If validation error decreases return to step 2, otherwise stop

Rumelhart, Hinton and Williams: Learning representations by back-propagating errors. Nature, 1986.

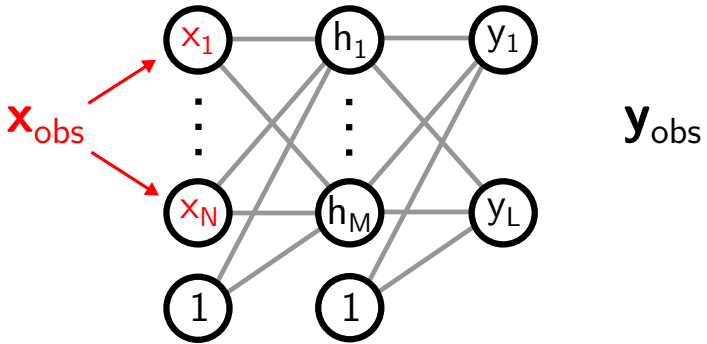
Illustration: Forward Pass



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

Rumelhart, Hinton and Williams: Learning representations by back-propagating errors. Nature, 1986.

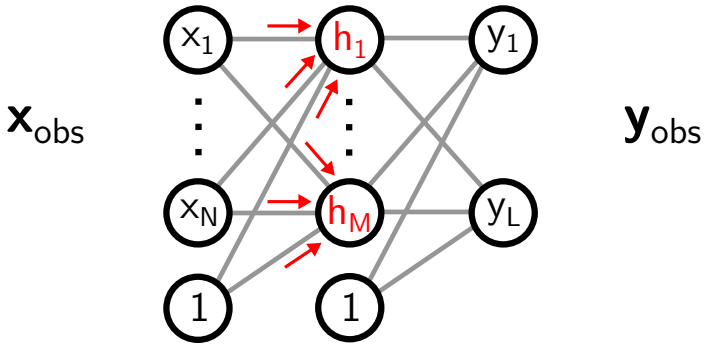
Illustration: Forward Pass



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

Rumelhart, Hinton and Williams: Learning representations by back-propagating errors. Nature, 1986.

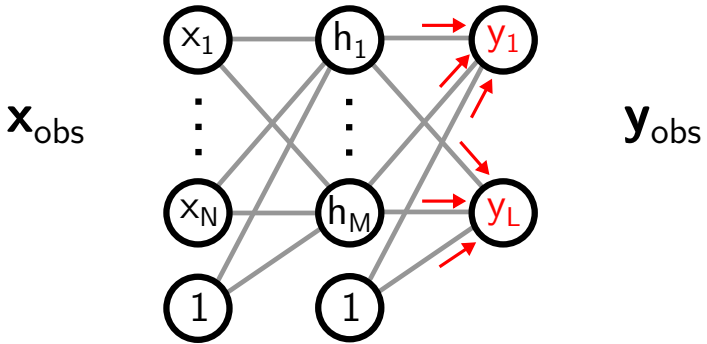
Illustration: Forward Pass



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

Rumelhart, Hinton and Williams: Learning representations by back-propagating errors. Nature, 1986.

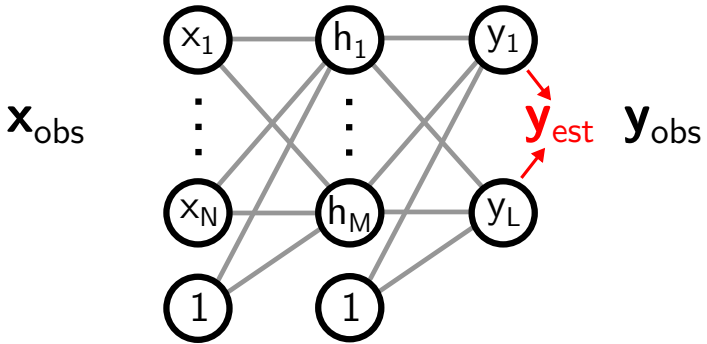
Illustration: Forward Pass



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

Rumelhart, Hinton and Williams: Learning representations by back-propagating errors. Nature, 1986.

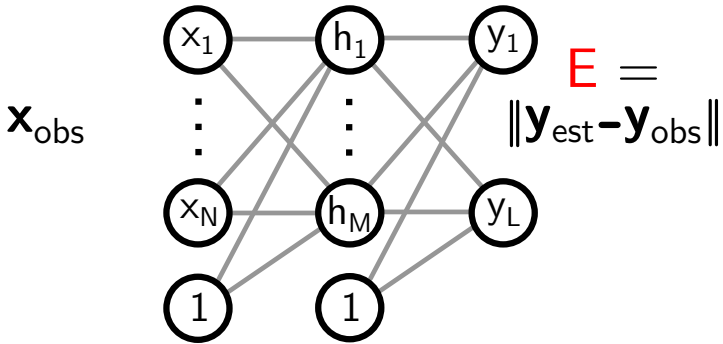
Illustration: Forward Pass



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

Rumelhart, Hinton and Williams: Learning representations by back-propagating errors. Nature, 1986.

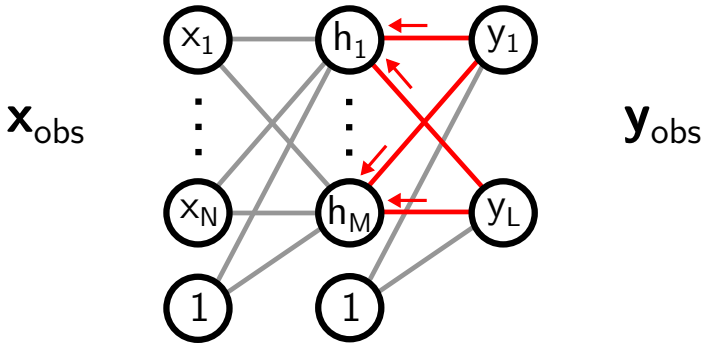
Illustration: Error Calculation



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

Rumelhart, Hinton and Williams: Learning representations by back-propagating errors. Nature, 1986.

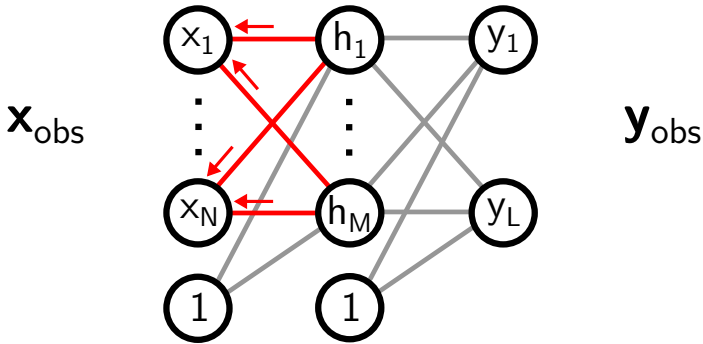
Illustration: Error Backpropagation



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

Rumelhart, Hinton and Williams: Learning representations by back-propagating errors. Nature, 1986.

Illustration: Error Backpropagation



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

Rumelhart, Hinton and Williams: Learning representations by back-propagating errors. Nature, 1986.

Comments on Backpropagation

Deep neural networks are usually trained using backpropagation.

- Unfortunately, neural nets are non-convex, many local minima
- Can get stuck in poor local minima, but: more parameters can help!
- Careful design needed
- Monitor loss, training and validation error

Overfitting remains a problem due to the large number of parameters.

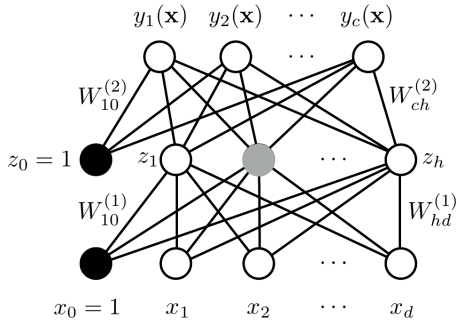
Measures to reduce overfitting?

- Weight decay (regularize weights): $E = E + \lambda \sum_j \sum_i w_{ji}^2$
- Drop-out (next slide)
- Data augmentation (next slide)

Drop-out

Idea:

- During forward pass: set every hidden unit to zero with probability 0.5
- Backward pass as before
- Avoids co-adaptation, introduces randomness, simulates network ensemble

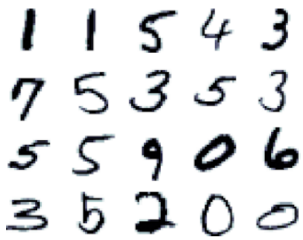


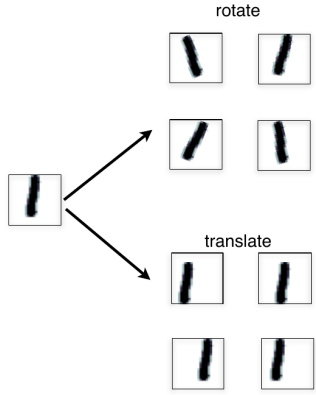
Hinton et al.: Improving neural networks by preventing co-adaptation of feature detectors. Arxiv, 2012.

Data Augmentation

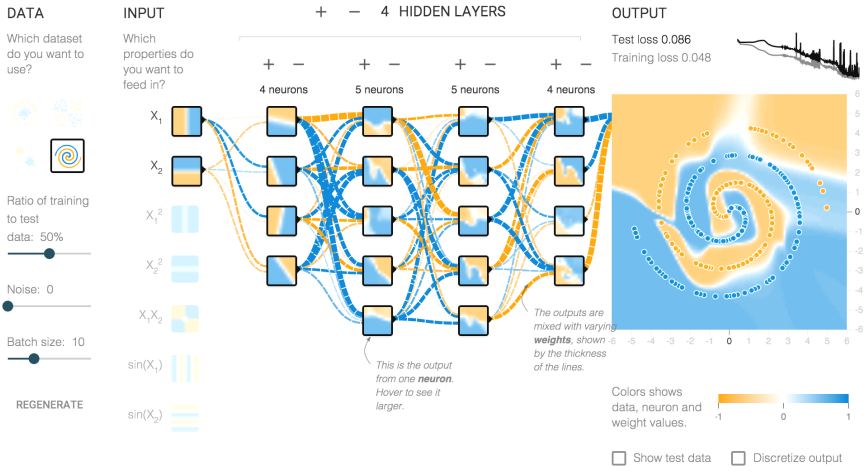
Idea:

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





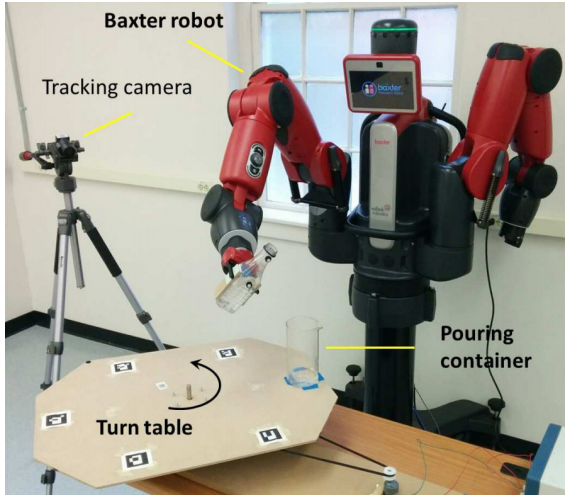
Tensorflow Playground



http://playground.tensorflow.org

Imitation Learning

Imitation Learning: Manipulation



Imitation Learning: Ghosting

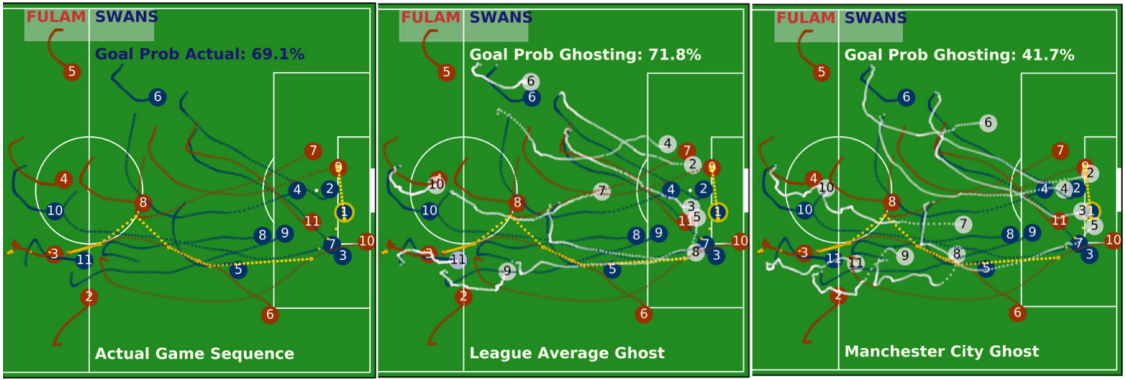
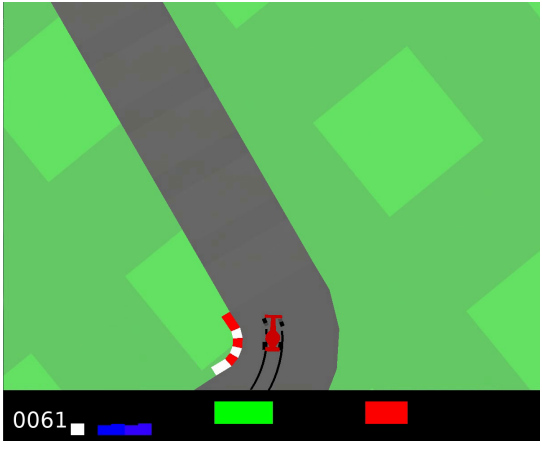


Figure 1 Our data-driven ghosting method can be applied to various game contexts to better understand defensive strategies. In the depicted scenario, Fulham (red) scores a goal on Swansea (blue). Ghosts (white) represent where Swansea defenders should have been according to a league average model (LAG) and Manchester City model (MUG).

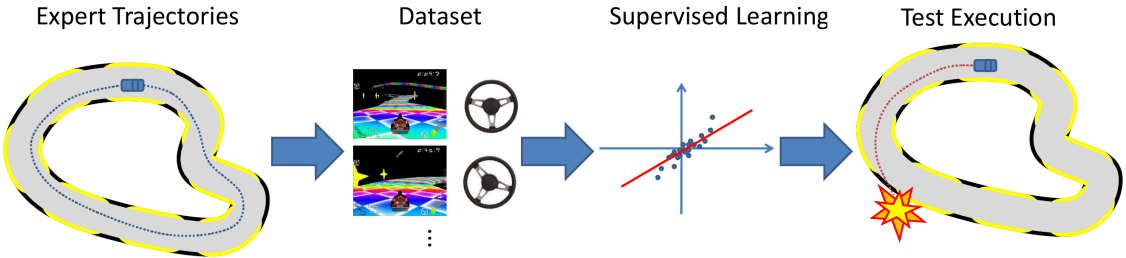
Imitation Learning: Car Racing



Trainer
(Human Driver)

Trainee
(Neural Network)

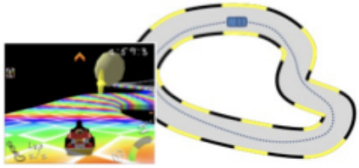
Imitation Learning in a Nutshell



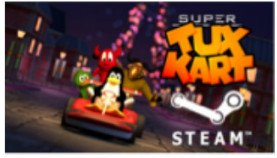
Hard coding policies is often difficult ⇒ Rather use a data-driven approach!

- **Given:** demonstrations or demonstrator
- **Goal:** train a policy to mimic decision
- **Variants:** behavior cloning (this lecture), inverse optimal control, …

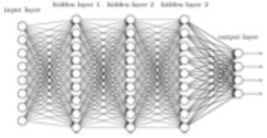
Ingredients of Imitation Learning



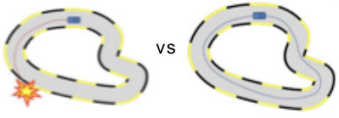
Demonstrations or Demonstrator



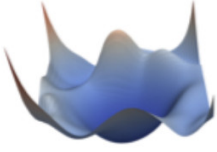
Environment / Simulator



Policy Class



Loss Function



Learning Algorithm

Formal Definition of Imitation Learning

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: $\pi_\theta : \mathcal{S} \to \mathcal{A}$ — we want to learn the policy parameters θ
- Optimal action: $a^* \in \mathcal{A}$ — provided by expert demonstrator
- Optimal policy: $\pi^* : \mathcal{S} \to \mathcal{A}$ — 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 $\tau = (s_0, a_0, s_1, a_1, \dots)$ — let's do an example!
- Loss function: $\mathcal{L}(a^*, a)$ — loss of action a given optimal action a^*

Formal Definition of Imitation Learning

General Imitation Learning:

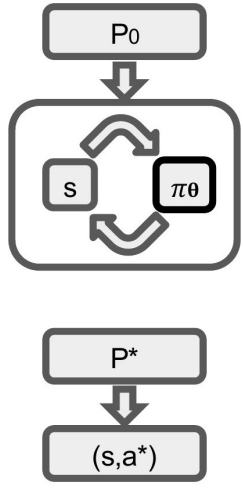
$$\underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{s\sim P(s|\pi_\theta)}\left[\mathcal{L}\left(\pi^*(s), \pi_\theta(s)\right)\right]$$

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

Behavior Cloning:

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

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

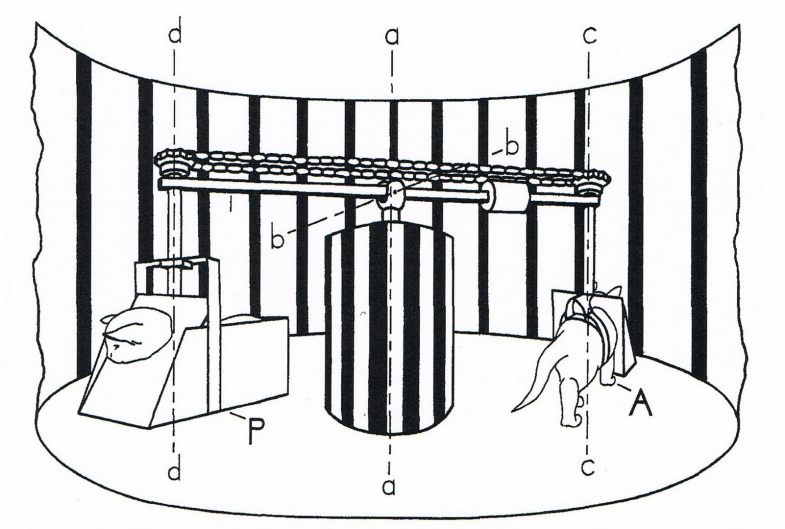


What is the problem?

Challenges of Behavior Cloning

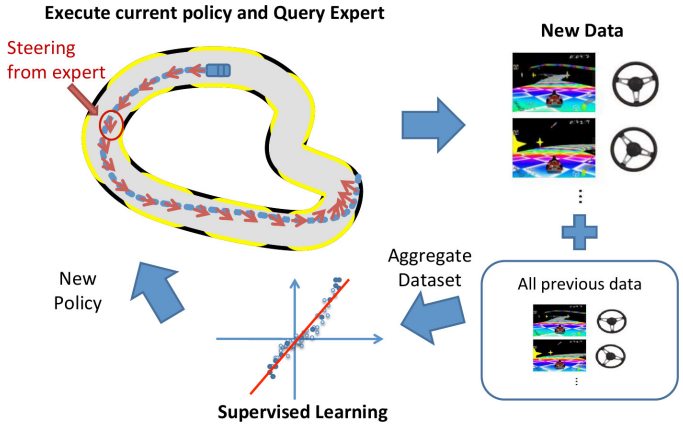
- Behavior cloning reasons only about immediate next step
- Behavior cloning makes IID assumption
  - Next state is sampled from states observed during expert demonstration
  - Thus, next state is sampled independently from action predicted by current policy
- What if π_θ makes a mistake?
  - Enters 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?

Experiment by Held and Hein



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

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.

Ross, Gordon and Bagnell: A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. AISTATS, 2011.

DAGGER: Dataset Aggregation

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

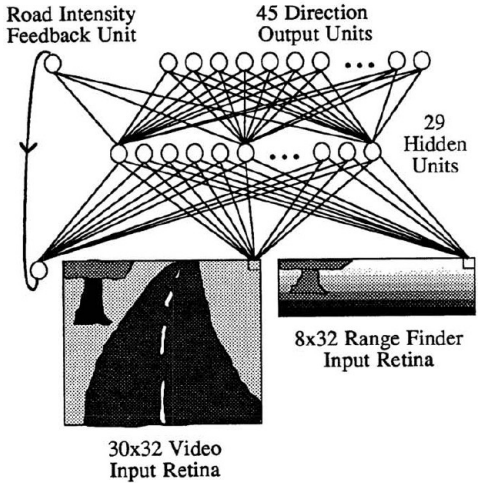
- Let T be the number of rollout time steps and ϵ be the probability of a mistake
- Behavior cloning: $O(T^2\epsilon)$ regret, DAGGER: no regret (in suitable conditions)
  - Regret = loss of current policy - loss of perfect policy
- But requires to flexibly run the environment and query the expert during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

Ross, Gordon and Bagnell: A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. AISTATS, 2011.

ALVINN: An Autonomous Land Vehicle in a Neural Network

ALVINN: An Autonomous Land Vehicle in a Neural Network

- Fully connected 3 layer neural net
- 36k parameters
- Maps road images to 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 70 mph



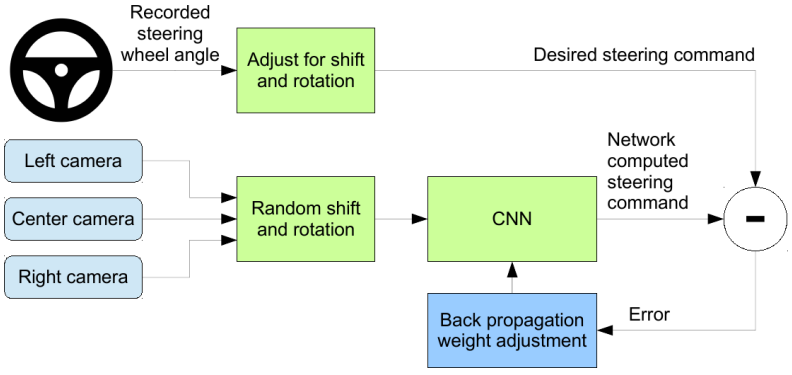
Pomerleau: ALVINN: An Autonomous Land Vehicle in a Neural Network. NIPS, 1988.

ALVINN: An Autonomous Land Vehicle in a Neural Network

Pomerleau: ALVINN: An Autonomous Land Vehicle in a Neural Network. NIPS, 1988.

End-to-End Learning for Self-Driving Cars

PilotNet: System Overview

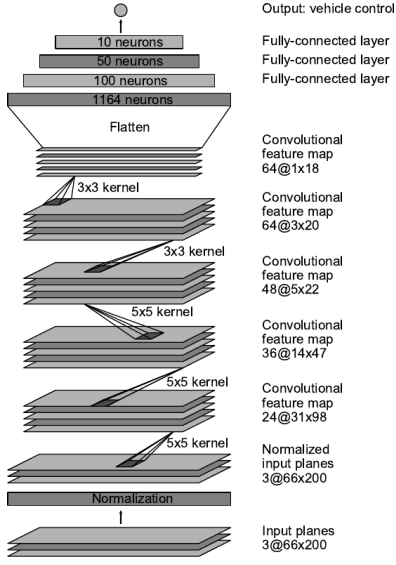


- Steering command represented as $1/r$ where r = turning radius in meters
- Data augmentation by 3 cameras and virtually shifted / rotated images assuming the world is flat (homography), adjusting the steering angle appropriately

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

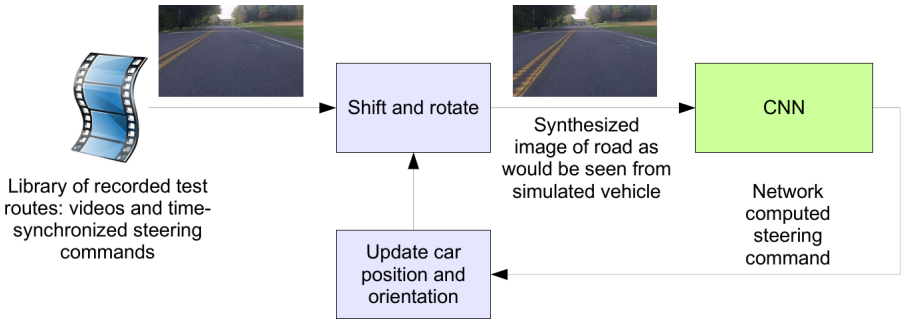
PilotNet: Architecture

- 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



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

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

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

PilotNet: Simulation

- Interventions (1m deviation) subtract 6 seconds from autonomy measure

PilotNet: Video

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

Further Readings

- 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.

Next Time:

Direct Perception

(in two weeks)

Questions?