High Level Computer Vision

Intro to Deep Learning for Computer Vision

Bernt Schiele - schiele@mpi-inf.mpg.de
Mario Fritz - mfritz@mpi-inf.mpg.de

https://www.mpi-inf.mpg.de/hlcv

most slides from: Rob Fergus & Marc'Aurelio Ranzato

Overview

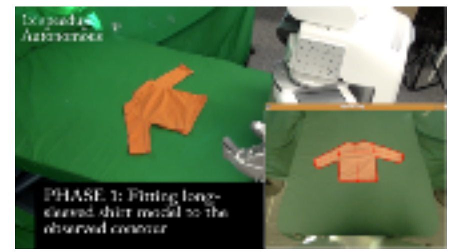
- My research
- Recent advance in Deep Learning
- What is Deep Learning?
- Model representation
- Learning Algorithm: Backprop
- Convolutional Neural Networks

- please note: no lecture next week

Research Profile

- Computer Vision
  - Object/Person/Activity/Material Rec. & De
  - Image/Video (Co) Segmentation
  - Vision & Human Gaze
  - Vision & Language
- Machine Learning
  - Topic Models/Bayesian Non-Parameterics
  - Unsupervised/Weakly/Zero-Shot/Active L.
  - Computation with Budget Constraints
  - Deep Learning
- Additional Research Areas
  - Natural Language Processing
  - Robotics
  - Graphics
  - HCI / UbiComp
  - Natural Sciences & Privacy

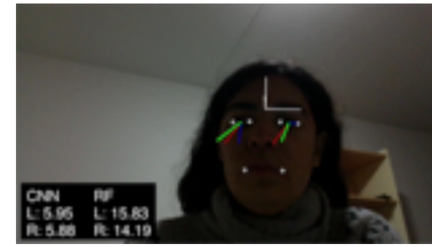




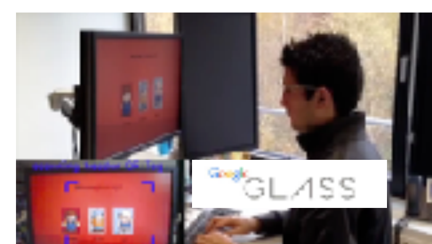
Research Profile

- Computer Vision
  - Object/Person/Activity/Material Rec. & De
  - Image/Video (Co) Segmentation
  - Vision & Human Gaze
  - Vision & Language
- Machine Learning
  - Topic Models/Bayesian Non-Parameterics
  - Unsupervised/Weakly/Zero-Shot/Active L.
  - Computation with Budget Constraints
  - Deep Learning
- Additional Research Areas
  - Natural Language Processing
  - Robotics
  - Graphics
  - HCI / UbiComp
  - Natural Sciences & Privacy

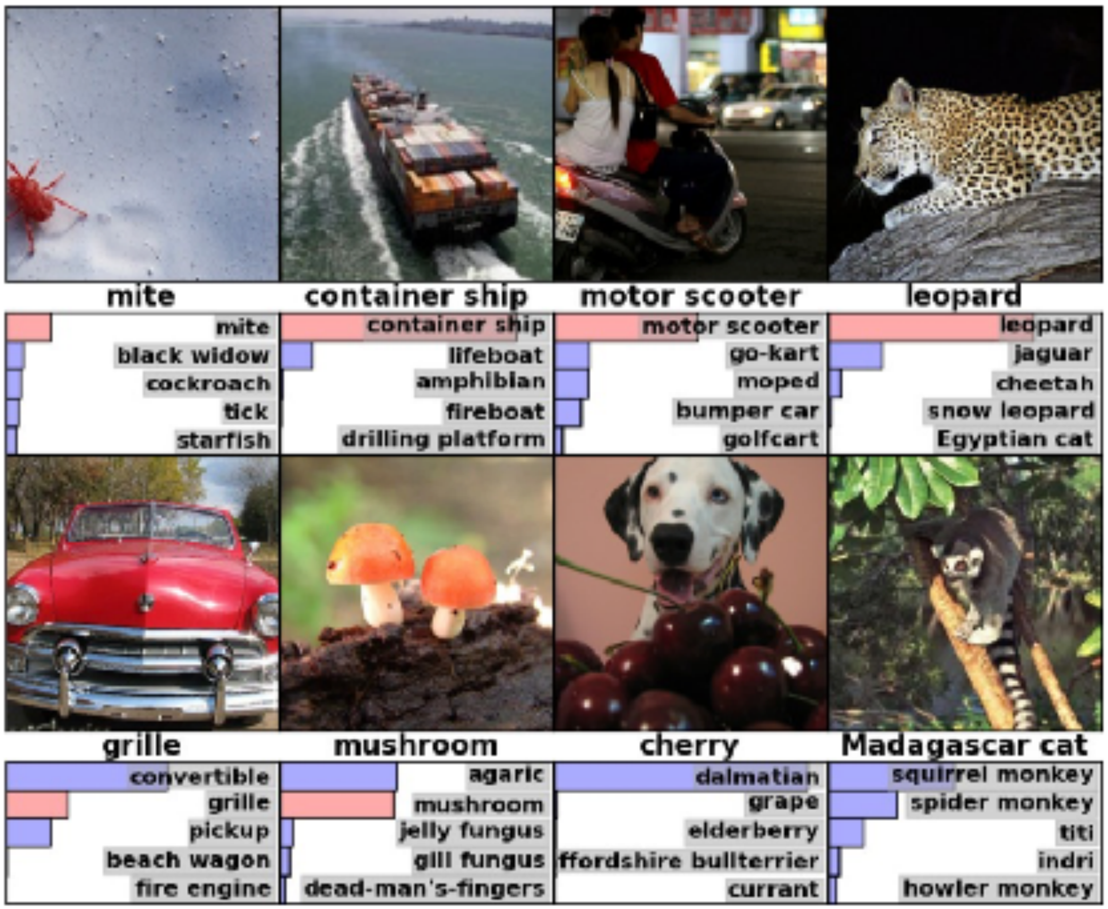






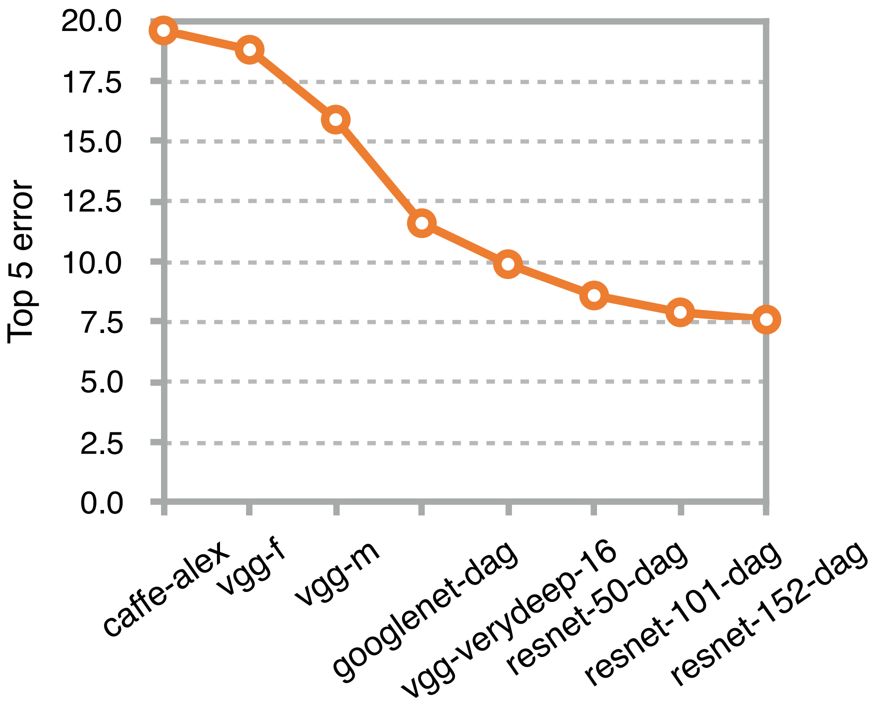
Computer Vision: Objekterkennung

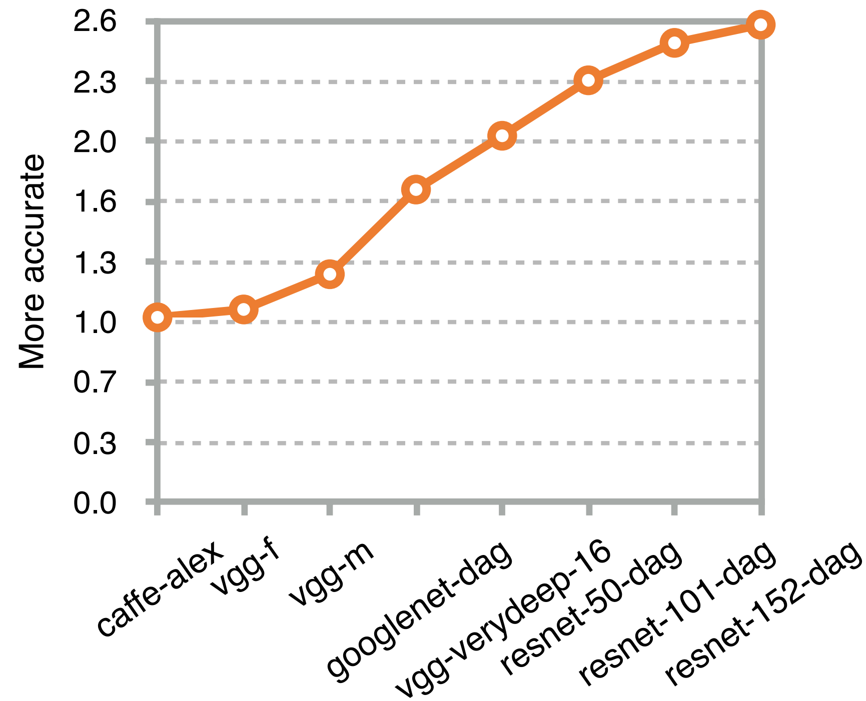


1000 Klassen; 1ms pro Bild; 92.5% Erkennungsrate

http://demo.caffe.berkeleyvision.org

Architectures get deeper

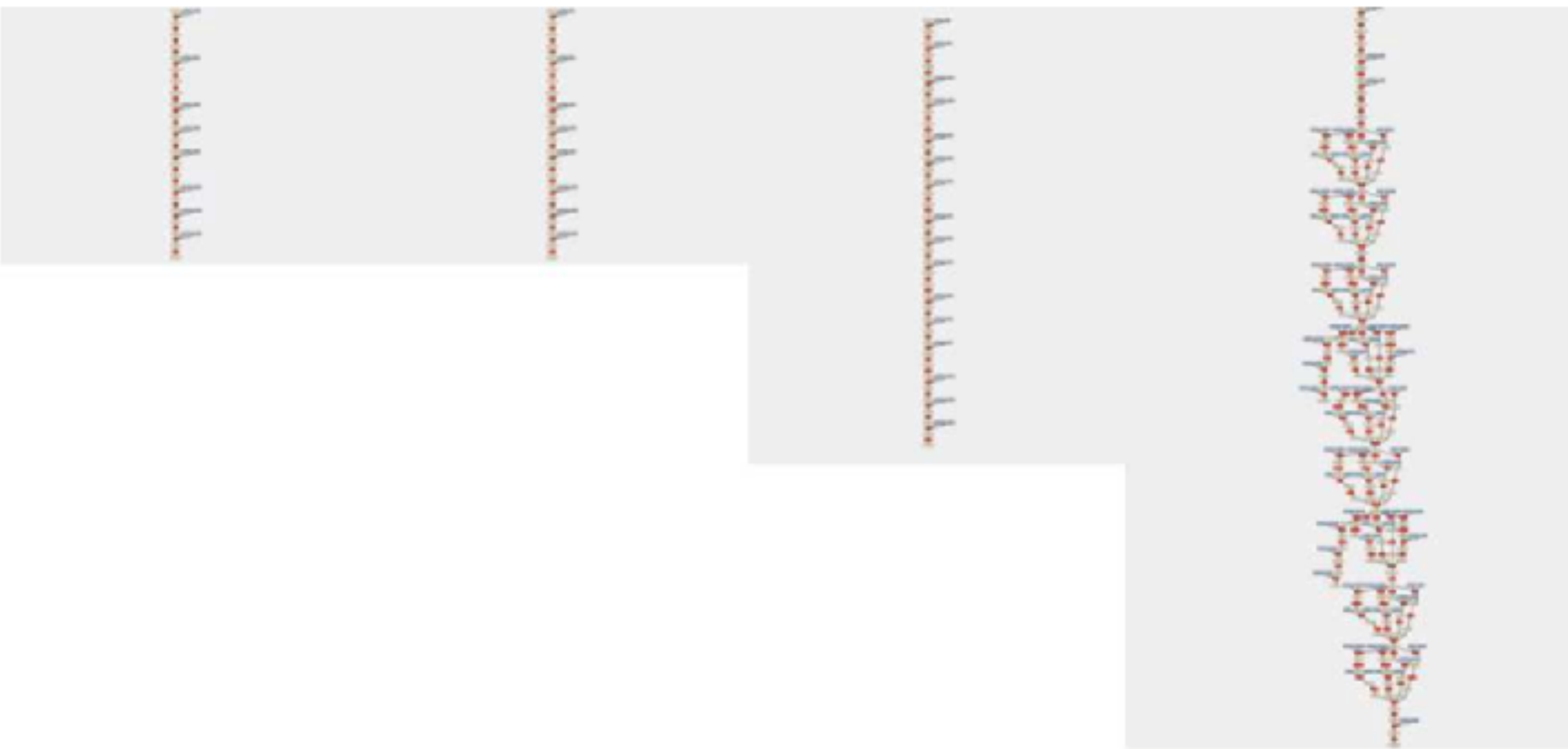




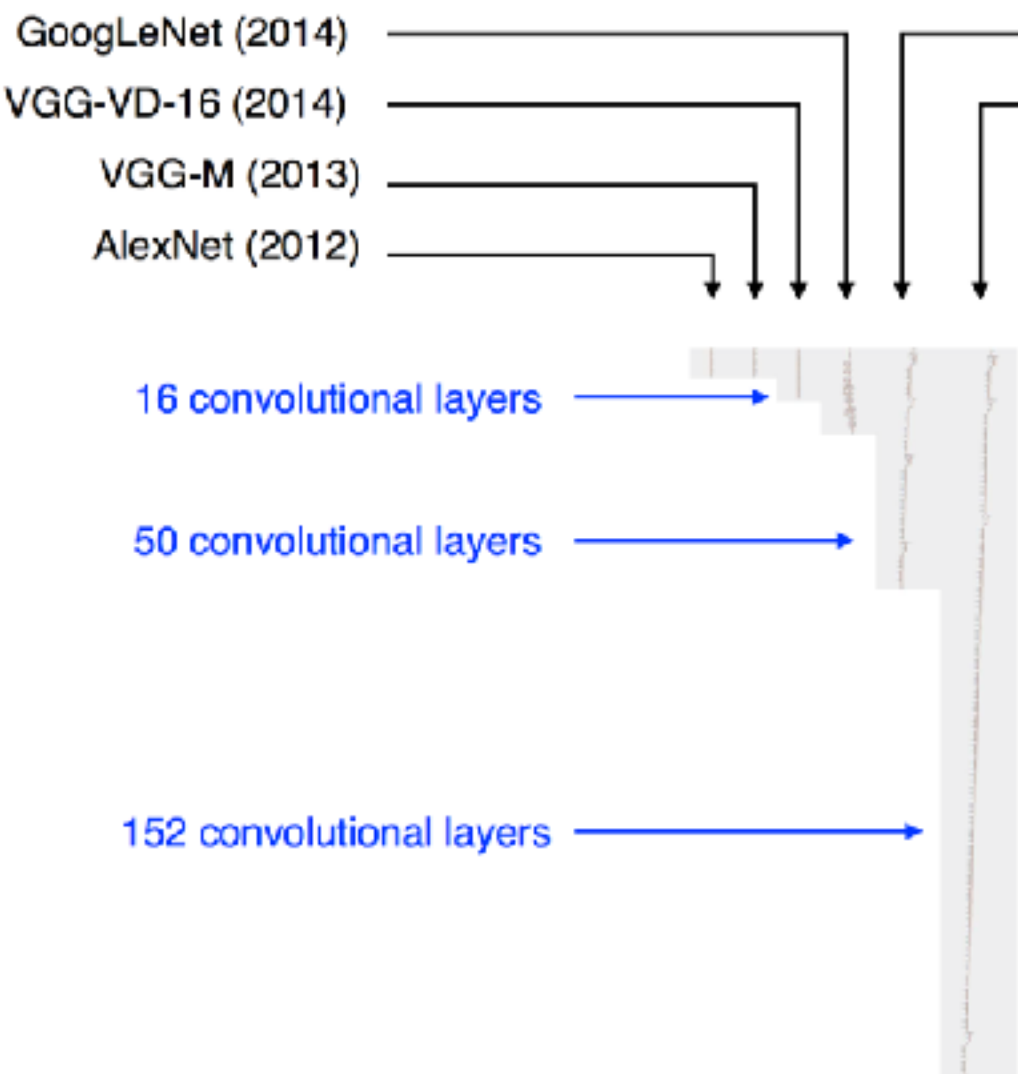
~ 2.6x improvement in 3 years

Architectures get deeper

AlexNet (2012) VGG-M (2013) VGG-VD-16 (2014) GoogLeNet (2014)



Architectures get deeper



Krizhevsky, I. Sutskever, and G. E. Hinton. *ImageNet classification with deep convolutional neural networks*. In Proc. NIPS, 2012.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going deeper with convolutions*. In Proc. CVPR, 2015.

K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. In Proc. ICLR, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition*. In Proc. CVPR, 2016.

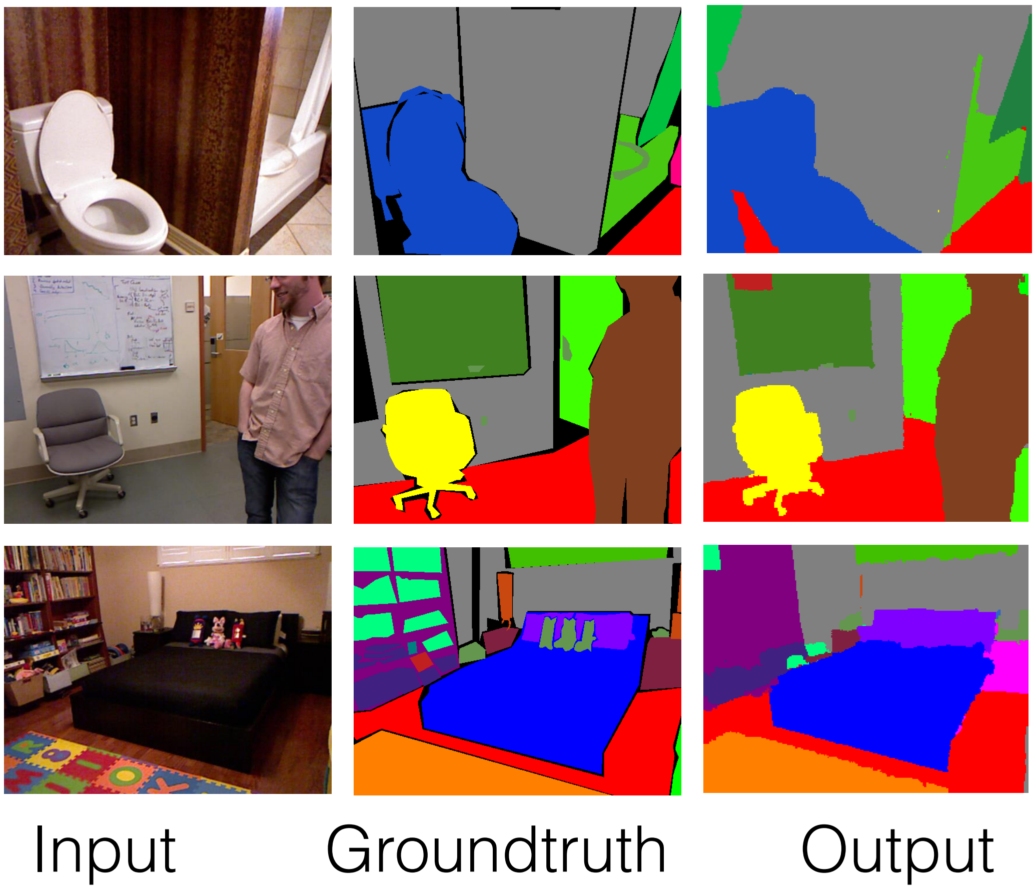
Computer Vision: Semantic Segmentation

Badrinarayanan et al. 2015
SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling

Computer Vision: Semantic Segmentation

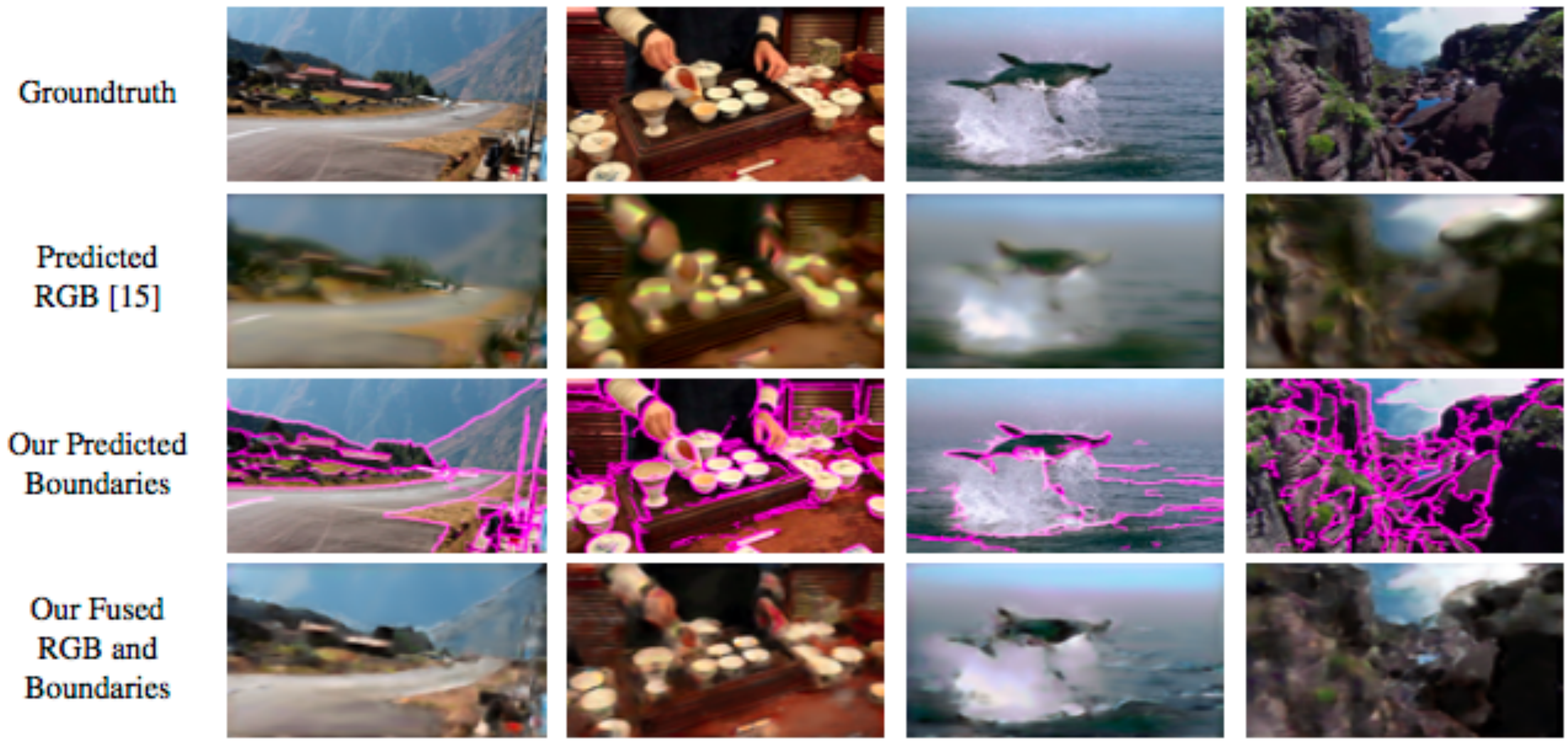
Badrinarayanan et al. 2015
SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling

Computer Vision: Semantic Segmentation



Yang He; Wei-Chen Chiu; Margret Keuper; Mario Fritz
STD2P: RGBD Semantic Segmentation Using Spatio-Temporal Data-Driven Pooling
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, (to appear)

“Predicting the Future”



Apratim Bhattacharyya; Mateusz Malinowski; Bernt Schiele; Mario Fritz
Long-Term Image Boundary Extrapolation
arXiv:1611.08841 [cs.CV], 2016.

Computer Vision & NLP: Image Captioning

Ours: a person on skis jumping over a ramp

Ours: a skier is making a turn on a course

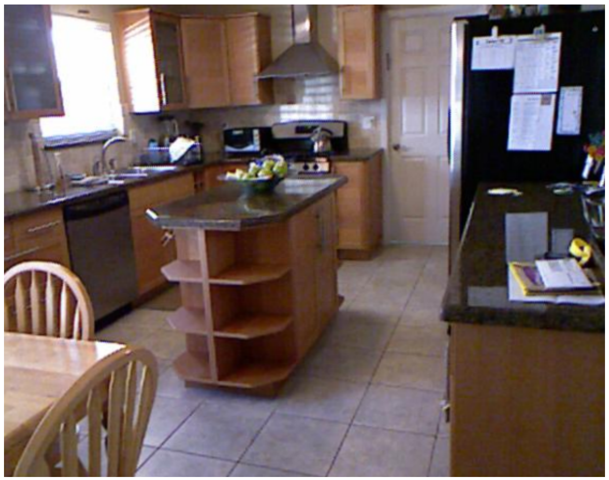
Ours: a cross country skier makes his way through the snow

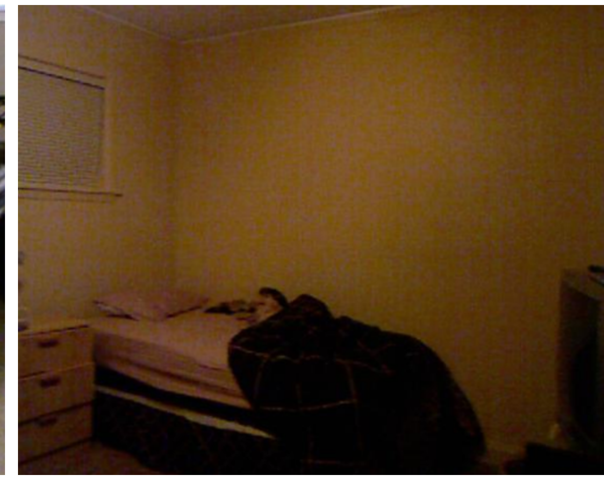
Ours: a skier is headed down a steep slope

Baseline: a man riding skis down a snow covered slope

Rakshith Shetty; Marcus Rohrbach; Lisa Anne Hendricks; Mario Fritz; Bernt Schiele
Speaking the Same Language: Matching Machine to Human Captions by Adversarial Training
arXiv:1703.10476 [cs.CV], 2017

Computer Vision + NLP: Visual Turing Test





What is on the refrigerator?

magnet, paper

What is the color of the comforter?

blue, white

How many drawers are there?

3

Mateusz Malinowski; Mario Fritz
A Multi-World Approach to Question Answering about Real-World Scenes based on Uncertain Input
Neural Information Processing Systems (NIPS), 2014.

Mateusz Malinowski; Marcus Rohrbach; Mario Fritz
Ask Your Neurons: A Neural-based Approach to Answering Questions about Images
IEEE International Conference on Computer Vision (ICCV), 2015,

Graphics

e.g.
Konstantinos Rematas; Tobias Ritschel; Mario Fritz; Efstratios Gavves; Tinne Tuytelaars
Deep Reflectance Maps
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

Graphics

e.g.
Konstantinos Rematas; Tobias Ritschel; Mario Fritz; Efstratios Gavves; Tinne Tuytelaars
Deep Reflectance Maps
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

AI and Planning

Google Official Blog
Insights from Googlers into our products, technology, and the Google culture

AlphaGo: using machine learning to master the ancient game of Go

January 27, 2016

The game of Go originated in China more than 2,500 years ago. Confucius wrote about the game, and it is considered one of the four essential arts required of any true Chinese scholar. Played by more than 40 million people worldwide, the rules of the game are simple: Players take turns to place black or white stones on a board, trying to capture the opponent's stones or surround empty space to make points of territory. The game is played primarily through intuition and feel, and because of its beauty, subtlety and intellectual depth it has captured the human imagination for centuries.

But as simple as the rules are, Go is a game of profound complexity. There are 1,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000 possible positions—that's more than the number of atoms in the universe, and more than a googol times larger than chess.

AI and Planning

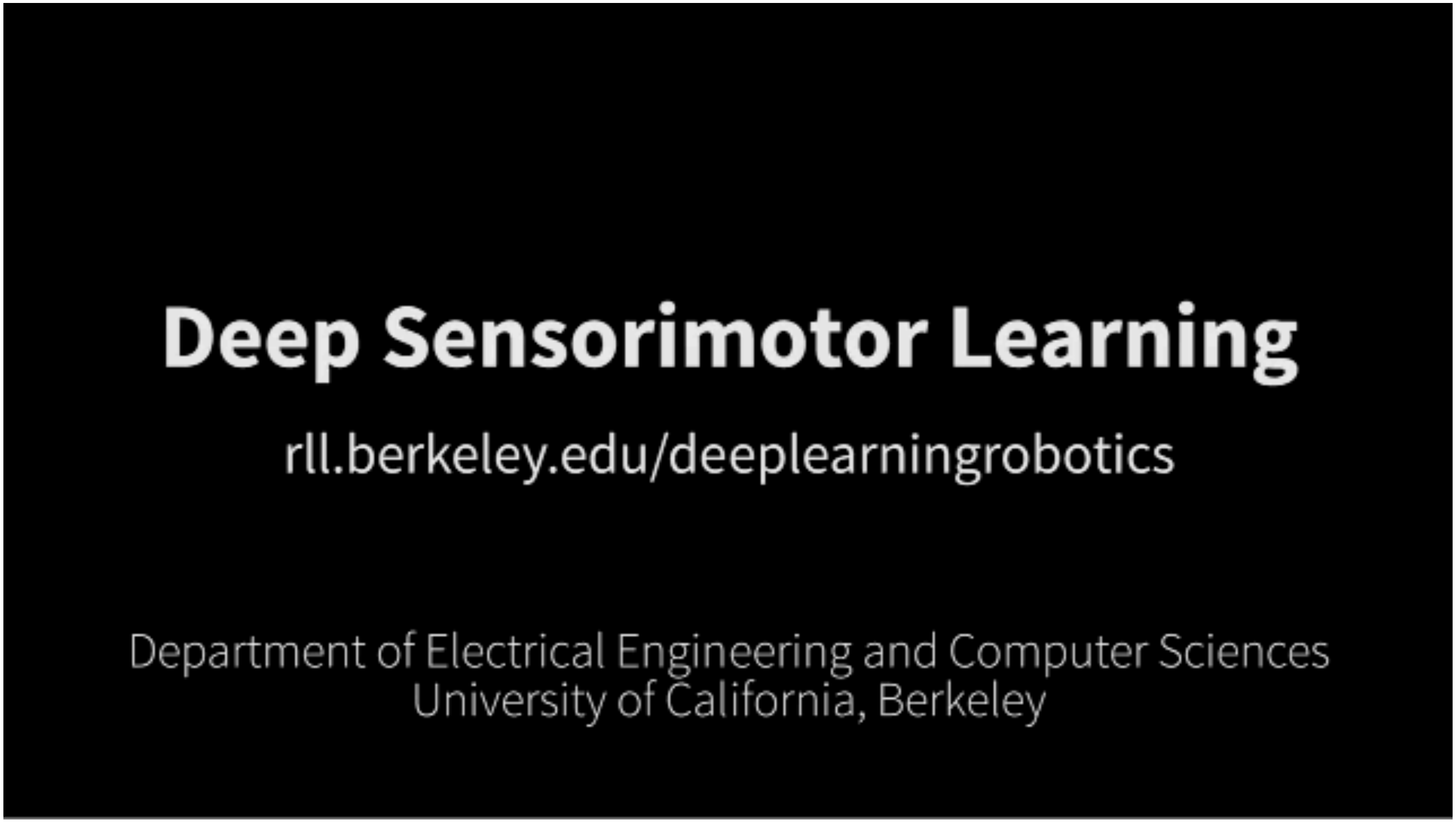
Google Official Blog
Insights from Googlers into our products, technology, and the Google culture

AlphaGo: using machine learning to master the ancient game of Go
January 27, 2016

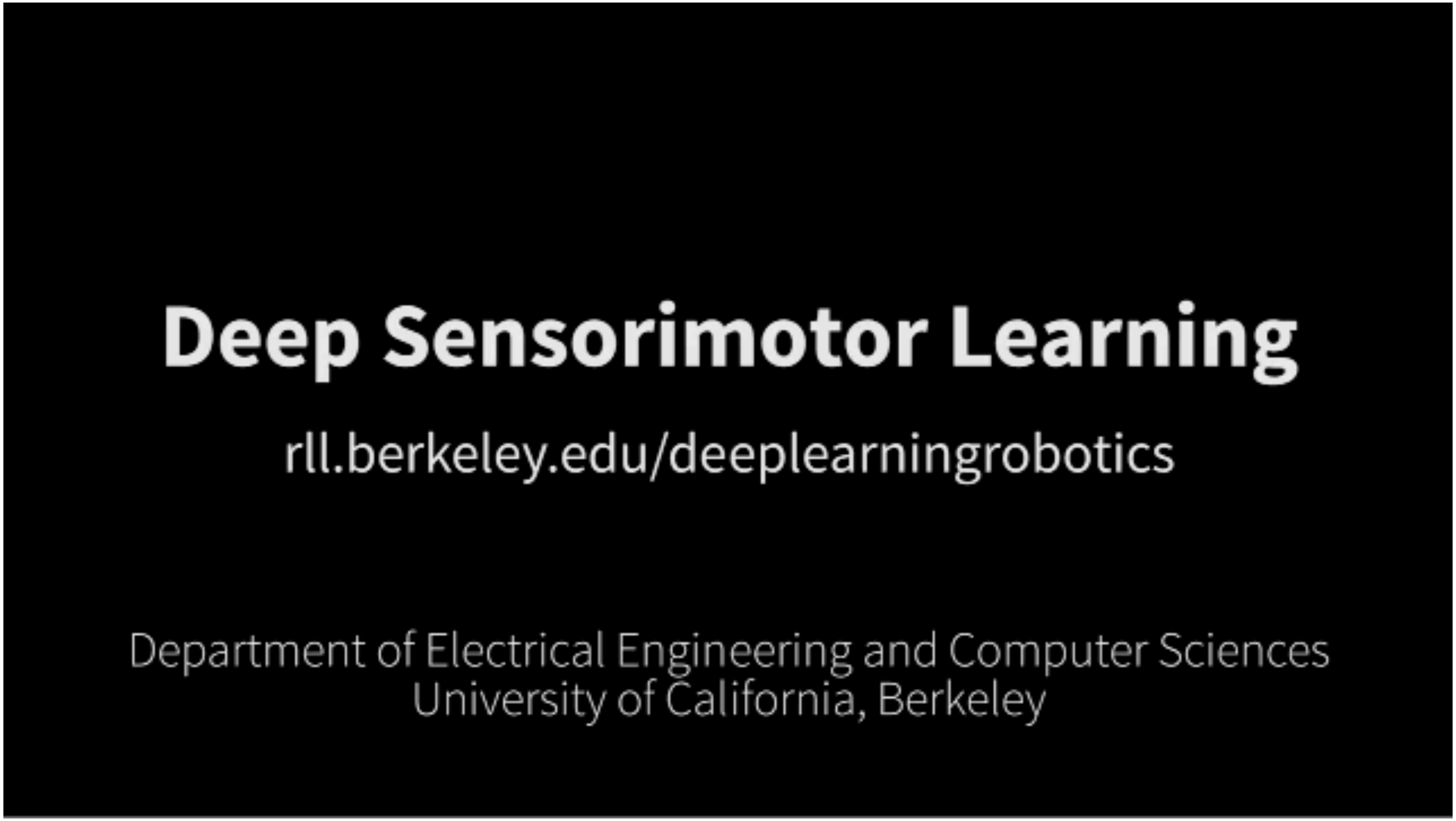
The game of Go originated in China more than 2,500 years ago. Confucius wrote about the game, and it is considered one of the four essential arts required of any true Chinese scholar. Played by more than 40 million people worldwide, the rules of the game are simple: Players take turns to place black or white stones on a board, trying to capture the opponent's stones or surround empty space to make points of territory. The game is played primarily through intuition and feel, and because of its beauty, subtlety and intellectual depth it has captured the human imagination for centuries.

But as simple as the rules are, Go is a game of profound complexity. There are 1,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000 possible positions—that's more than the number of atoms in the universe, and more than a googol times larger than chess.

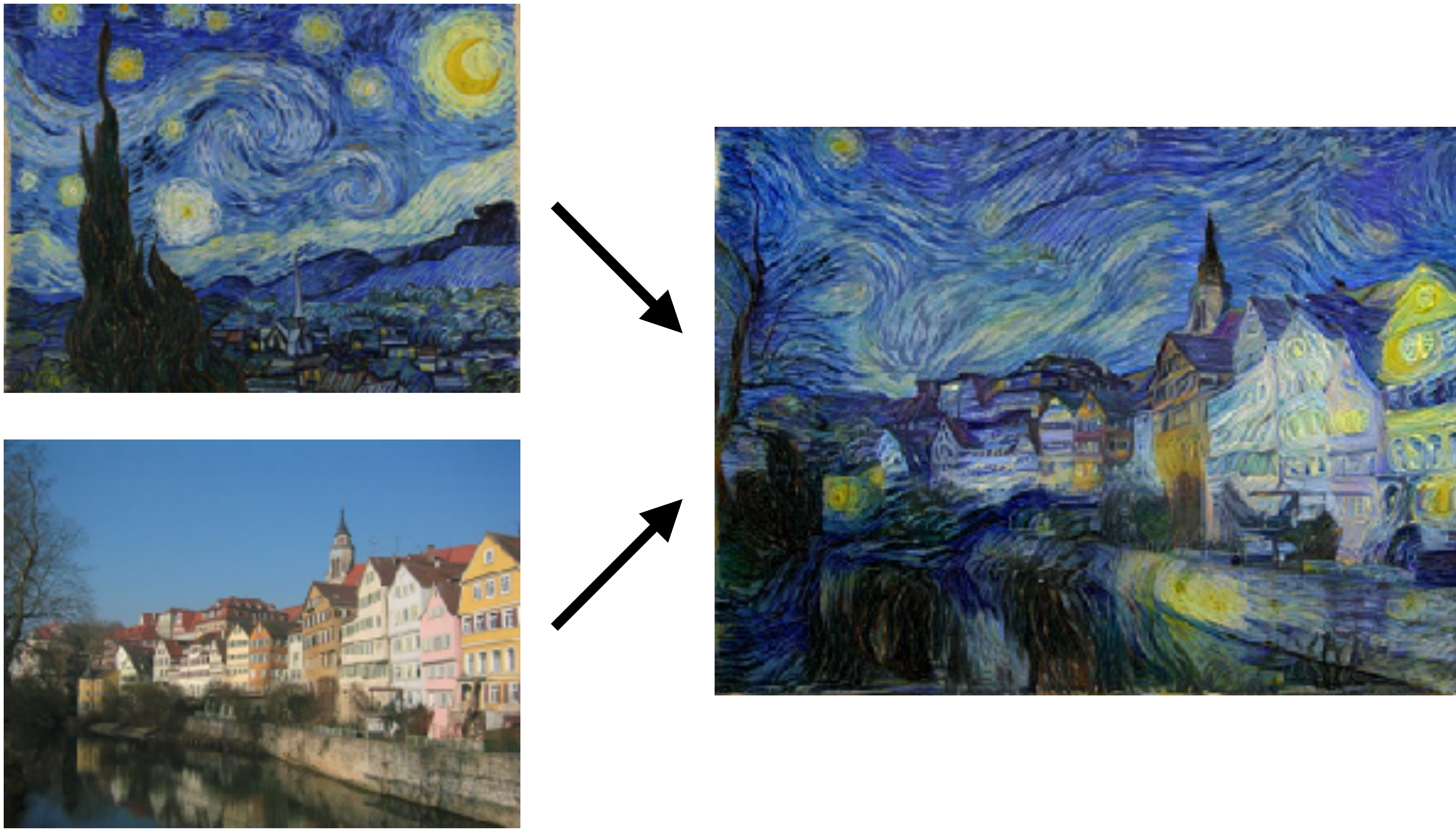
Robotics


Deep Sensorimotor Learning
rll.berkeley.edu/deeplearningrobotics
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Robotics

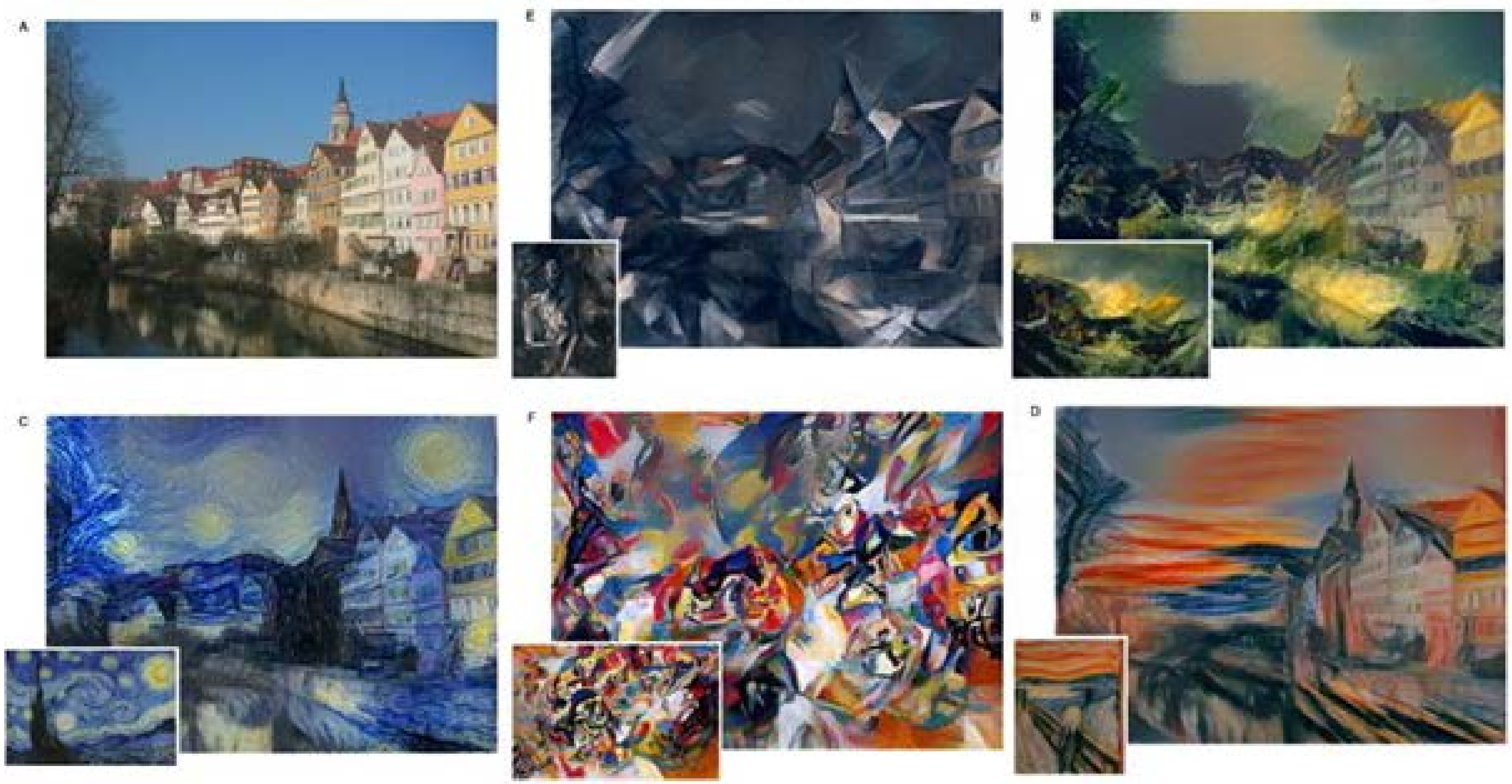

Deep Sensorimotor Learning
rll.berkeley.edu/deeplearningrobotics
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Creative Aspects



deepart.io

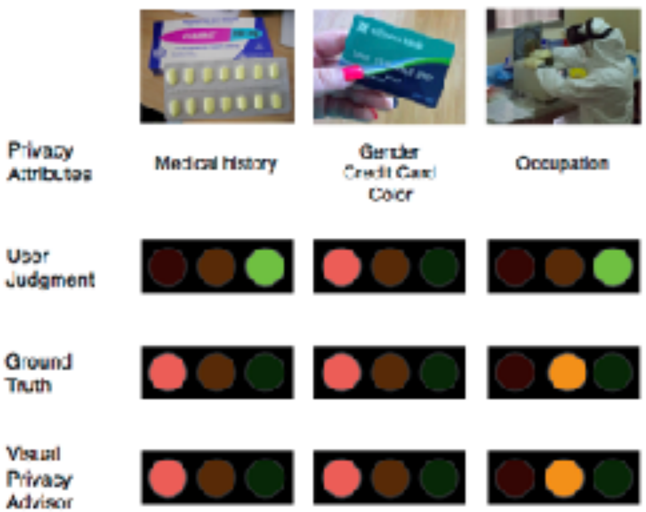
Imitating famous painters

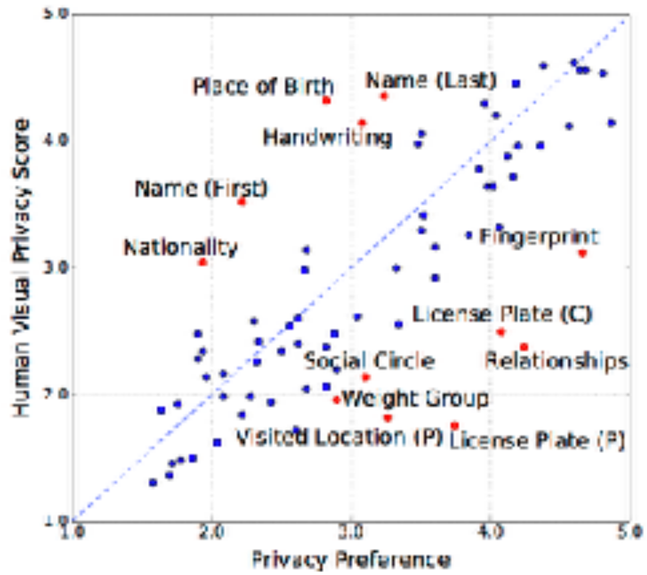


Music

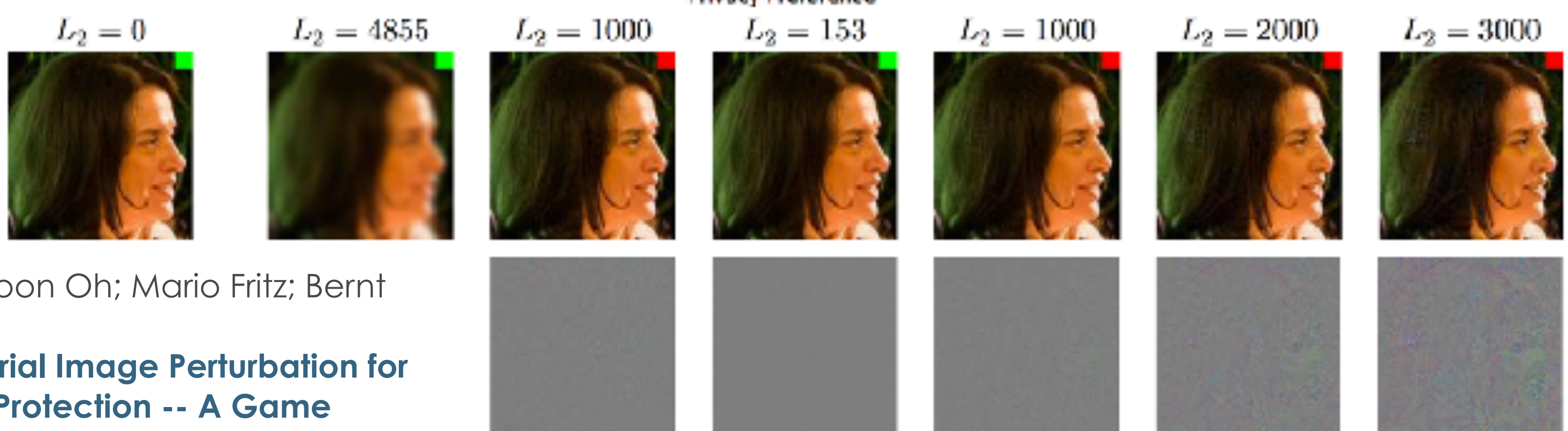
- bachbot.com

Privacy & Safety





Tribhuvanesh Orekondy; Bernt Schiele; Mario Fritz
Towards a Visual Privacy Advisor: Understanding and Predicting Privacy Risks in Images
arXiv:1703.10660 [cs.CV], 2017.



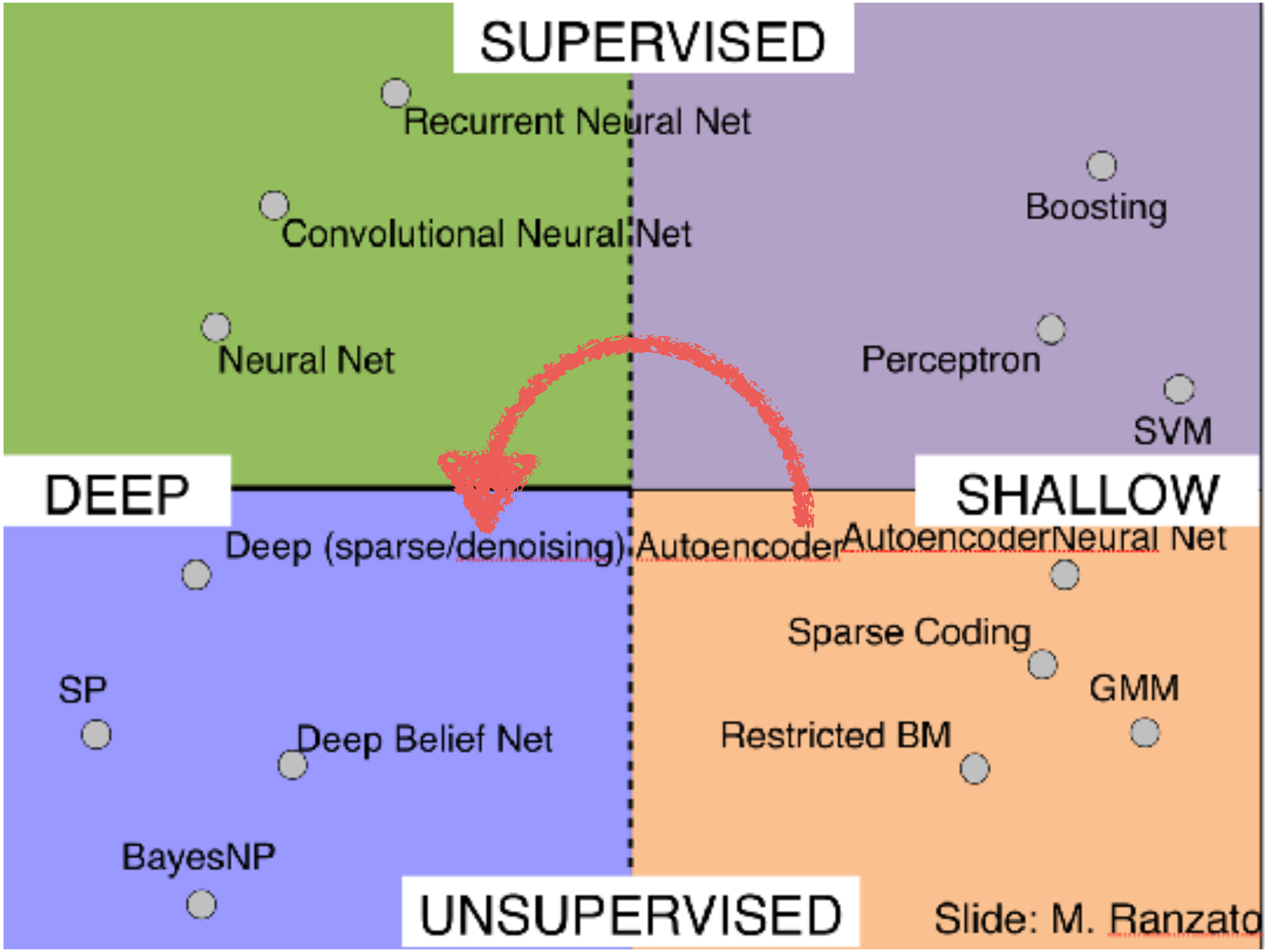
Seong Joon Oh; Mario Fritz; Bernt Schiele
Adversarial Image Perturbation for Privacy Protection -- A Game Theory Perspective
arXiv:1703.09471 [cs.CV], 2017.

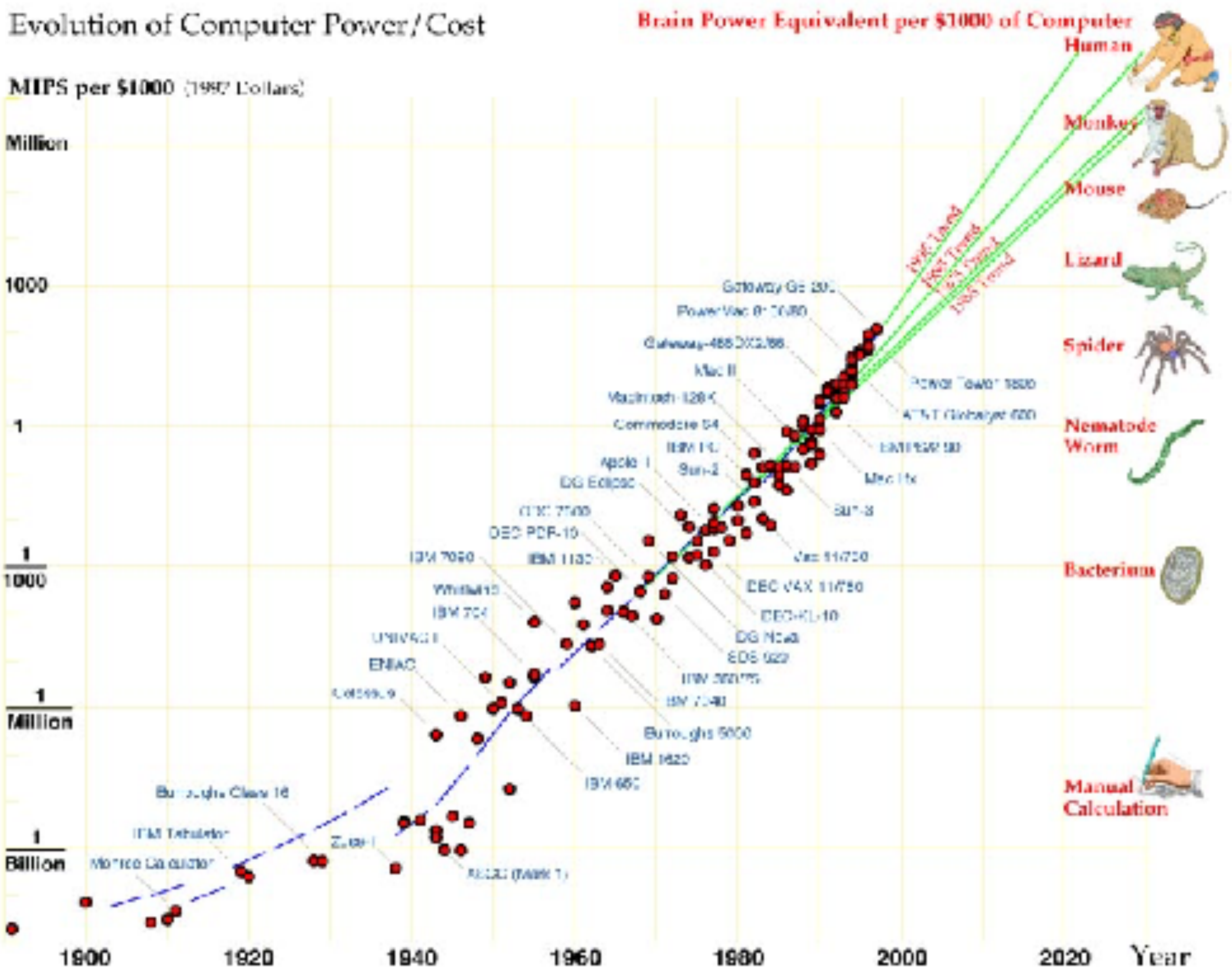
Deep Learning

- Deep Learning is based on
  - Availability of large datasets
  - Massive parallel compute power
  - Machine learning

- Strong improvements due to
  - Internet
  - GPUs
  - Hierarchical models with end-to-end learning

Overview of Deep Learning





Evolution of Computer Power/Cost
MIPS per $1000 (1997 Dollars)
Brain Power Equivalent per $1000 of Computer
Human
Monkey
Mouse
Lizard
Spider
Nematode Worm
Bacterium
Manual Calculation
Million
1000
1
1/1000
1/Million
1/Billion
1900
1920
1940
1960
1980
2000
2020
Year
ENIAC
IBM 704
IBM 7090
DEC PDP-10
Apple II
DG Eclipse
Sun-2
Commodore 64
Mac II
Sun-3
DEC VAX 11/750
DEC-KL-10
IBM 1130
IBM 1620
IBM 650
Burroughs Class 16
IBM Tabulator
Monroe Calculator
Power Tower 180e

Traditional Approach

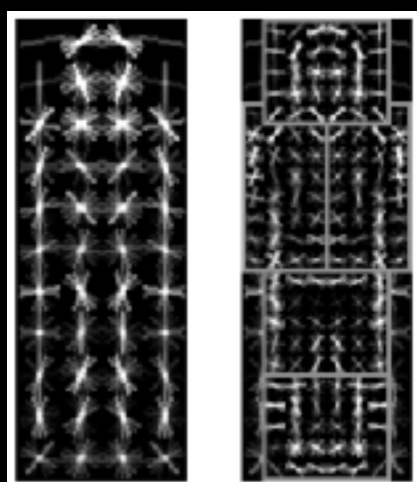
 → feature extraction (hand crafted) → Classification → **Car**

x $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ y

- Fixed feature extraction
  - too general
  - too specific
  - often not task specific
- How to increase capacity of leaning algorithms?
  - linear?
  - kernalized / lifted?

Motivation

- Features are key to recent progress in recognition
- Multitude of hand-designed features currently in use
  - SIFT, HOG, LBP, MSER, Color-SIFT………….
- Where next? Better classifiers? Or keep building more features?



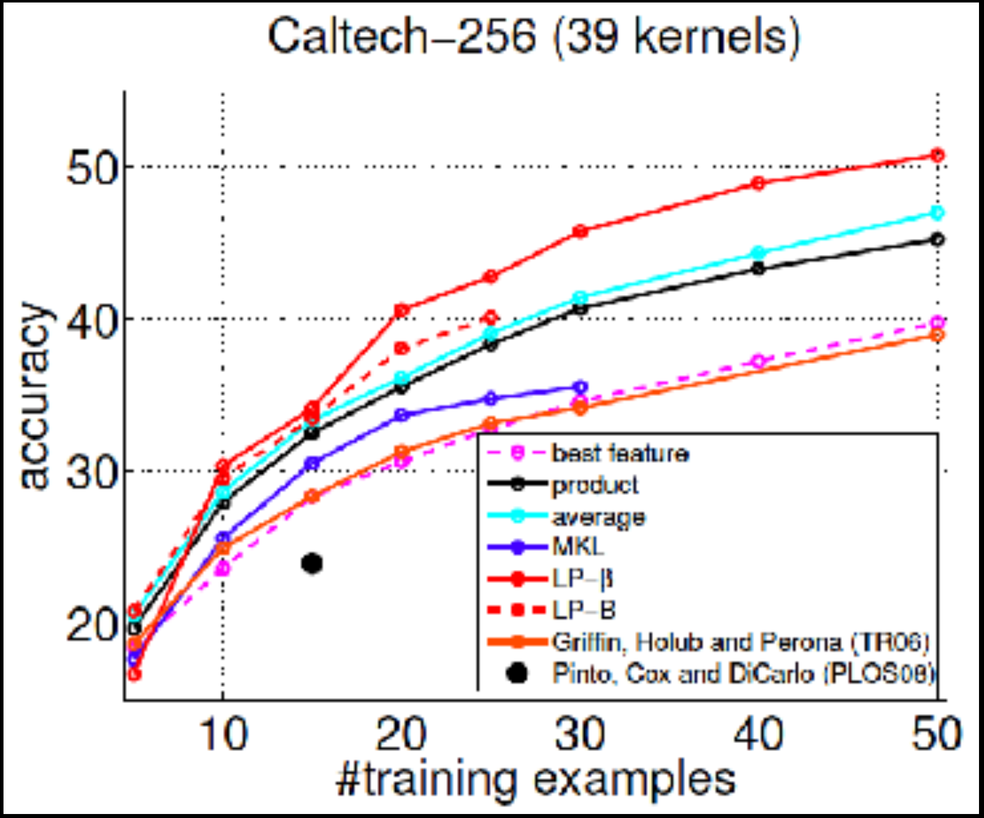
Felzenszwalb, Girshick, McAllester and Ramanan, PAMI 2007

Yan & Huang
(Winner of PASCAL 2010 classification competition)

Hand-Crafted Features

- LP-β Multiple Kernel Learning (MKL)
  - Gehler and Nowozin, On Feature Combination for Multiclass Object Classification, ICCV'09
- 39 different kernels
  - PHOG, SIFT, V1S+, Region Cov. Etc.
- MKL only gets few % gain over averaging features

→ Features are doing the work



Deep Learning: Trainable features

$\rightarrow$ Features $\text{Merkmale} = g(x, \lambda)$ $\rightarrow$ Classifier $y = f(\text{Merkmale}, \omega)$ $\rightarrow$ **Car** y

x

- Parameterized feature extraction
- Features should be
  - efficient to compute
  - efficient to train (differentiable)

Deep Learning: Joint Training of all Parameters

$\rightarrow$ Features $\text{Merkmale} = g(x, \lambda)$ $\rightarrow$ Classifier $y = f(\text{Merkmale}, \omega)$ $\rightarrow$ **Car** y

x

"End-to-End" System

- Parameterized feature extraction
- Features should be
  - efficient to compute
  - efficient to train (differe
- Joint training of feature extraction and classification
- Feature extraction and classification merge into one pipeline

Deep Learning: Joint Training of all Parameters

x → Features: $\text{Merkmale} = g(x, \lambda)$ → Classifier: $y = f(\text{Merkmale}, \omega)$ → **Car** y

"End-to-End" System

- All parts are adaptive
- No differentiation between feature extraction and classification
- Non linear transformation from input to desired output

Deep Learning: Complex Functions by Composition

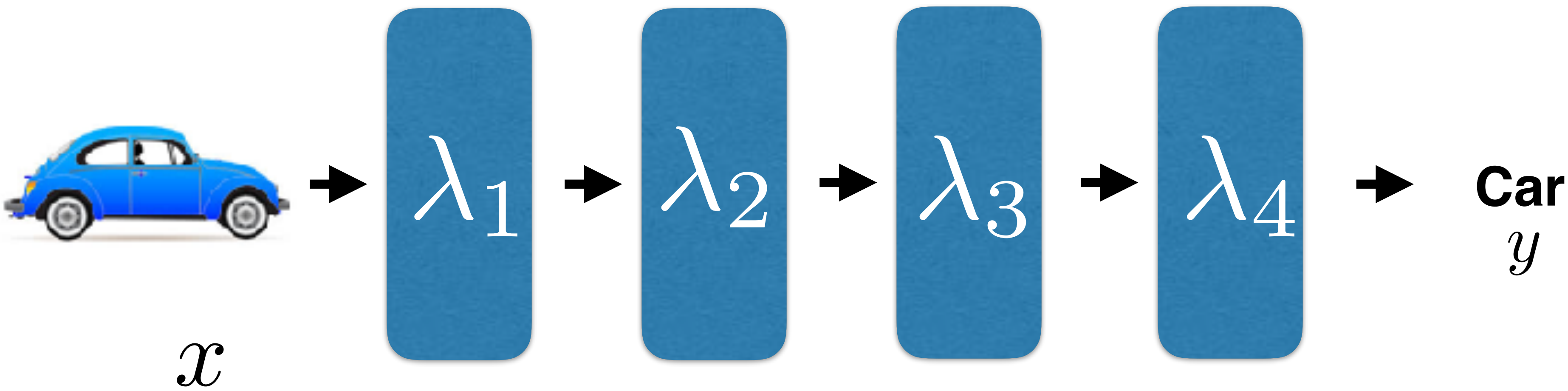
$\rightarrow$ **Features** $\text{Merkmale} = g(x, \lambda)$ $\rightarrow$ **Classifier** $y = f(\text{Merkmale}, \omega)$ $\rightarrow$ **Car** y

x

"End-to-End" System

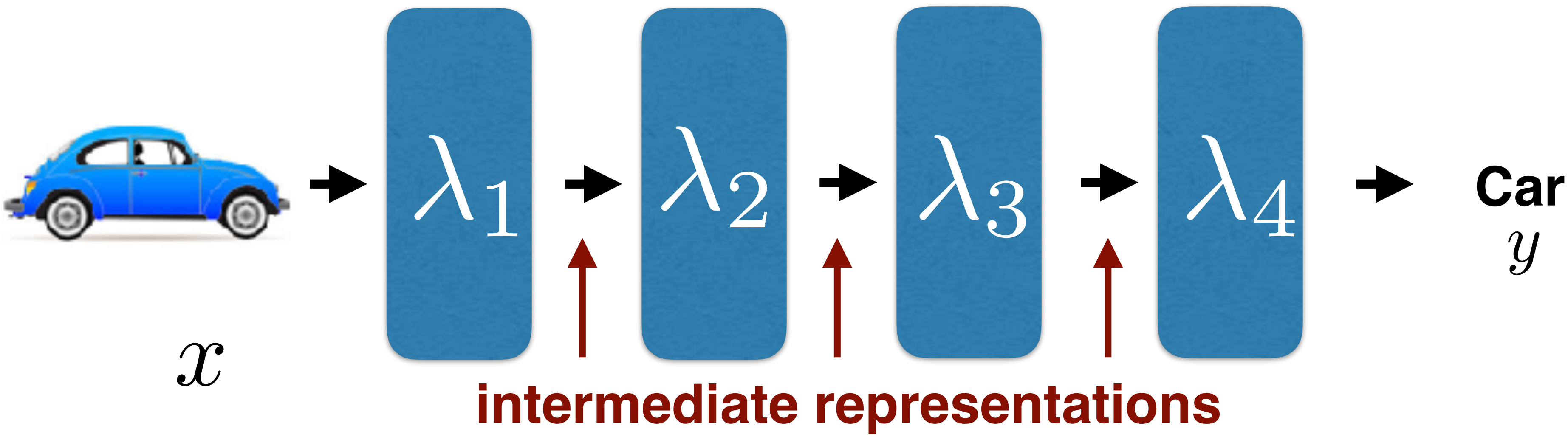
- How can we build such systems?
- What is the parameterisation (hypothesis)?
- Composition of simple building blocks can lead to complex systems (e.g. neurons - brain)

Deep Learning: Complex Functions by Composition



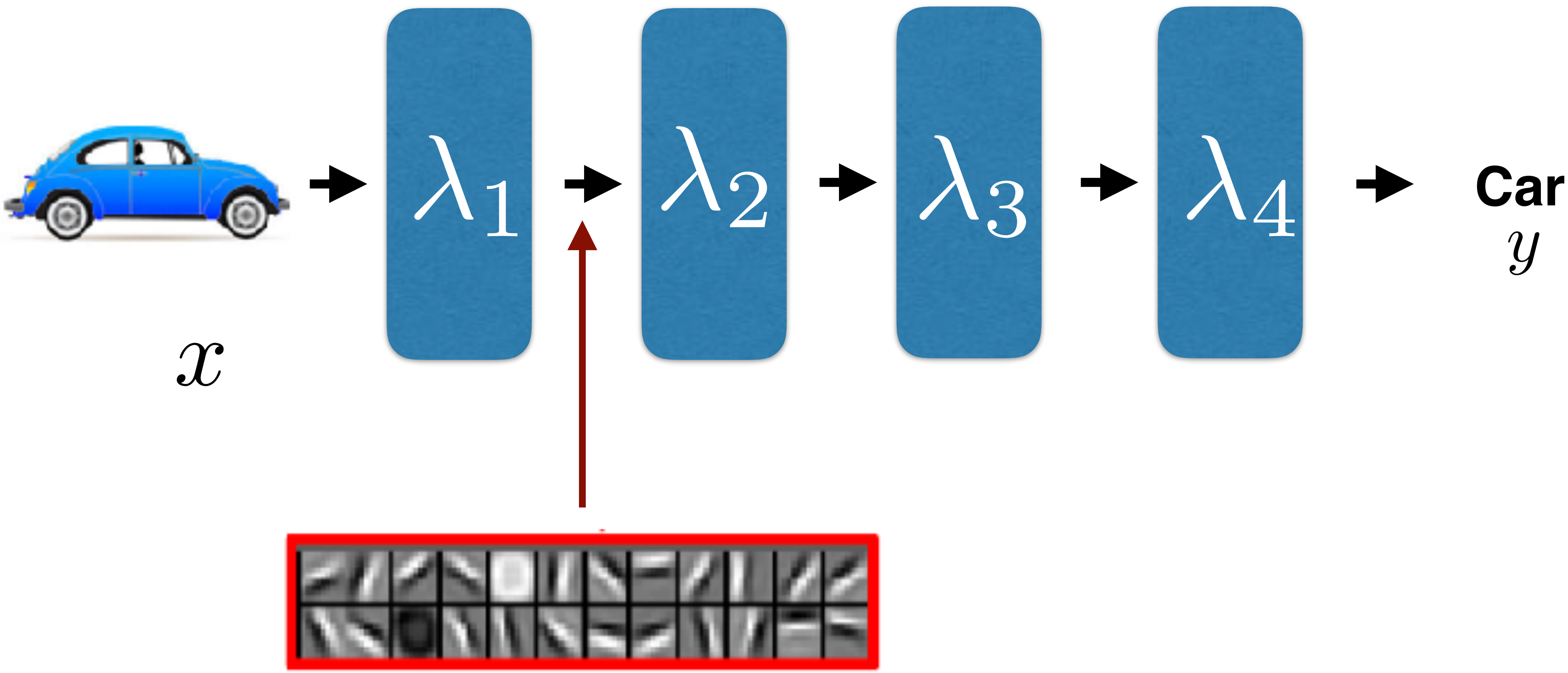
- How can we build such systems?
- What is the parameterisation (hypothesis)?
- Composition of simple building blocks can lead to complex systems (e.g. neurons - brain)Jeder Block hat trainierbare Parameter
- Each block has trainable parameters λ_i

Deep Learning: Complex Functions by Composition



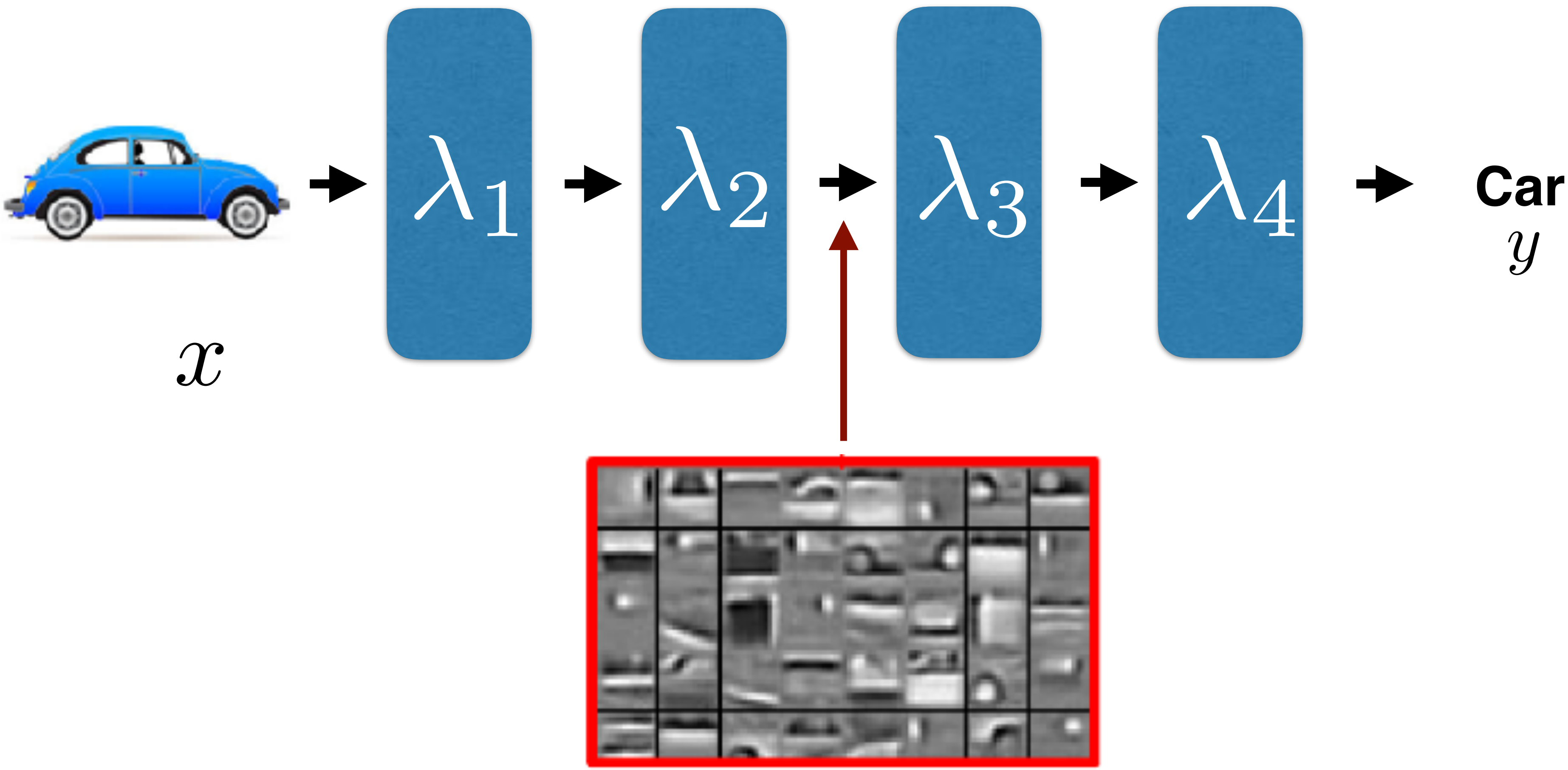
- How can we build such systems?
- What is the parameterisation (hypothesis)?
- Composition of simple building blocks can lead to complex systems (e.g. neurons - brain)Jeder Block hat trainierbare Parameter
- Each block has trainable parameters λ_i

Deep Learning: Complex Functions by Composition



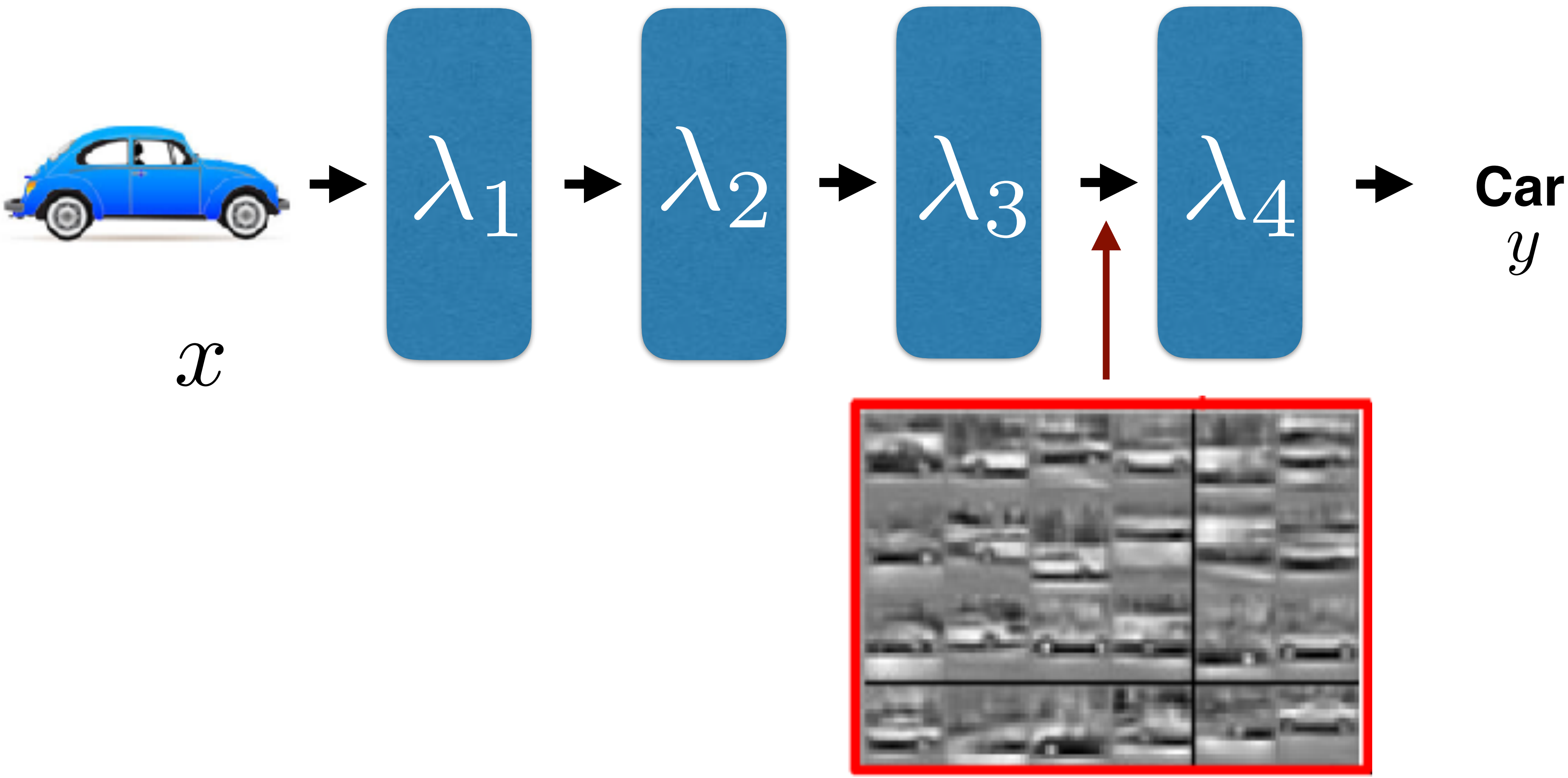
Lee et al. “Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations”

Deep Learning: Complex Functions by Composition



Lee et al. “Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations”

Deep Learning: Complex Functions by Composition



Lee et al. "Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations"

Summary of Main Ideas in Deep Learning

1. Learning of feature extraction
2. Efficient and trainable systems by differentiable building blocks
3. Composition of deep architectures via non-linear modules
4. “End-to-End” training: no differentiation between feature extraction and classification

Summary of Main Ideas in Deep Learning

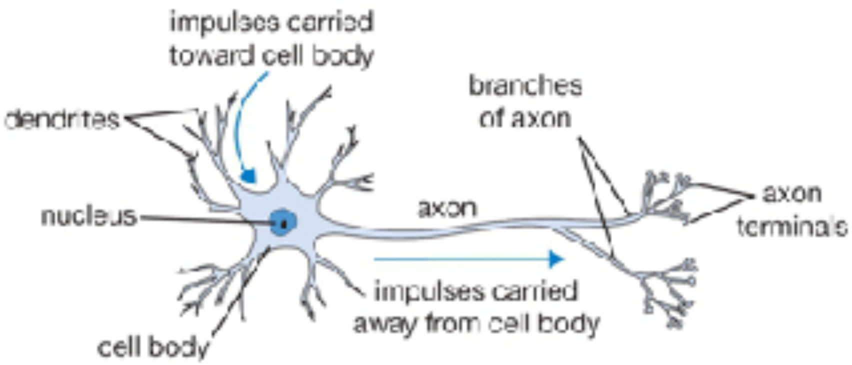
1. Learning of feature extraction

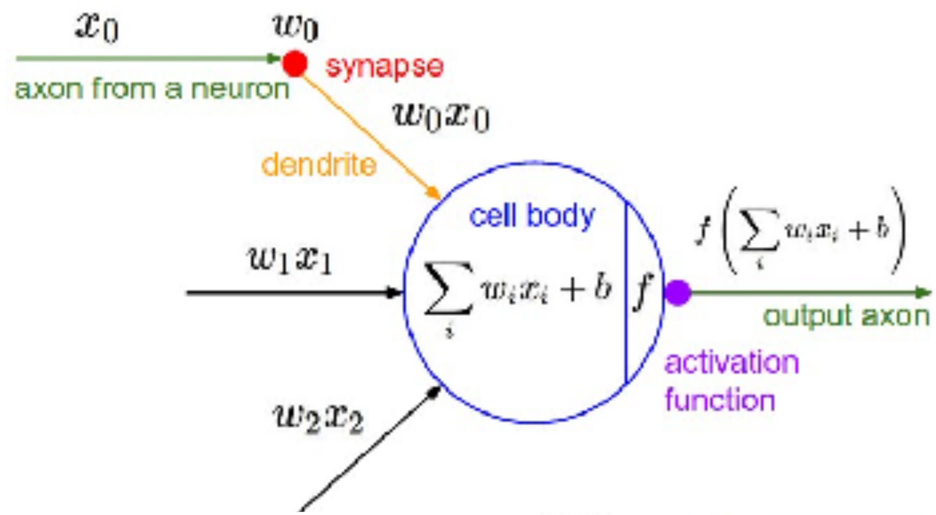
2. Efficient and trainable systems by **differentiable** building blocks

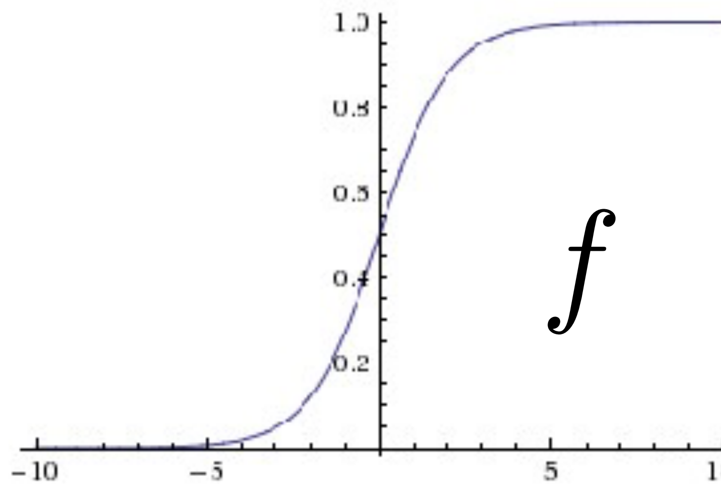
3. Composition of deep architectures via **non-linear modules**

4. “End-to-End” training: no differentiation between feature extraction and classification

Some Inspiration from the Brain







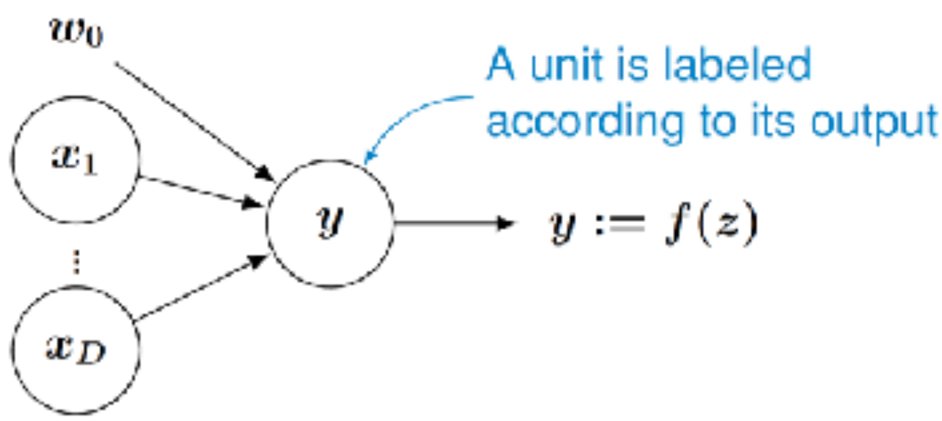
- Motivation from the brain
  - effective
  - universal learning machine?
- Properties we want:
  - differentiable
  - efficient
- Highly simplified neural model
  - Combination of inputs is linear
  - No spike
  - No temporal model
  - Fixed topology

[Reading - Chapter 5.1 - 5.3 @ Bishop 2006]

Short Intro: "Standard" Neural Networks

Core component of a neural network: *processing unit* = neuron of the human brain.

A processing unit maps multiple input values onto one output value y:



- $x_1, \ldots, x_D$ are inputs, e.g. from other processing units within the network.
- w_0 is an external input called *bias*.
- The *propagation rule* maps all input values onto the actual input z.
- The *activation function* is applied to obtain $y = f(z)$.

slide taken from David Stutz (Aachen)

Short Intro: Perceptron

Introduced by Rosenblatt in [Rosenblatt 58].

The (single-layer) *perceptron* consists of D input units and C output units.

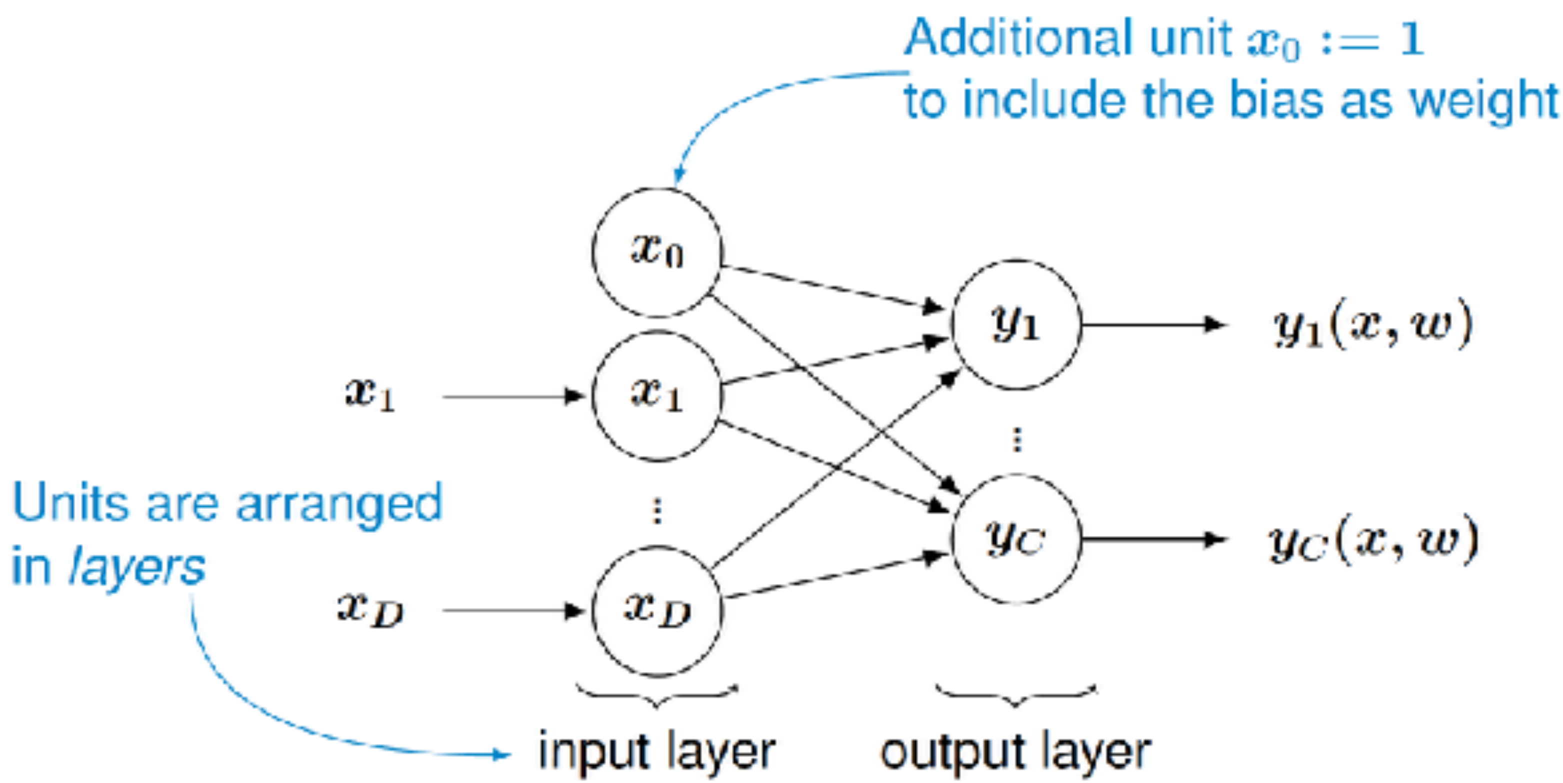
- Propagation rule: weighted sum over inputs x_i with weights w_{ij}.
- Input unit i: single input value $z = x_i$ and identity activation function.
- Output unit j calculates the output

$$y_j(x, w) = f(z_j) = f\left(\sum_{k=1}^{D} w_{jk} x_k + w_{j0}\right) \overset{x_0 := 1}{=} f\left(\sum_{k=0}^{D} w_{jk} x_k\right).$$

propagation rule with additional bias w_{j0}

slide taken from David Stutz (Aachen)

Short Intro: Perceptron



slide taken from David Stutz (Aachen)

Short Intro: Perceptron - Activation Functions

Used propagation rule: weighted sum over all inputs.

How to choose the activation function $f(z)$?

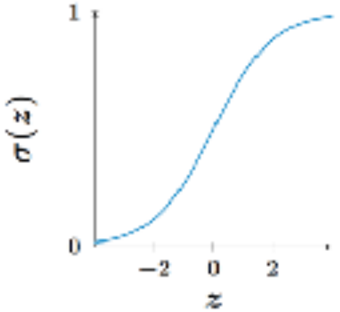
- Heaviside function $h(z)$ models the electrical impulse of neurons in the human brain:

$$h(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}.$$

In general we prefer monotonic, differentiable activation functions.

- Logistic sigmoid $\sigma(z)$ as differentiable version of the Heaviside function:

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



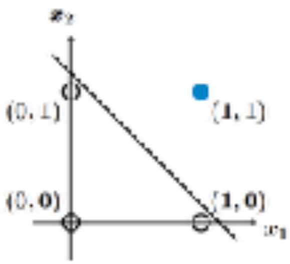
- Or its extension for multiple output units, the softmax activation function:

$$\sigma(z, i) = \frac{\exp(z_i)}{\sum_{k=1}^{C} \exp(z_k)}.$$

Single Layer Perceptron

Which target functions can be modeled using a single-layer perceptron?

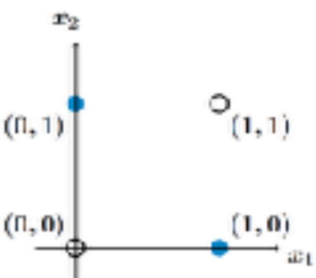
- **A single-layer perceptron represents a hyperplane in multidimensional space.**



Modeling boolean AND with target function $g(x_1, x_2) \in \{0, 1\}$.

Problem: How to model boolean exclusive OR (XOR) using a line in two-dimensional space?

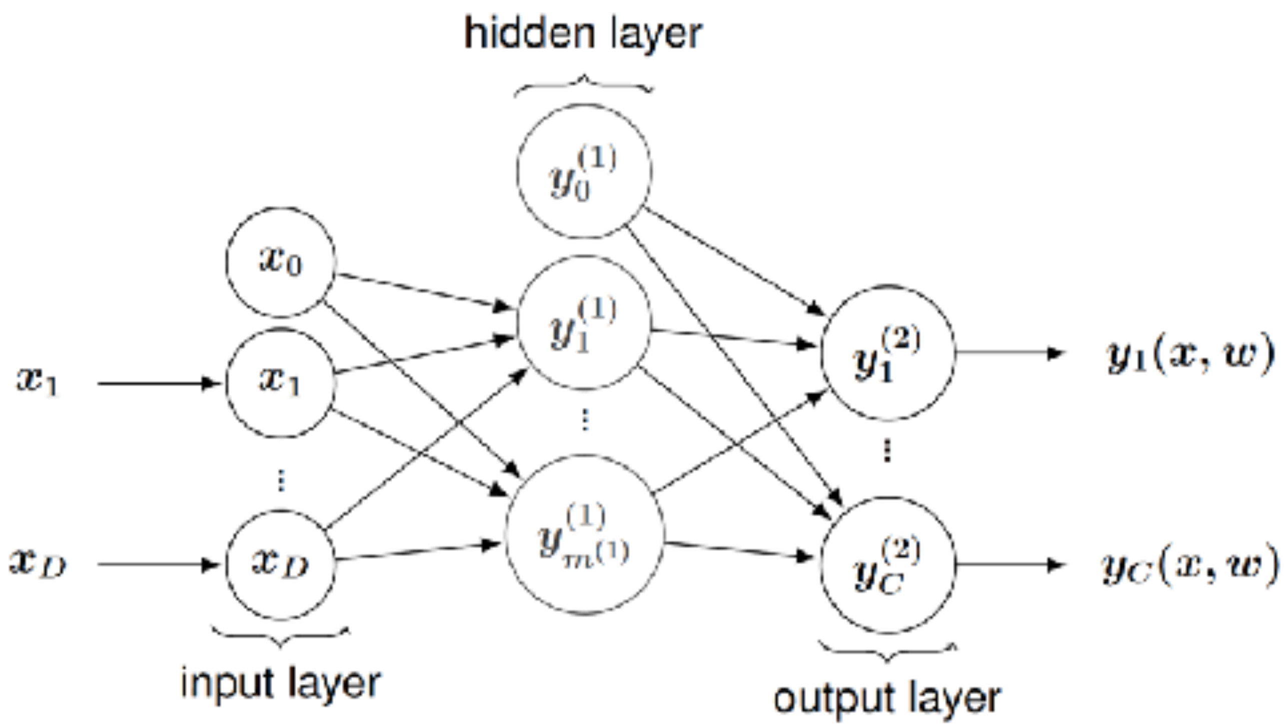
- **Boolean XOR cannot be modeled using a single-layer perceptron.**



Boolean exclusive OR target function.

slide taken from David Stutz (Aachen)

Short Intro: Two-Layer Perceptron



slide taken from David Stutz (Aachen)

Short Intro: Multi-Layer Perceptron (MLP)

Idea: Add additional $L > 0$ *hidden* layers in between the input and output layer.

- $m^{(l)}$ hidden units in layer (l) with $m^{(0)} := D$ and $m^{(L+1)} := C$.
- Hidden unit i in layer l calculates the output

layer

$$y_i^{(l)} = f\left(\sum_{k=0}^{m^{(l-1)}} w_{ik} y_k^{(l-1)}\right).$$

unit

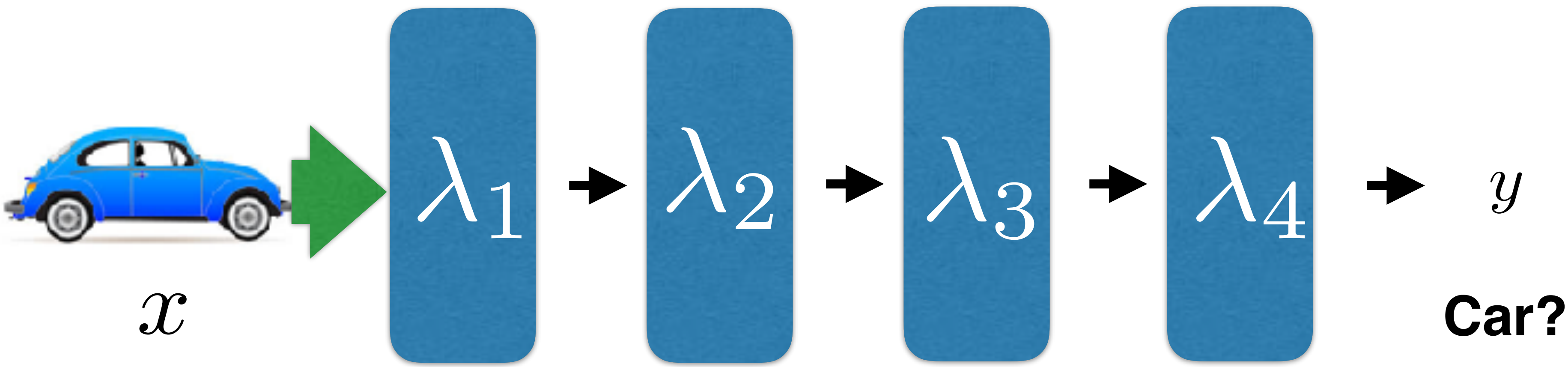
A *multilayer perceptron* models a function

$$y(\cdot, w) : \mathbb{R}^D \mapsto \mathbb{R}^C, x \mapsto y(x, w) = \begin{pmatrix} y_1(x, w) \\ \vdots \\ y_C(x, w) \end{pmatrix} = \begin{pmatrix} y_1^{(L+1)} \\ \vdots \\ y_C^{(L+1)} \end{pmatrix}$$

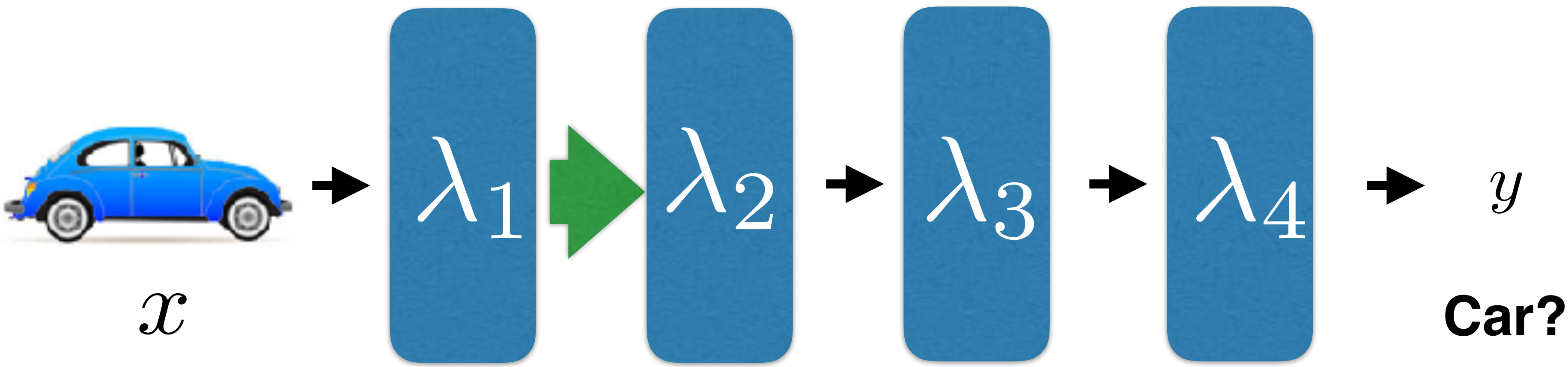
where $y_i^{(L+1)}$ is the output of the i-th output unit.

slide taken from David Stutz (Aachen)

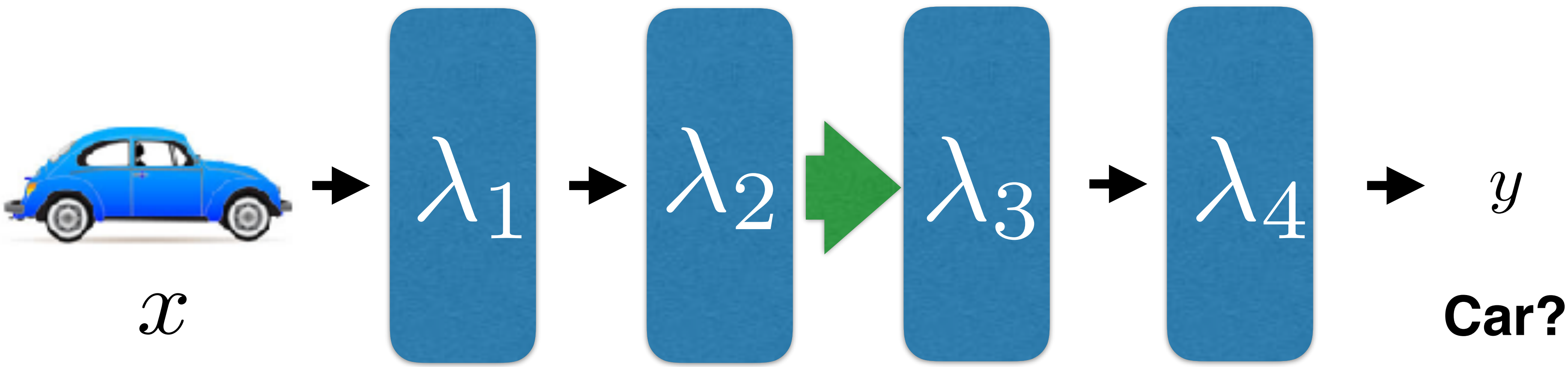
Training: Overview



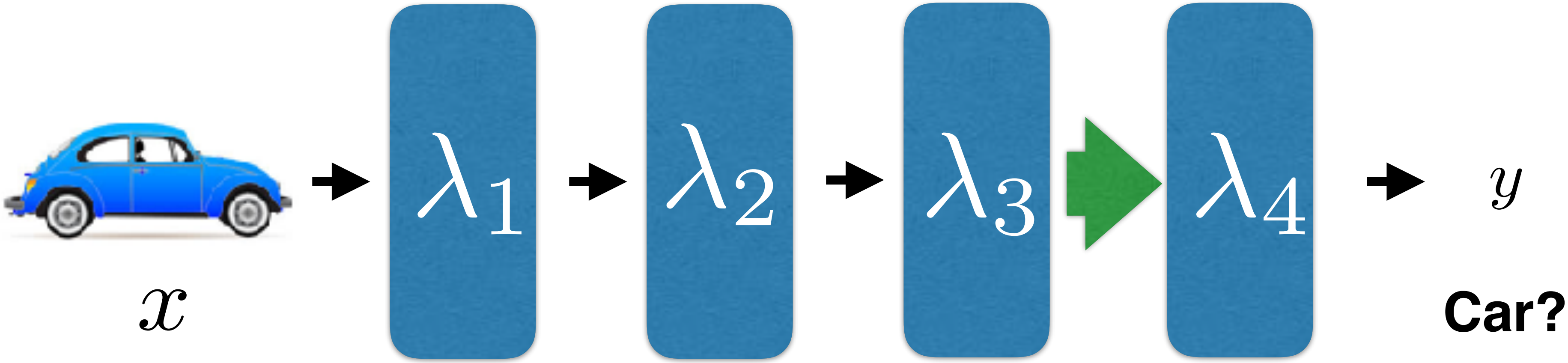
Training: Overview



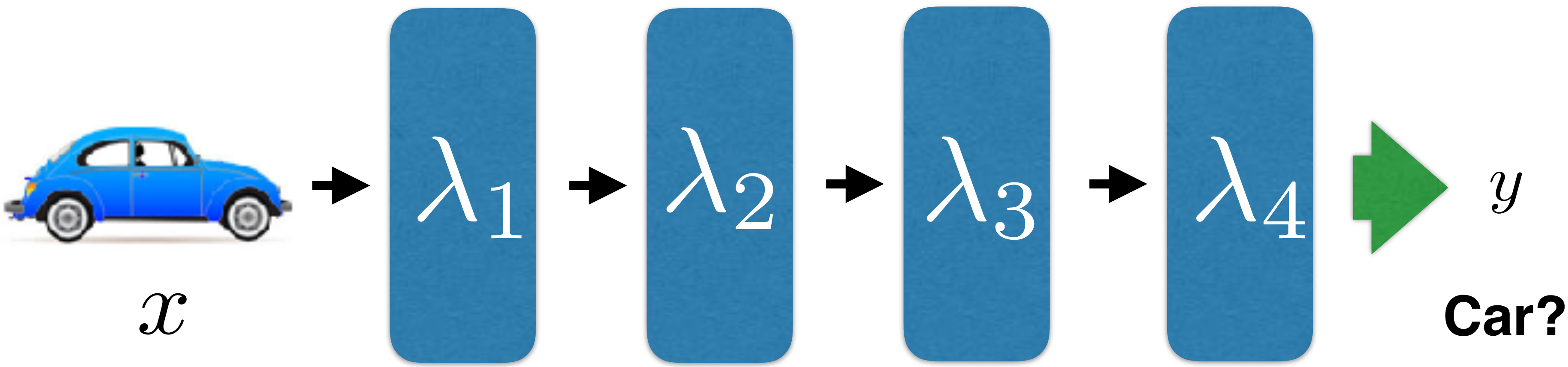
Training: Overview



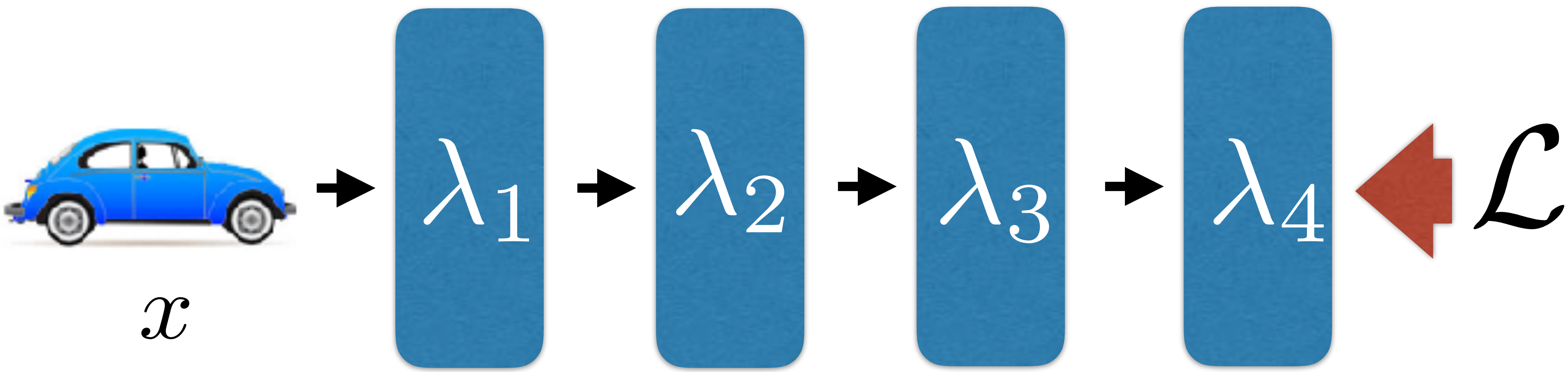
Training: Overview



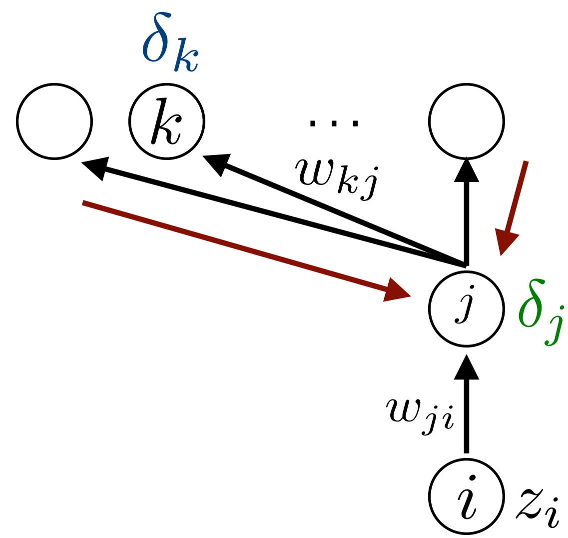
Training: Overview



Training: Overview



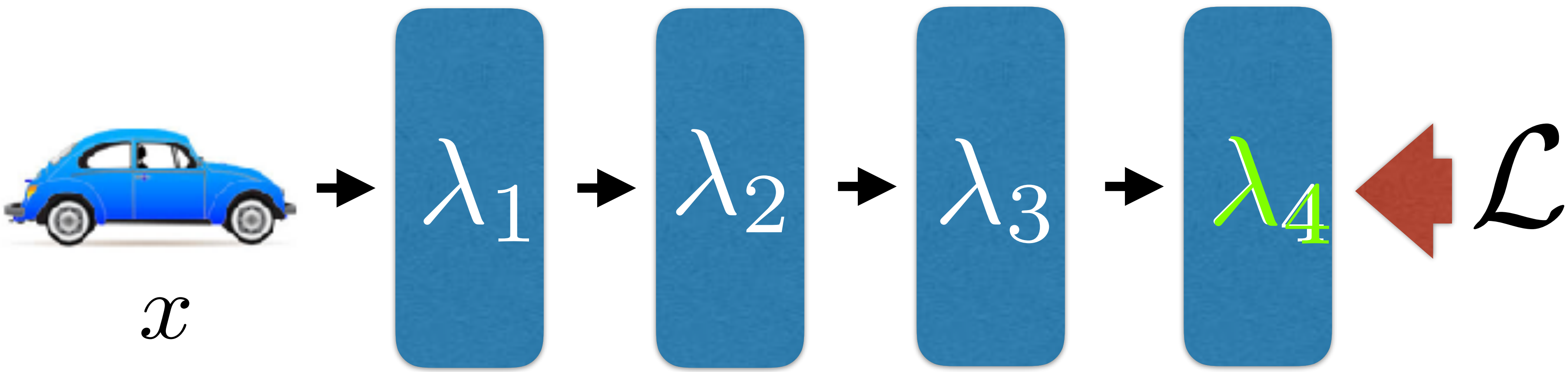
- Use Chain Rule to compute gradient recursively



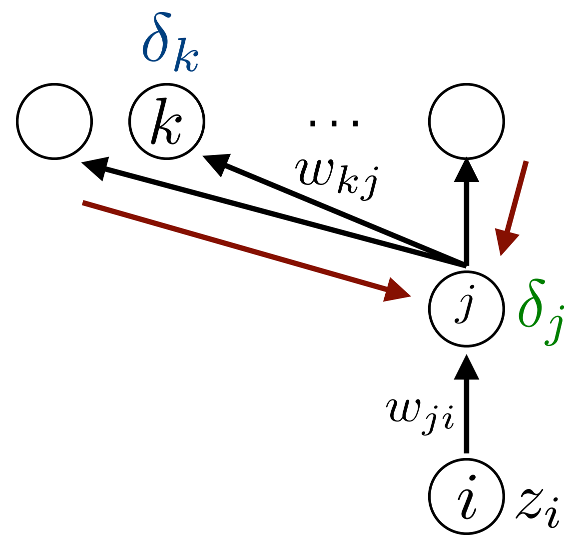
$$\delta_j = f'(a_j) \sum_k w_{kj} \delta_k$$

$$w_{ji}^{\text{neu}} = w_{ji}^{\text{alt}} - \eta \delta_j z_i$$

Training: Overview



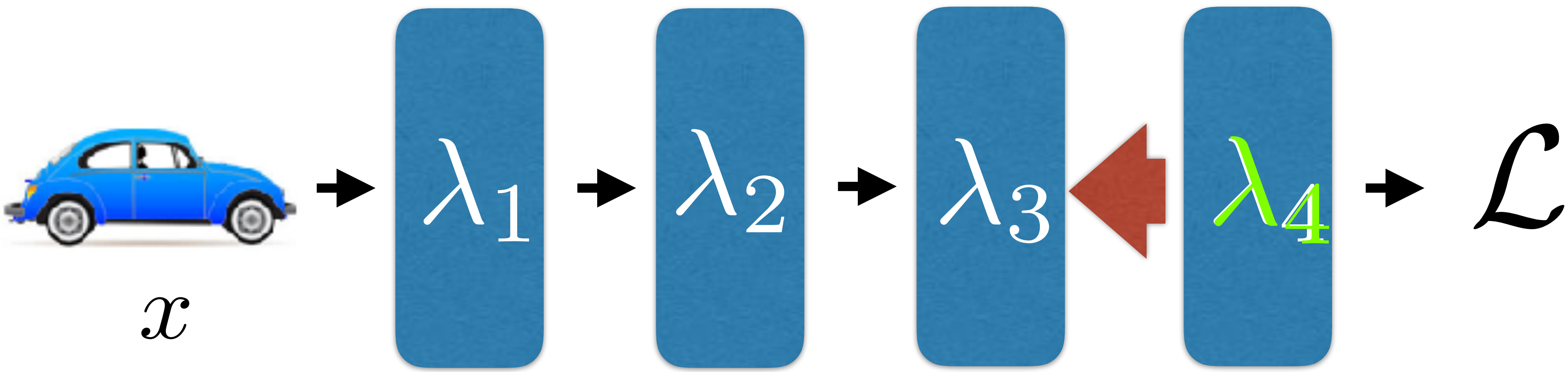
- Use Chain Rule to compute gradient recursively



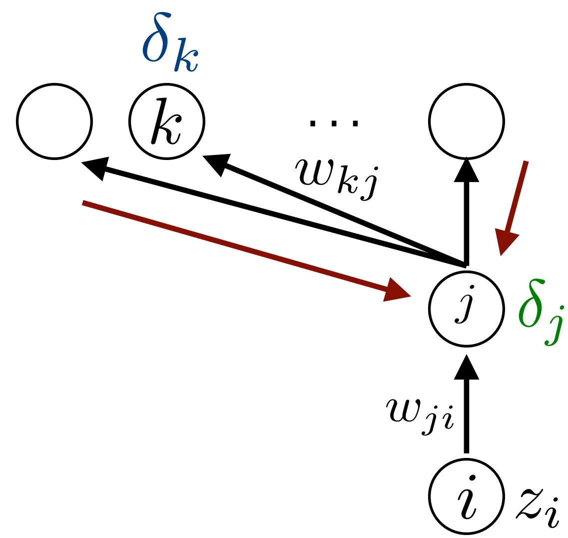
$$\delta_j = f'(a_j) \sum_k w_{kj} \delta_k$$

$$w_{ji}^{\text{neu}} = w_{ji}^{\text{alt}} - \eta \delta_j z_i$$

Training: Overview



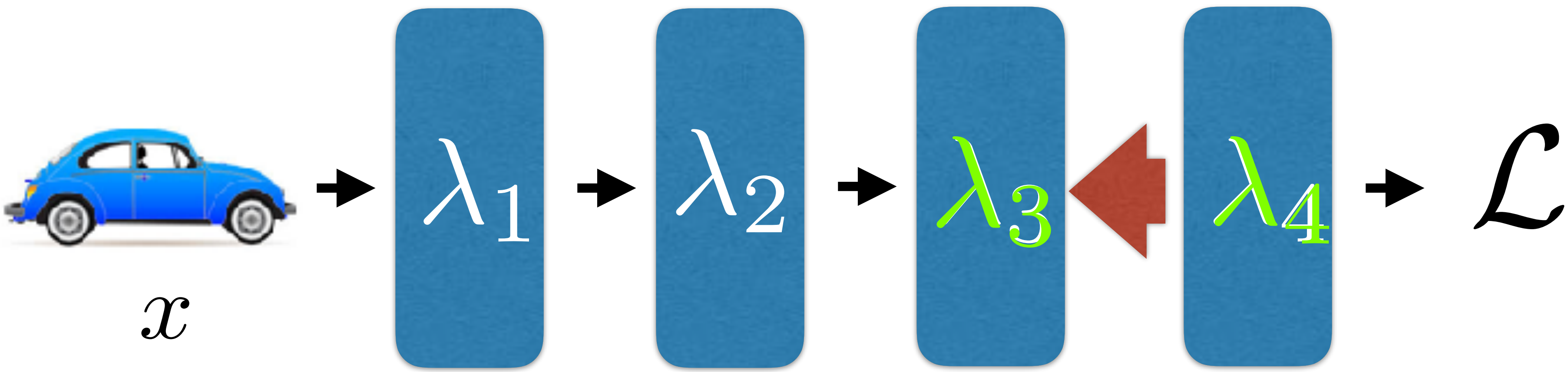
- Use Chain Rule to compute gradient recursively



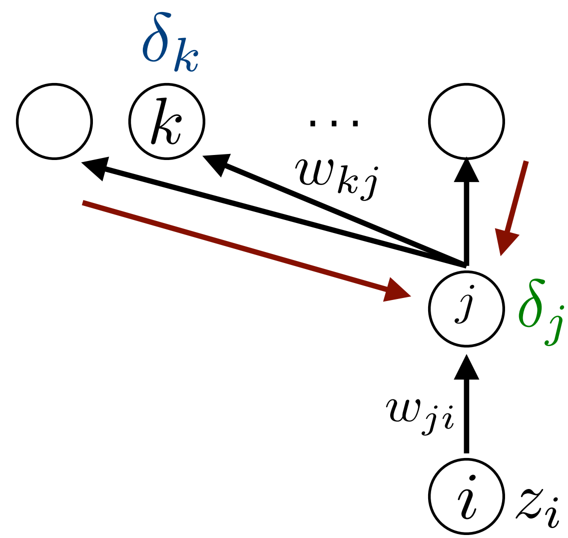
$$\delta_j = f'(a_j) \sum_k w_{kj} \delta_k$$

$$w_{ji}^{\text{neu}} = w_{ji}^{\text{alt}} - \eta \delta_j z_i$$

Training: Overview



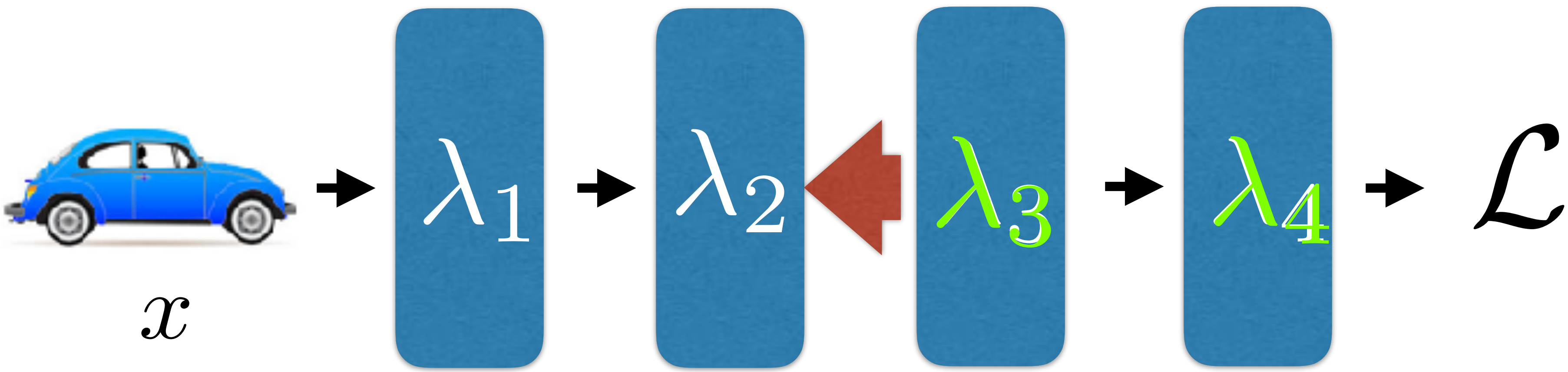
- Use Chain Rule to compute gradient recursively



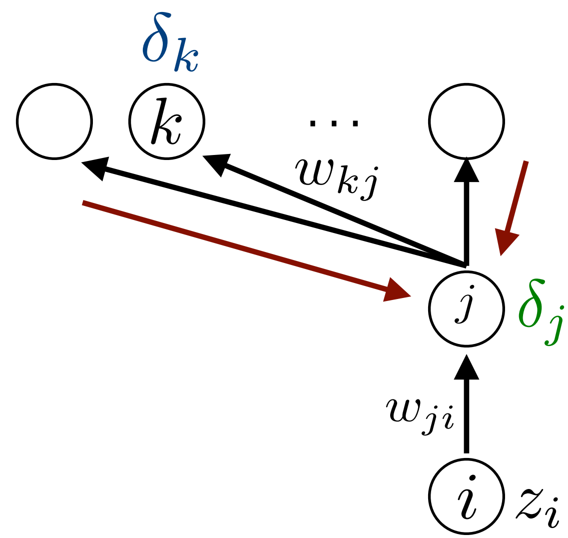
$$\delta_j = f'(a_j) \sum_k w_{kj} \delta_k$$

$$w_{ji}^{\text{neu}} = w_{ji}^{\text{alt}} - \eta \delta_j z_i$$

Training: Overview



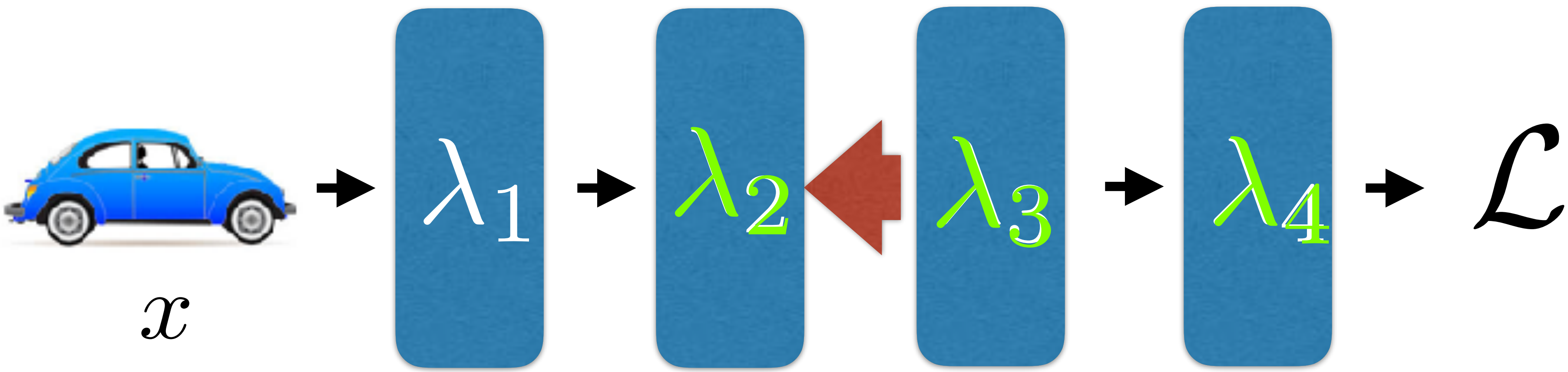
- Use Chain Rule to compute gradient recursively



$$\delta_j = f'(a_j) \sum_k w_{kj} \delta_k$$

$$w_{ji}^{\text{neu}} = w_{ji}^{\text{alt}} - \eta \delta_j z_i$$

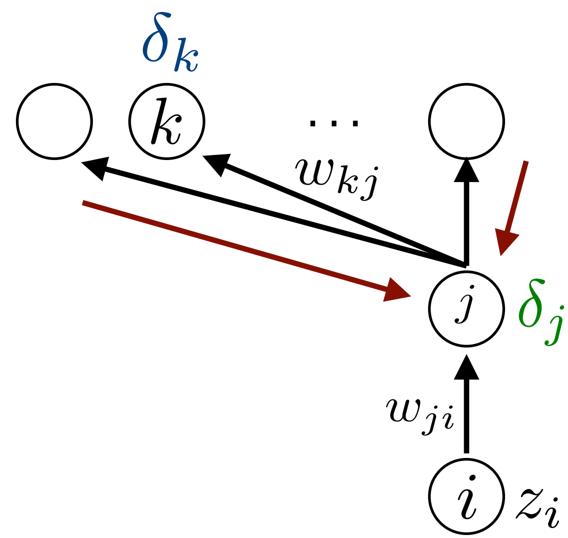
Training: Overview



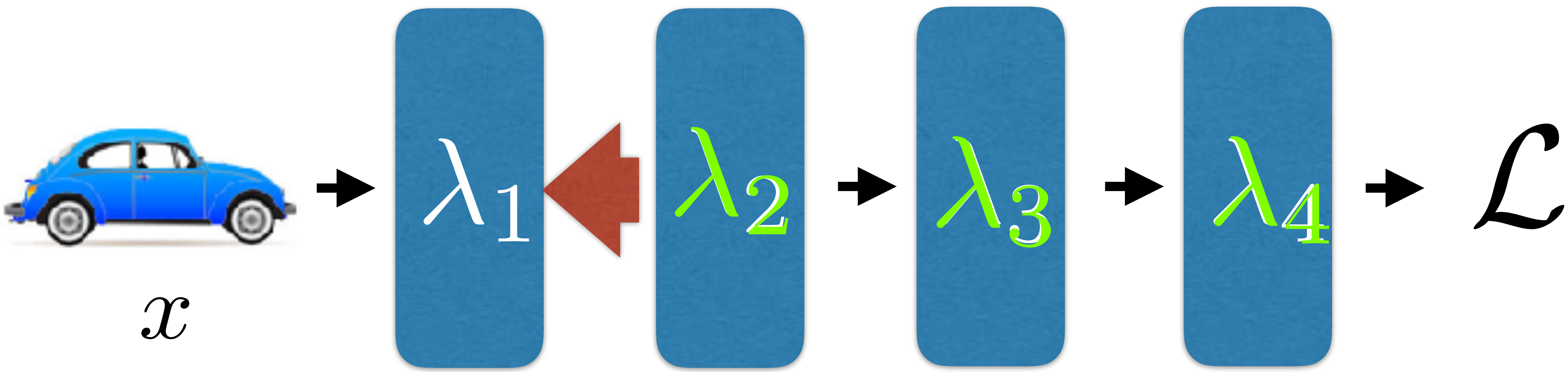
- Use Chain Rule to compute gradient recursively

$$\delta_j = f'(a_j)\sum_k w_{kj}\delta_k$$

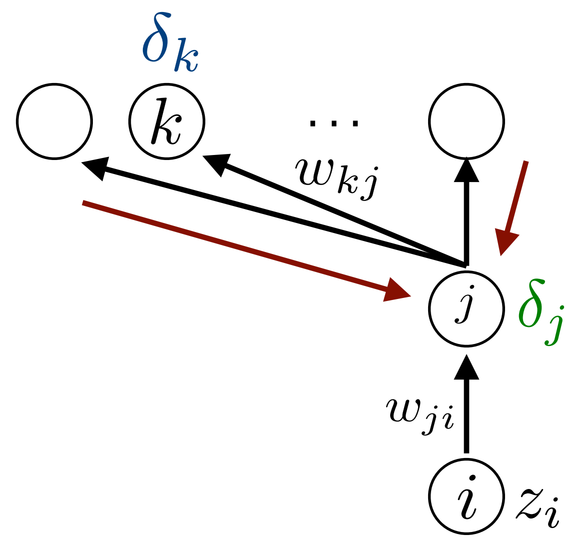
$$w_{ji}^{\text{neu}} = w_{ji}^{\text{alt}} - \eta\delta_j z_i$$



Training: Overview



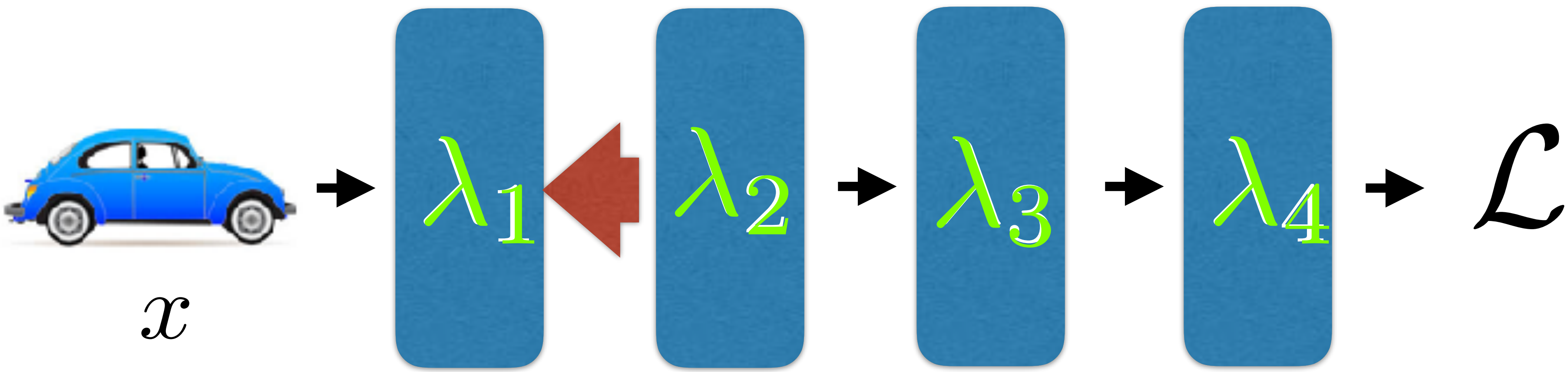
- Use Chain Rule to compute gradient recursively



$$\delta_j = f'(a_j) \sum_k w_{kj} \delta_k$$

$$w_{ji}^{\text{neu}} = w_{ji}^{\text{alt}} - \eta \delta_j z_i$$

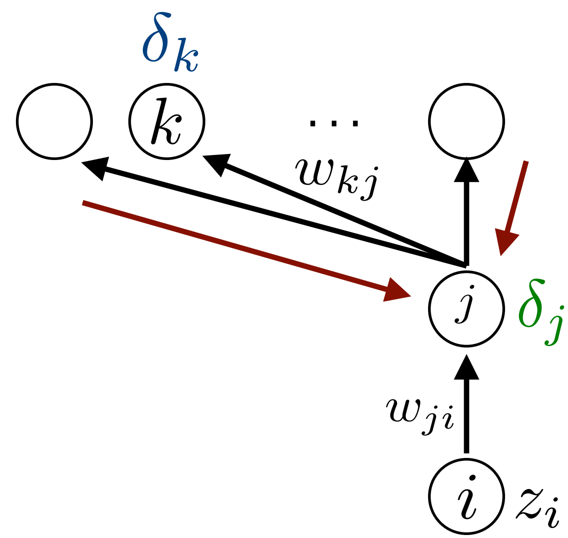
Training: Overview



- Use Chain Rule to compute gradient recursively

$$\delta_j = f'(a_j) \sum_k w_{kj} \delta_k$$

$$w_{ji}^{\text{neu}} = w_{ji}^{\text{alt}} - \eta \delta_j z_i$$



Network Training

Training a neural network means adjusting the weights to get a good approximation of the target function.

How does a neural network learn?

- *Supervised learning*: *Training set* T provides both input values and the corresponding target values:

input value – *pattern*

$$T := \{(x_n, t_n) : 1 \leq n \leq N\}. \quad (6)$$

target value

- Approximation performance of the neural network can be evaluated using a distance measure between approximation and target function.

slide taken from David Stutz (Aachen)

Network Training - Error Measures

Sum-of-squared error function:

weight vector

k-th component of modeled function y

$$E(w) = \sum_{n=1}^{N} E_n(w) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{C} (y_k(x_n, w) - t_{nk})^2.$$

k-th entry of t_n

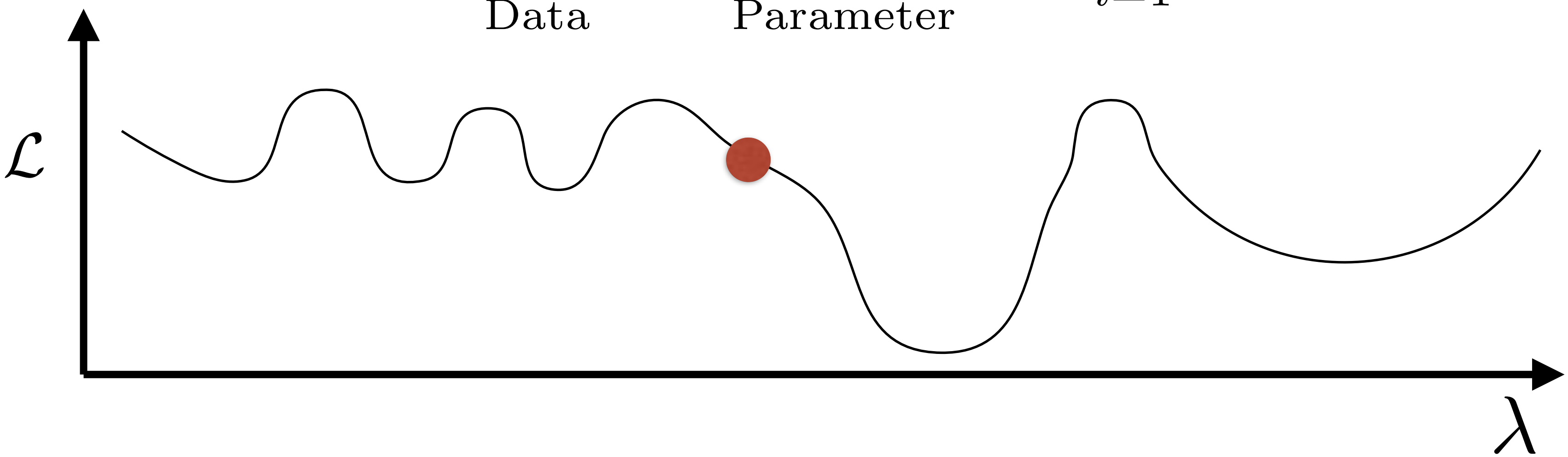
Cross-entropy error function:

$$E(w) = \sum_{n=1}^{N} E_n(w) = - \sum_{n=1}^{N} \sum_{k=1}^{C} t_{nk} \log y_k(x_n, w).$$

slide taken from David Stutz (Aachen)

Learning by Optimization

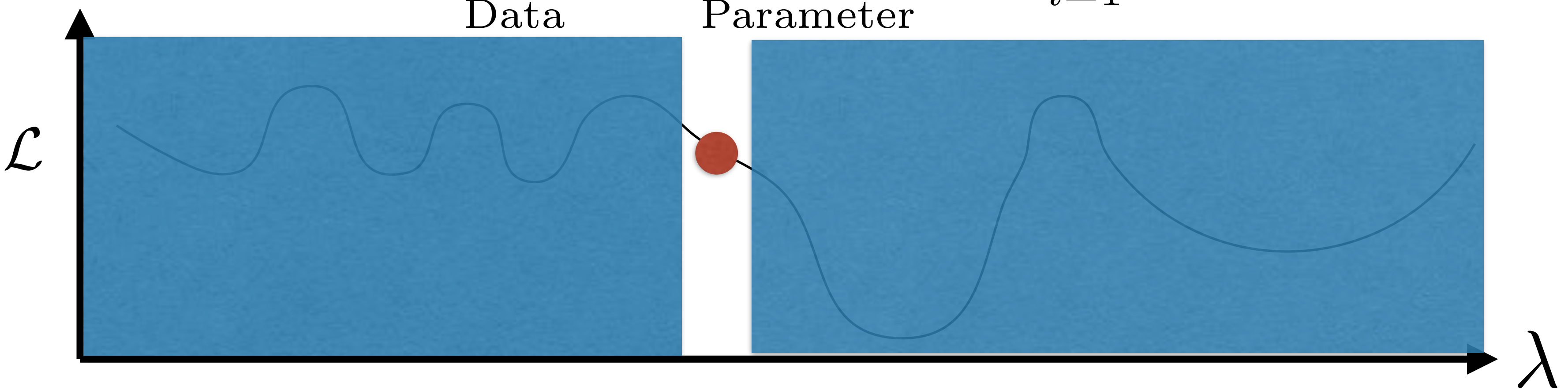
$$\mathcal{L}(\underbrace{x_1, \ldots, x_N}_{\text{Data}}, \underbrace{\lambda_1, \ldots, \lambda_L}_{\text{Parameter}}) = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$



- Highly non-convex energy

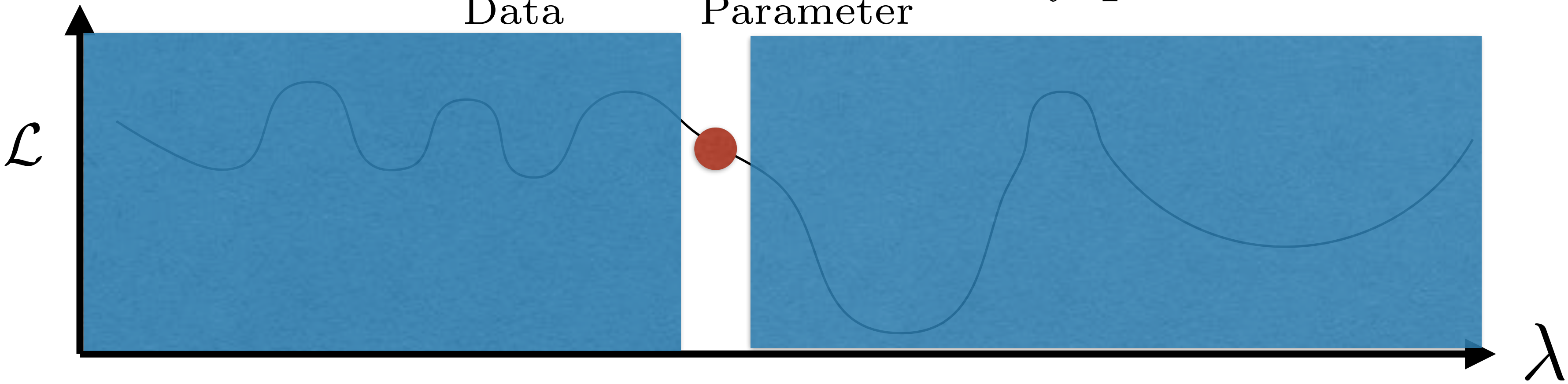
Learning by Optimization

$$\mathcal{L}(\underbrace{x_1,\ldots,x_N}_{\text{Data}},\underbrace{\lambda_1,\ldots,\lambda_L}_{\text{Parameter}}) = \sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$



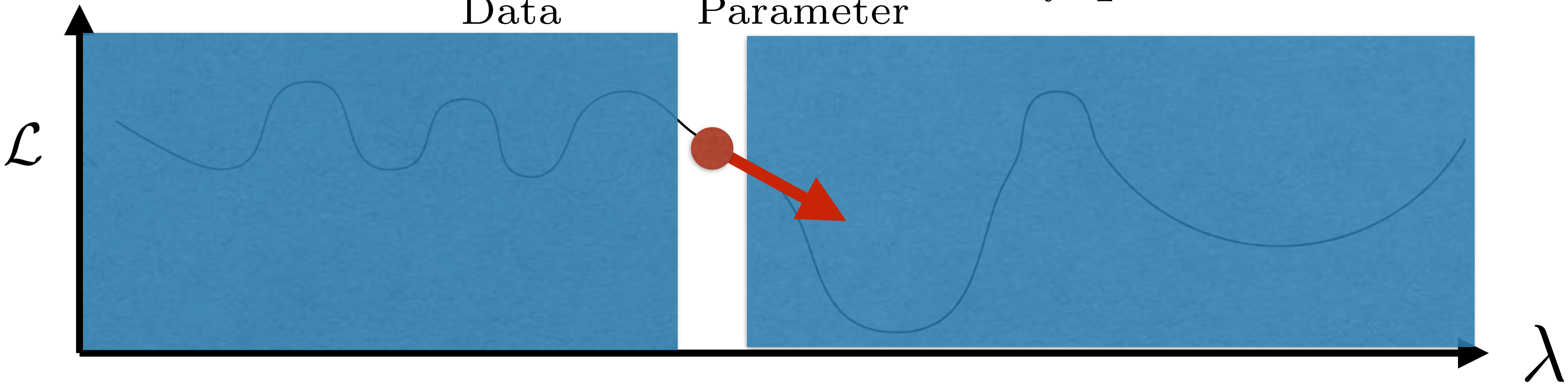
Learning by Optimization

$$\mathcal{L}(\underbrace{x_1, \ldots, x_N}_{\text{Data}}, \underbrace{\lambda_1, \ldots, \lambda_L}_{\text{Parameter}}) = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$



Lernen durch Optimierung

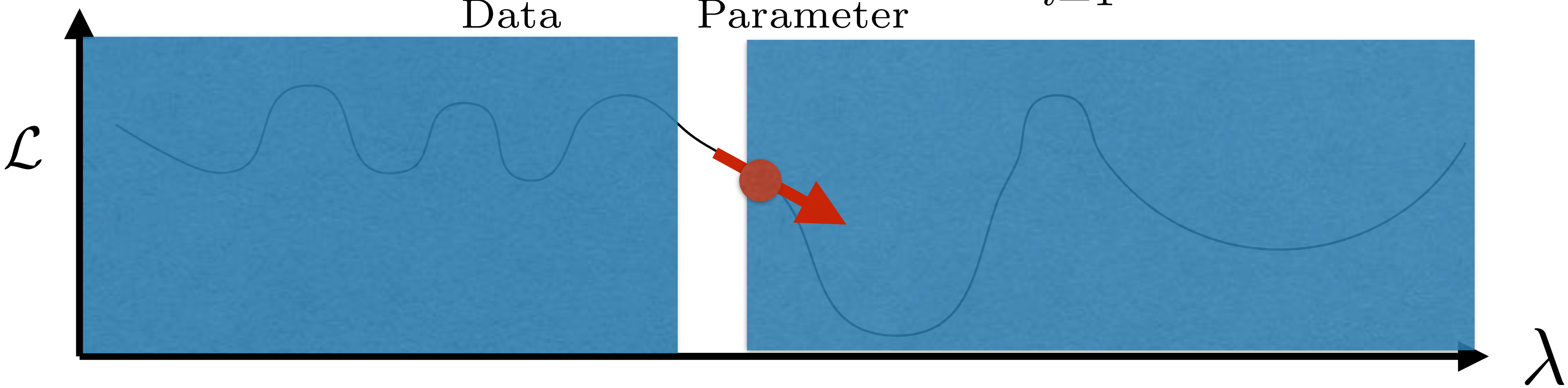
$$\mathcal{L}(\underbrace{x_1, \ldots, x_N}_{\text{Data}}, \underbrace{\lambda_1, \ldots, \lambda_L}_{\text{Parameter}}) = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$



- Gradient Descent

Lernen durch Optimierung

$$\mathcal{L}(\underbrace{x_1, \ldots, x_N}_{\text{Data}}, \underbrace{\lambda_1, \ldots, \lambda_L}_{\text{Parameter}}) = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$



- Gradient Descent

Network Training - Approaches

Idea: Adjust the weights such that the error is minimized.

Stochastic training Randomly choose an input value x_n and update the weights based on the error $E_n(w)$.

Mini-batch training Process a subset $M \subseteq \{1, \dots, N\}$ of all input values and update the weights based on the error $\sum_{n \in M} E_n(w)$.

Batch training Process all input values x_n, $1 \leq n \leq N$ and update the weights based on the overall error $E(w) = \sum_{n=1}^{N} E_n(w)$.

slide taken from David Stutz (Aachen)

Network Training - Parameter Optimization

How to minimize the error $E(w)$?

Problem: $E(w)$ can be nonlinear and may have multiple local minima.

Iterative optimization algorithms:

- Let $w[0]$ be a starting vector for the weights.
- $w[t]$ is the weight vector in the t-th iteration of the optimization algorithm.
- In iteration $[t+1]$ choose a *weight update* $\Delta w[t]$ and set

$$w[t+1] = w[t] + \Delta w[t].$$

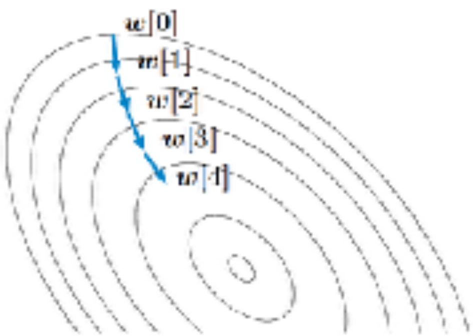
- Different optimization algorithms choose different weight updates.

slide taken from David Stutz (Aachen)

Parameter Optimization by Gradient Descent

Idea: In each iteration take a step in the direction of the negative gradient.

- The direction of the steepest descent.



- Weight update $\Delta w[t]$ is given by

$$\Delta w[t] = -\gamma \frac{\partial E}{\partial w[t]}.$$

learning rate – *step size*

slide taken from David Stutz (Aachen)

Backpropagation = Parameter Optimization by Gradient Descent

Summary: We want to minimize the error $E(w)$ on the training set T to get a good approximation of the target function.

Using gradient descent and stochastic learning, the weight update in iteration $[t+1]$ is given by

$$w[t+1]_{ij}^{(l)} = w[t]_{ij}^{(l)} - \gamma \frac{\partial E_n}{\partial w[t]_{ij}^{(l)}}. \qquad (11)$$

How to evaluate the gradient $\frac{\partial E_n}{\partial w_{ij}^{(l)}}$ of the error function with respect to the current weight vector?

Using the chain rule we can write:

$$\frac{\partial E_n}{\partial w_{ij}^{(l)}} = \frac{\partial E_n}{\partial z_i^{(l)}} \underbrace{\frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}}_{=y_j^{(l-1)}}. \qquad (12)$$

slide taken from David Stutz (Aachen)

Backpropagation = Parameter Optimization by Gradient Descent

Error backpropagation allows to evaluate $\frac{\partial E_n}{\partial w_{ij}^{(l)}}$ for each weight in $\mathcal{O}(W)$ where W is the total number of weights:

(1) Calculate the *errors* $\delta_i^{(L+1)}$ for the output layer:

$$\delta_i^{(L+1)} := \frac{\partial E_n}{\partial z_i^{(L+1)}} = \frac{\partial E_n}{\partial y_i^{(L+1)}} f'\left(z_i^{(L+1)}\right). \quad (13)$$

- The output errors are often easy to calculate.
  - For example using the sum-of-squared error function and the identity as output activation function:

$$\delta_i^{(L+1)} = \frac{\partial \left[\frac{1}{2}\sum_{k=1}^{C}(y_k^{(L+1)} - t_{nk})^2\right]}{\partial y_i^{(L+1)}} \cdot 1 = y_i(x_n, w) - t_{ni}. \quad (14)$$

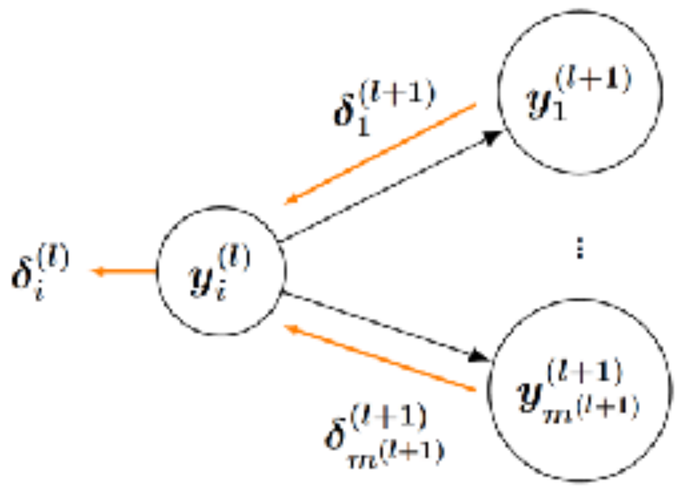
slide taken from David Stutz (Aachen)

Backpropagation = Parameter Optimization by Gradient Descent

(2) Backpropagate the errors $\delta_i^{(L+1)}$ through the network using

$$\delta_i^{(l)} := \frac{\partial E_n}{\partial z_i^{(l)}} = f'\left(z_i^{(l)}\right) \sum_{k=1}^{m^{(l+1)}} w_{ik}^{(l+1)} \delta_k^{(l+1)}. \tag{15}$$

- This can be evaluated recursively for each layer after determining the errors $\delta_i^{(L+1)}$ for the output layer.



slide taken from David Stutz (Aachen)

Backpropagation = Parameter Optimization by Gradient Descent

(3) Determine the needed derivatives using

$$\frac{\partial E_n}{\partial w_{ij}^{(l)}} = \frac{\partial E_n}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} y_j^{(l-1)}.$$

Now use the derivatives $\frac{\partial E_n}{\partial w_{ij}^{(l)}}$ to update the weights in each iteration.

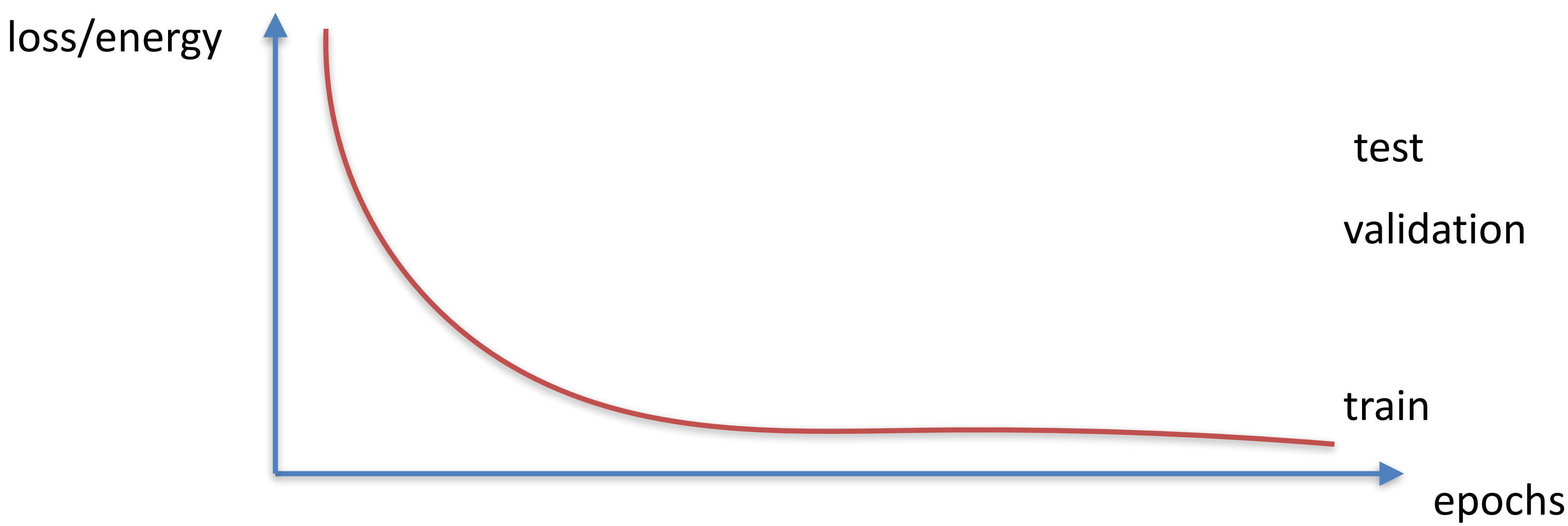
- In iteration step $[t+1]$ set

$$w[t+1]_{ij}^{(l)} = w[t]_{ij}^{(l)} - \gamma \frac{\partial E_n}{\partial w[t]_{ij}^{(l)}}.$$

slide taken from David Stutz (Aachen)

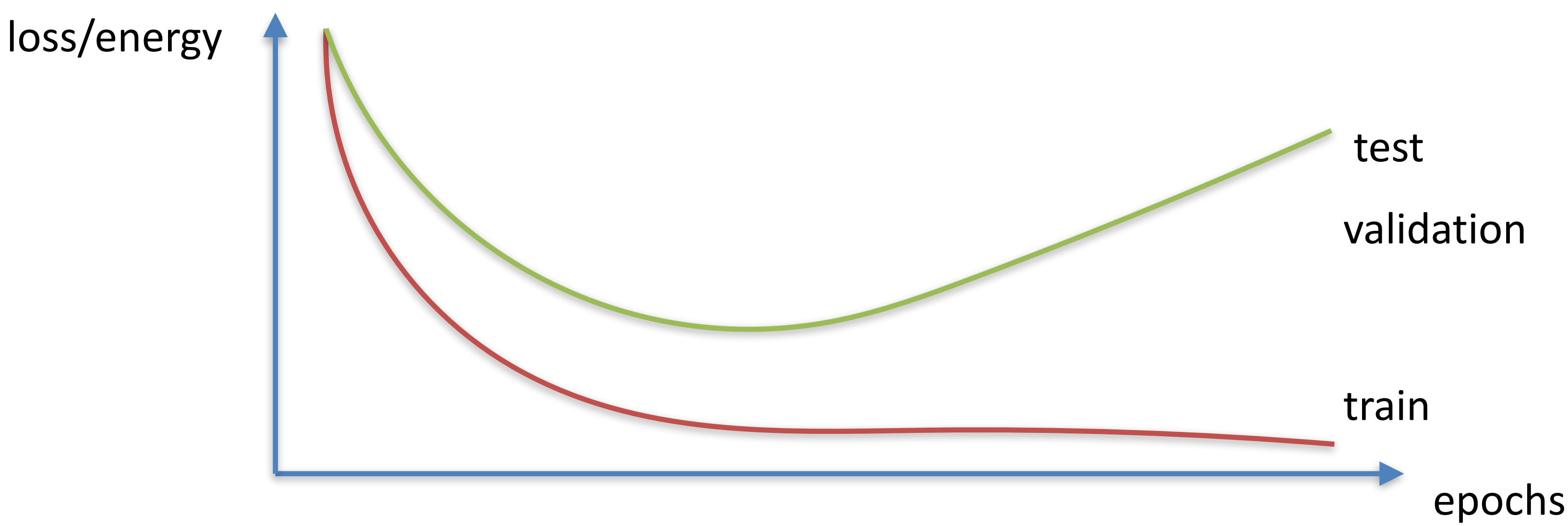
Gradient Checking & Training

- Only two things need to be implemented to define new layer:
  - forward pass
  - error back propagation
- Often buggy implementations lead to reduced energy/loss
  - Gradient check: compare numeric (finite differences) gradient to analytic
- Watch overfitting



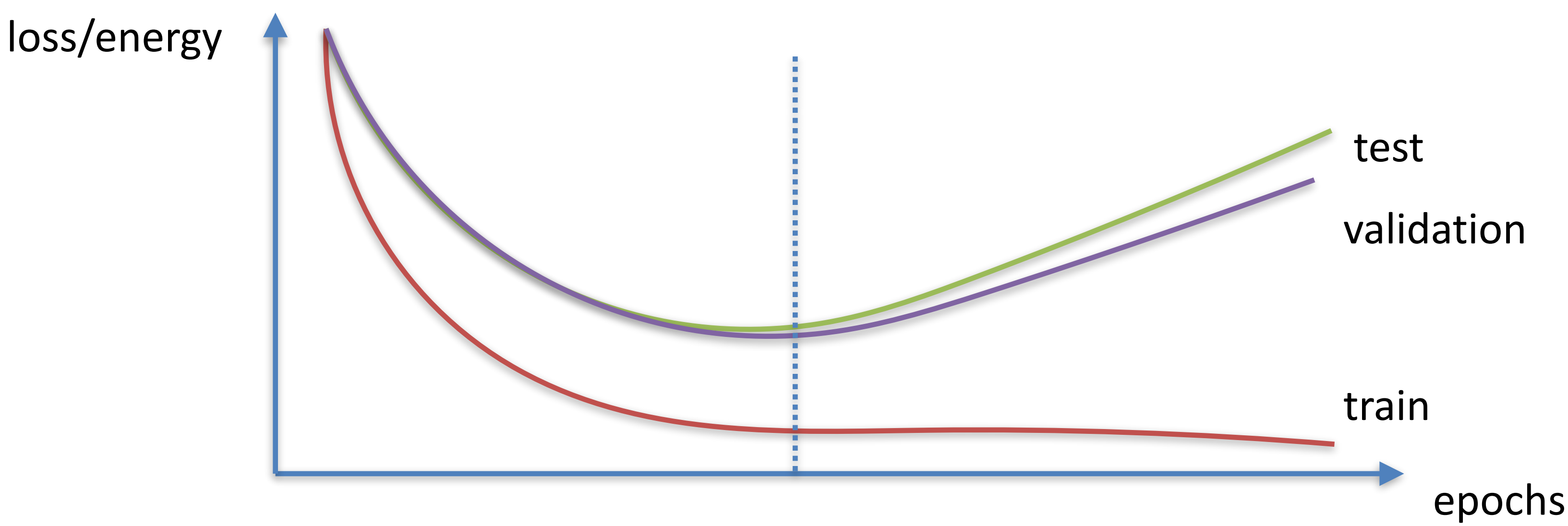
Gradient Checking & Training

- Only two things need to be implemented to define new layer:
  - forward pass
  - error back propagation
- Often buggy implementations lead to reduced energy/loss
  - Gradient check: compare numeric (finite differences) gradient to analytic
- Watch overfitting



Gradient Checking & Training

- Only two things need to be implemented to define new layer:
  - forward pass
  - error back propagation
- Often buggy implementations lead to reduced energy/loss
  - Gradient check: compare numeric (finite differences) gradient to analytic
- Watch overfitting

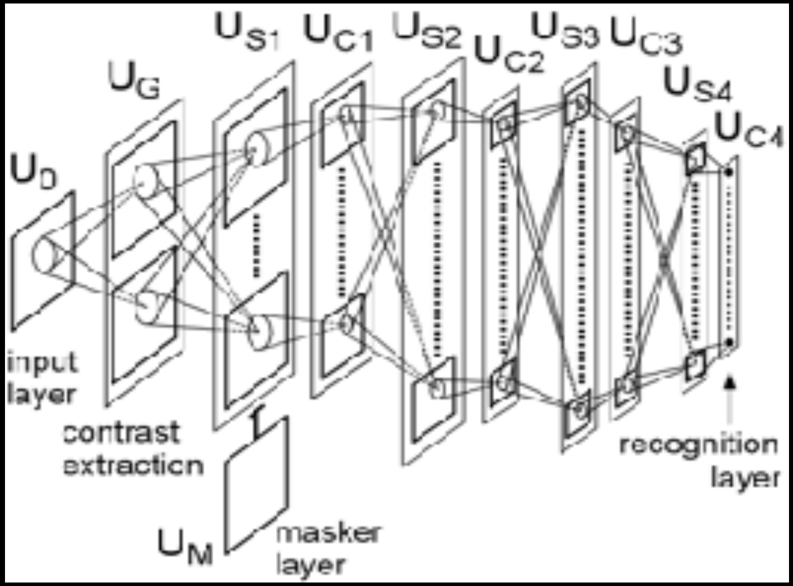


Convolutional Neural Networks

Multistage Hubel&Wiesel Architecture

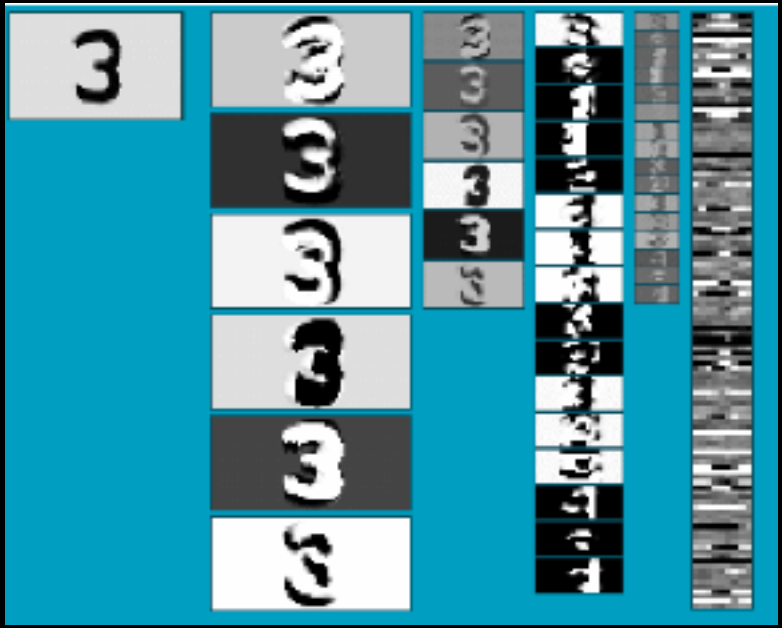
Slide: Y.LeCun

- [Hubel & Wiesel 1962]
  - simple cells detect local features
  - complex cells "pool" the outputs of simple cells within a retinotopic neighborhood.



Cognitron / Neocognitron
[Fukushima 1971-1982]

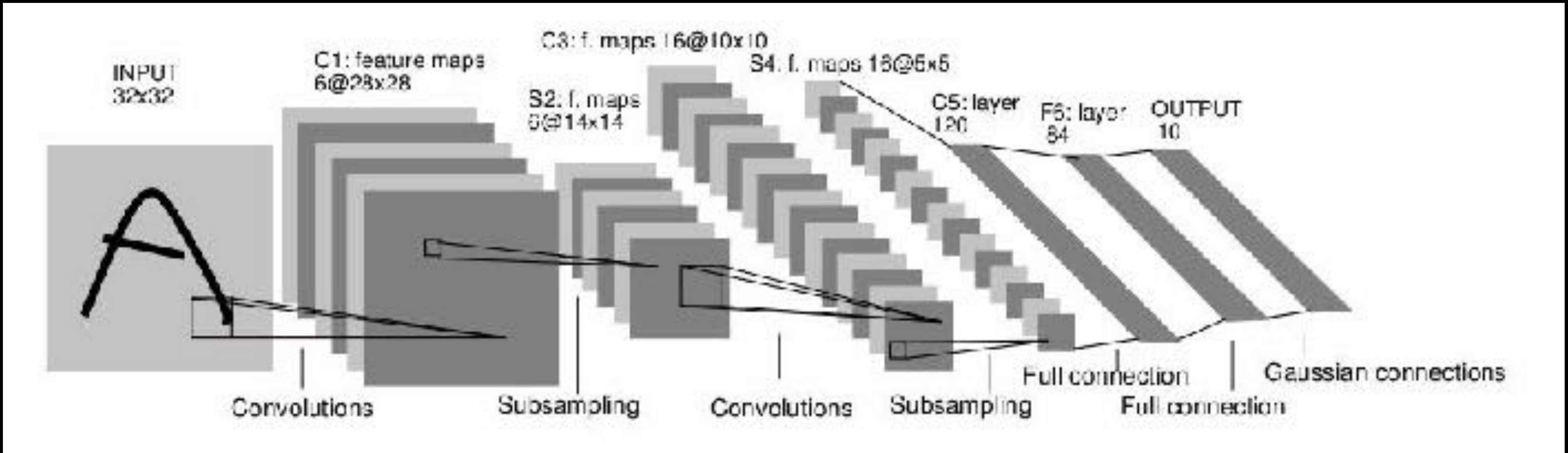
- Also HMAX [Poggio 2002-2006]



Convolutional Networks
[LeCun 1988-present]

Convolutional Neural Networks

- LeCun et al. 1989
- Neural network with specialized connectivity structure



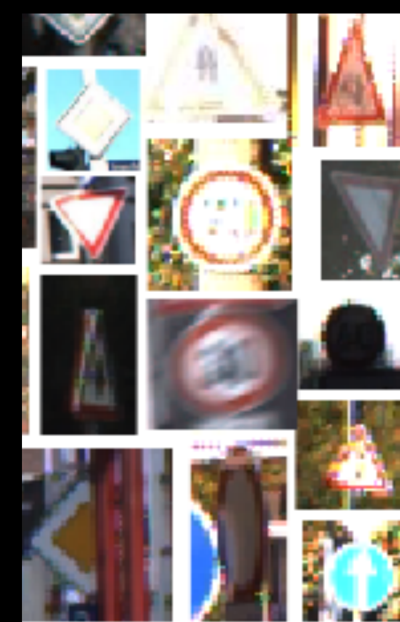
Convnet Successes

- Handwritten text/digits
  - MNIST (0.17% error [Ciresan et al. 2011])
  - Arabic & Chinese [Ciresan et al. 2012]

Convnet Successes

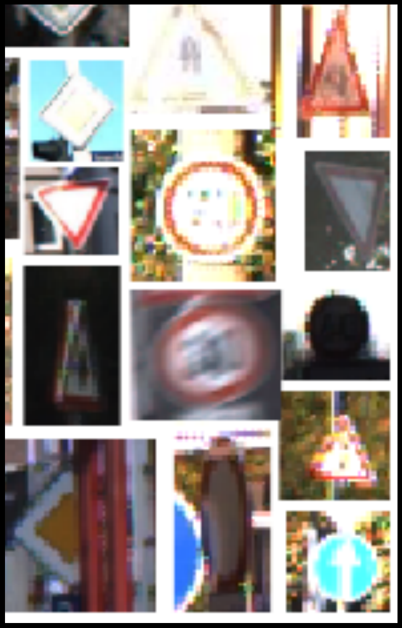
- Handwritten text/digits
  - MNIST (0.17% error [Ciresan et al. 2011])
  - Arabic & Chinese [Ciresan et al. 2012]

- Simpler recognition benchmarks
  - CIFAR-10 (9.3% error [Wan et al. 2013])
  - Traffic sign recognition
    - 0.56% error vs 1.16% for humans [Ciresan et al. 2011]



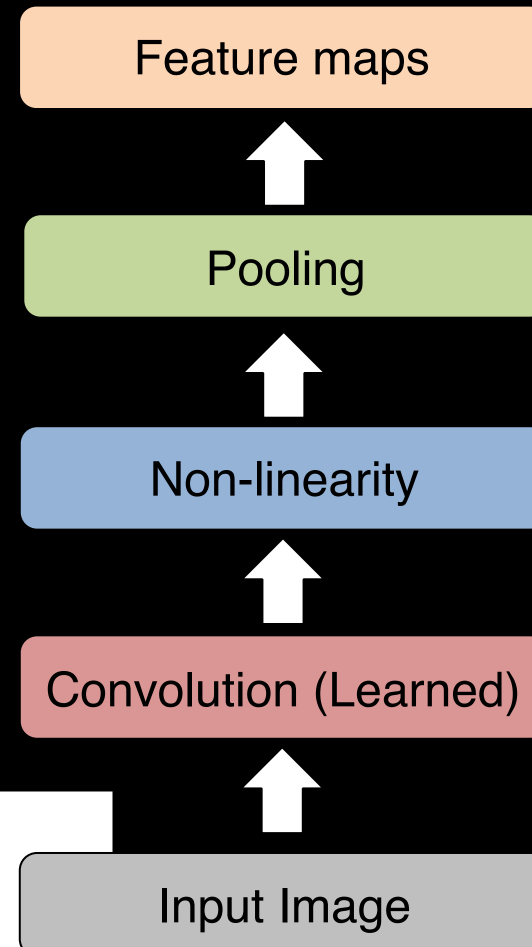
Convnet Successes

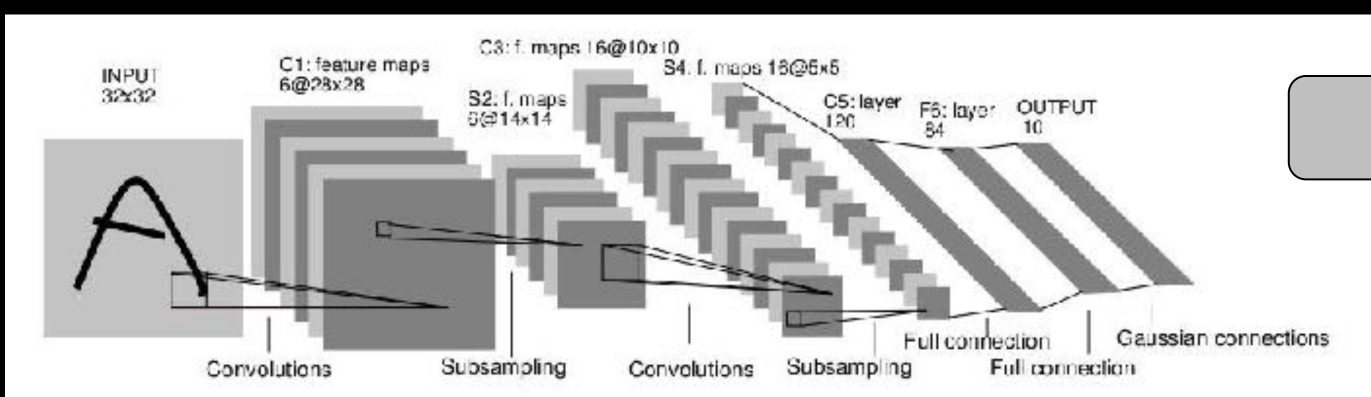
- Handwritten text/digits
  - MNIST (0.17% error [Ciresan et al. 2011])
  - Arabic & Chinese [Ciresan et al. 2012]
- Simpler recognition benchmarks
  - CIFAR-10 (9.3% error [Wan et al. 2013])
  - Traffic sign recognition
    - 0.56% error vs 1.16% for humans [Ciresan et al. 2011]
- But (until recently) less good at more complex datasets
  - E.g. Caltech-101/256 (few training examples)



Characteristics of Convnets

- Feed-forward:
  - Convolve input
  - Non-linearity (rectified linear)
  - Pooling (local max) / (=subsampling)
- Supervised
- Train convolutional filters by back-propagating classification error





[LeCun et al. 1989]

Application to ImageNet

IMAGENET [Deng et al. CVPR 2009]

- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk

Application to ImageNet

[Deng et al. CVPR 2009]

- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk

ImageNet Classification with Deep Convolutional Neural Networks [NIPS 2012]

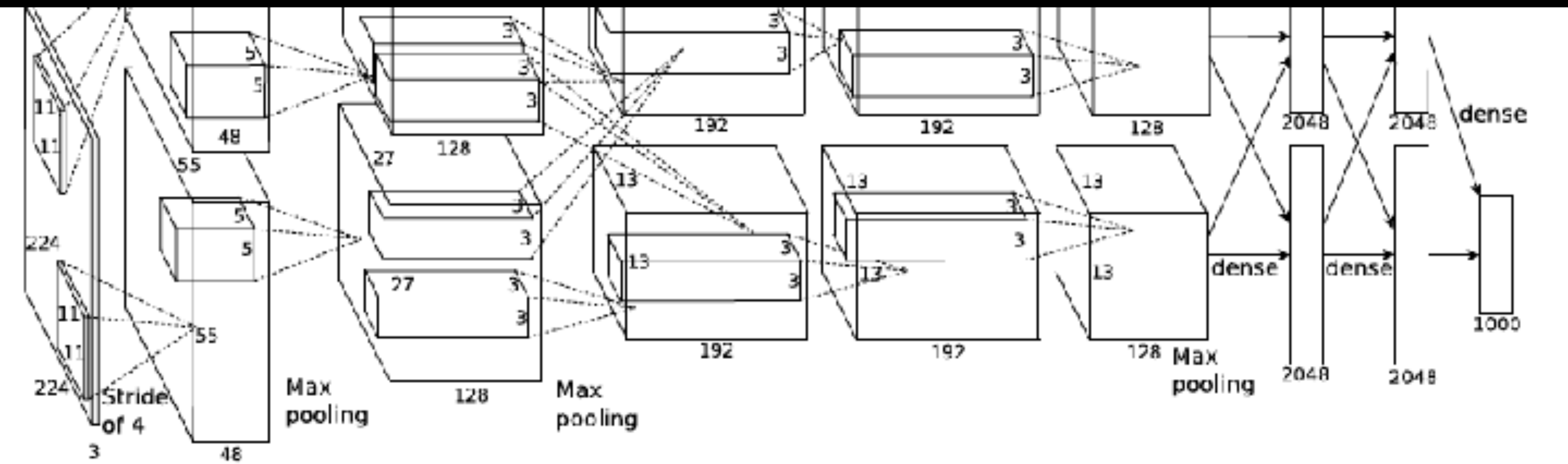
Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Krizhevsky et al. [NIPS2012]

- Same model as LeCun'98 but:
  - Bigger model (8 layers)
  - More data (10^6 vs 10^3 images)
  - GPU implementation (50x speedup over CPU)
  - Better regularization (DropOut)



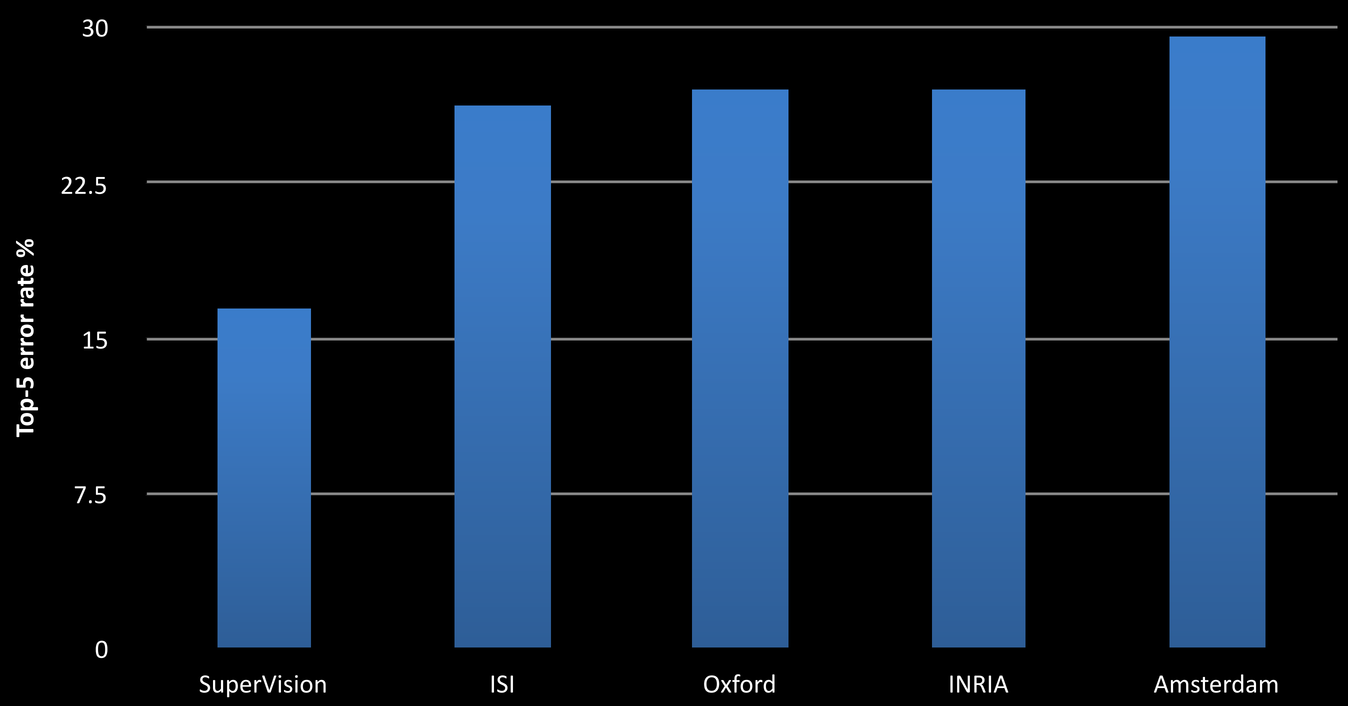
- 7 hidden layers, 650,000 neurons, 60,000,000 parameters
- Trained on 2 GPUs for a week

ImageNet Classification 2012

- Krizhevsky et al. - 16.4% error (top-5)
- Next best (non-convnet) – 26.2% error

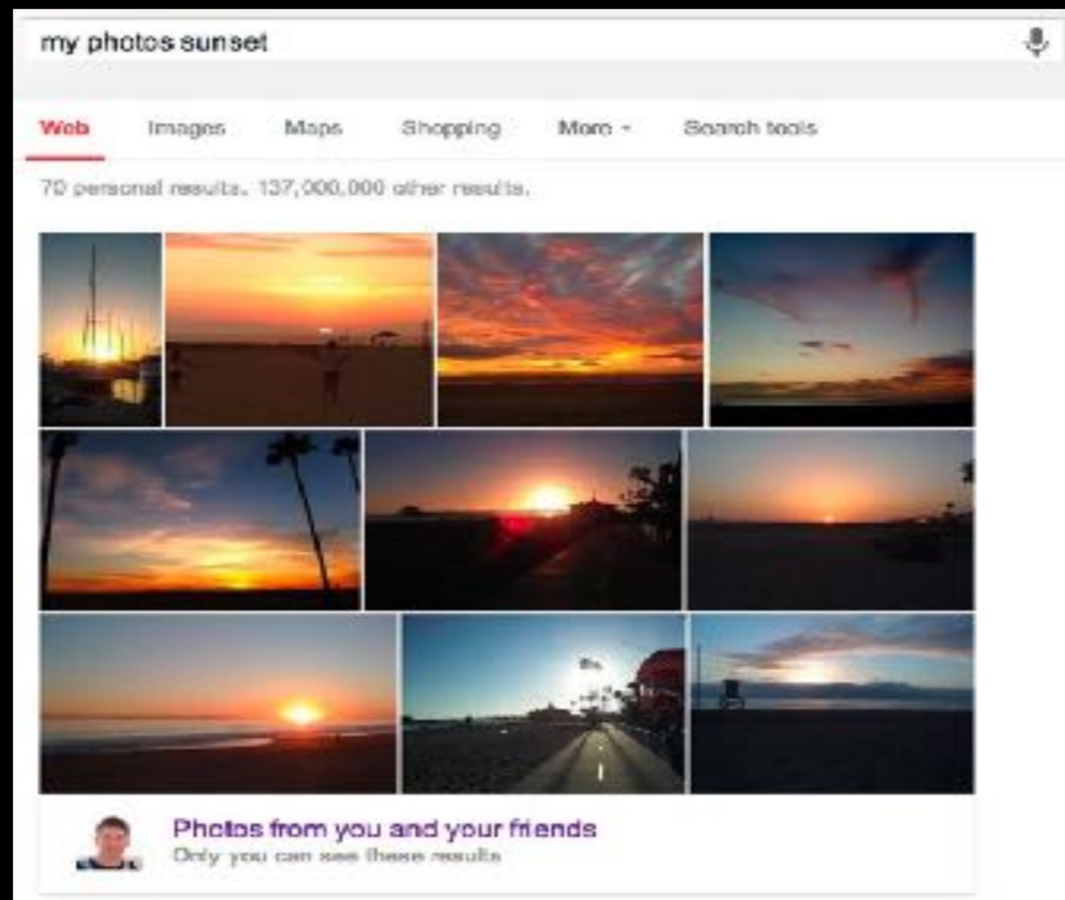
ImageNet Classification 2012

- Krizhevsky et al. - 16.4% error (top-5)
- Next best (non-convnet) – 26.2% error

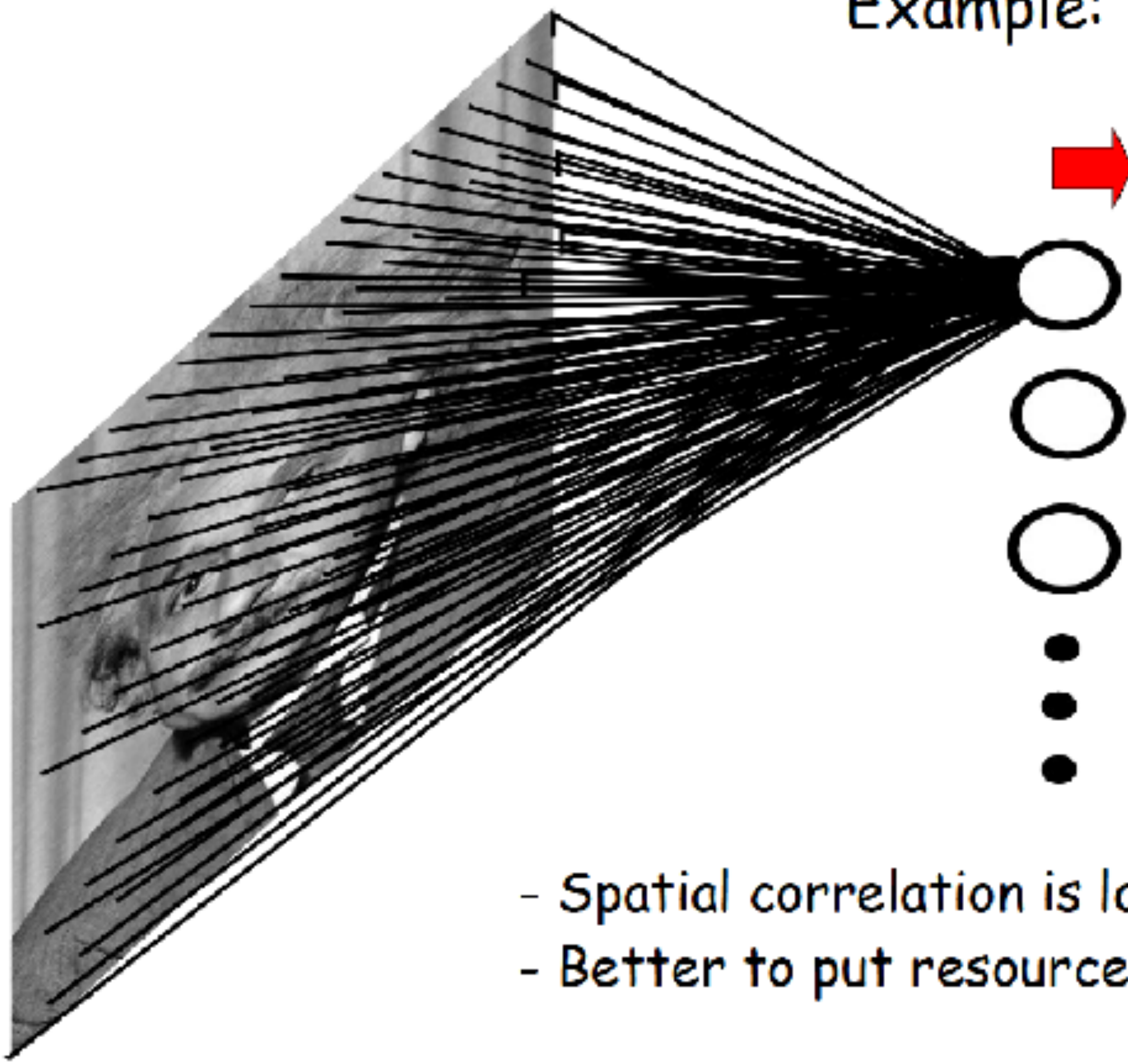


Commercial Deployment

- Google & Baidu, Spring 2013 for personal image search



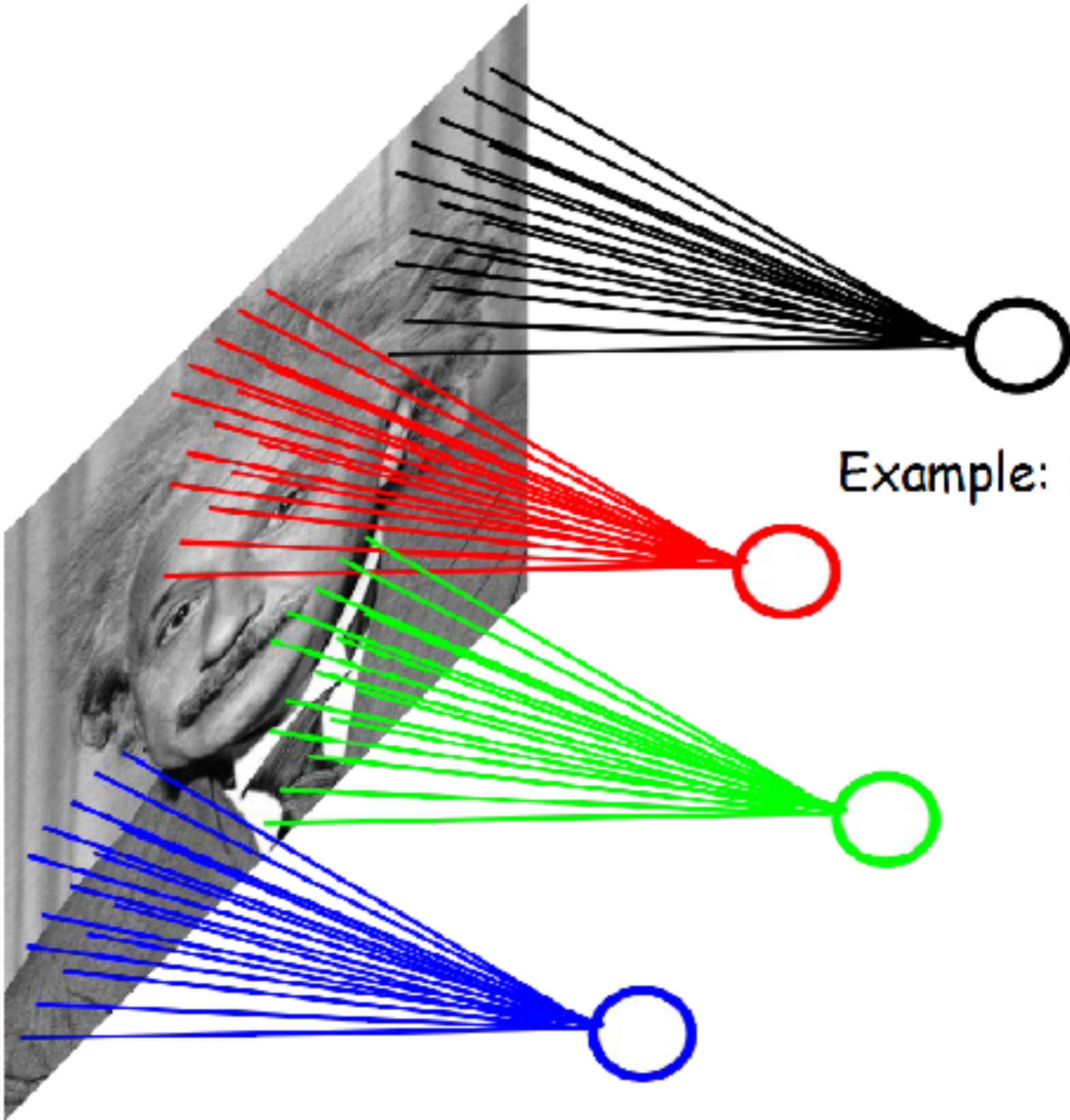
FULLY CONNECTED NEURAL NET



Example: 1000x1000 image
1M hidden units
➡ 10^12 parameters!!!

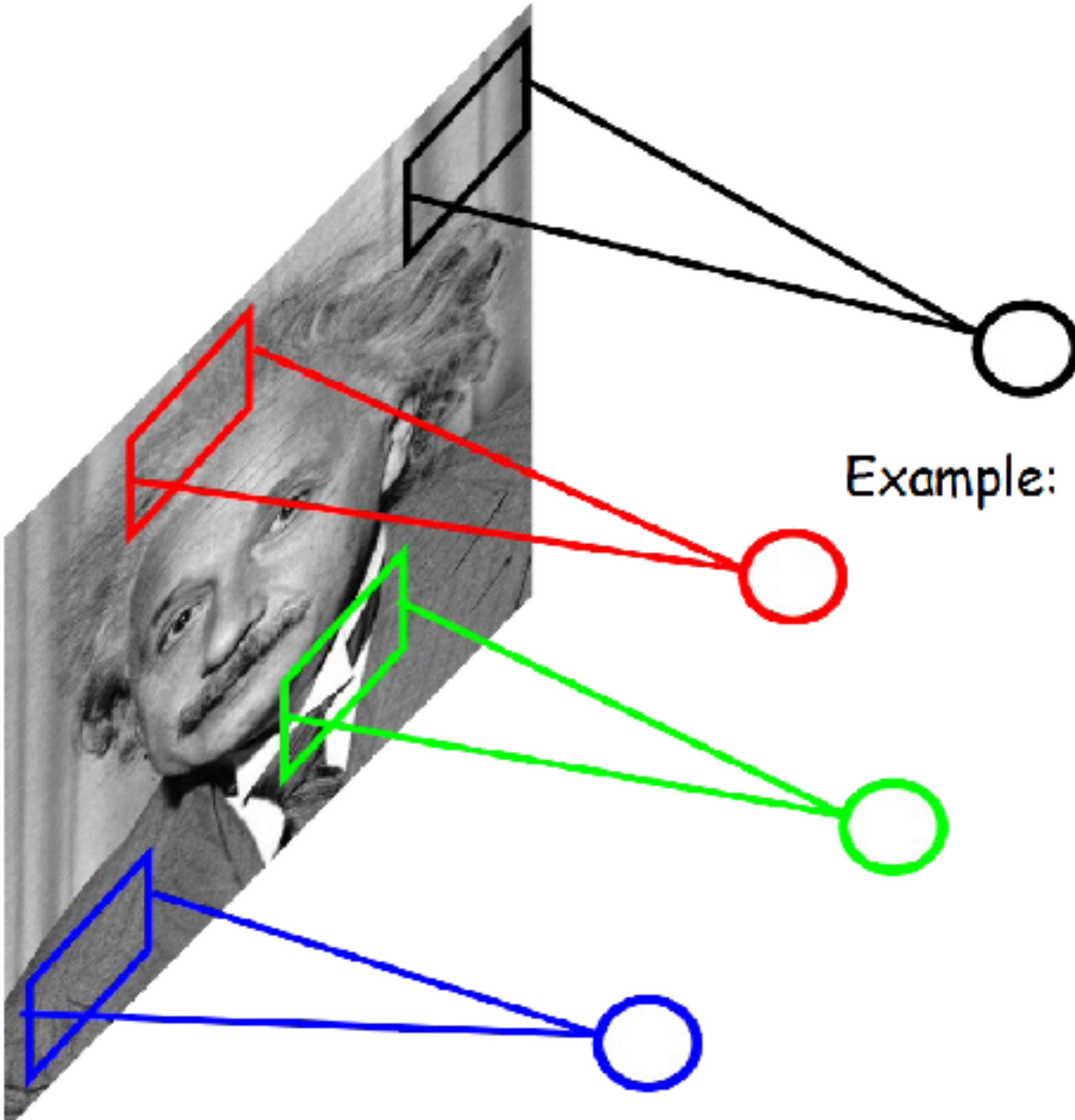
- Spatial correlation is local
- Better to put resources elsewhere!

LOCALLY CONNECTED NEURAL NET



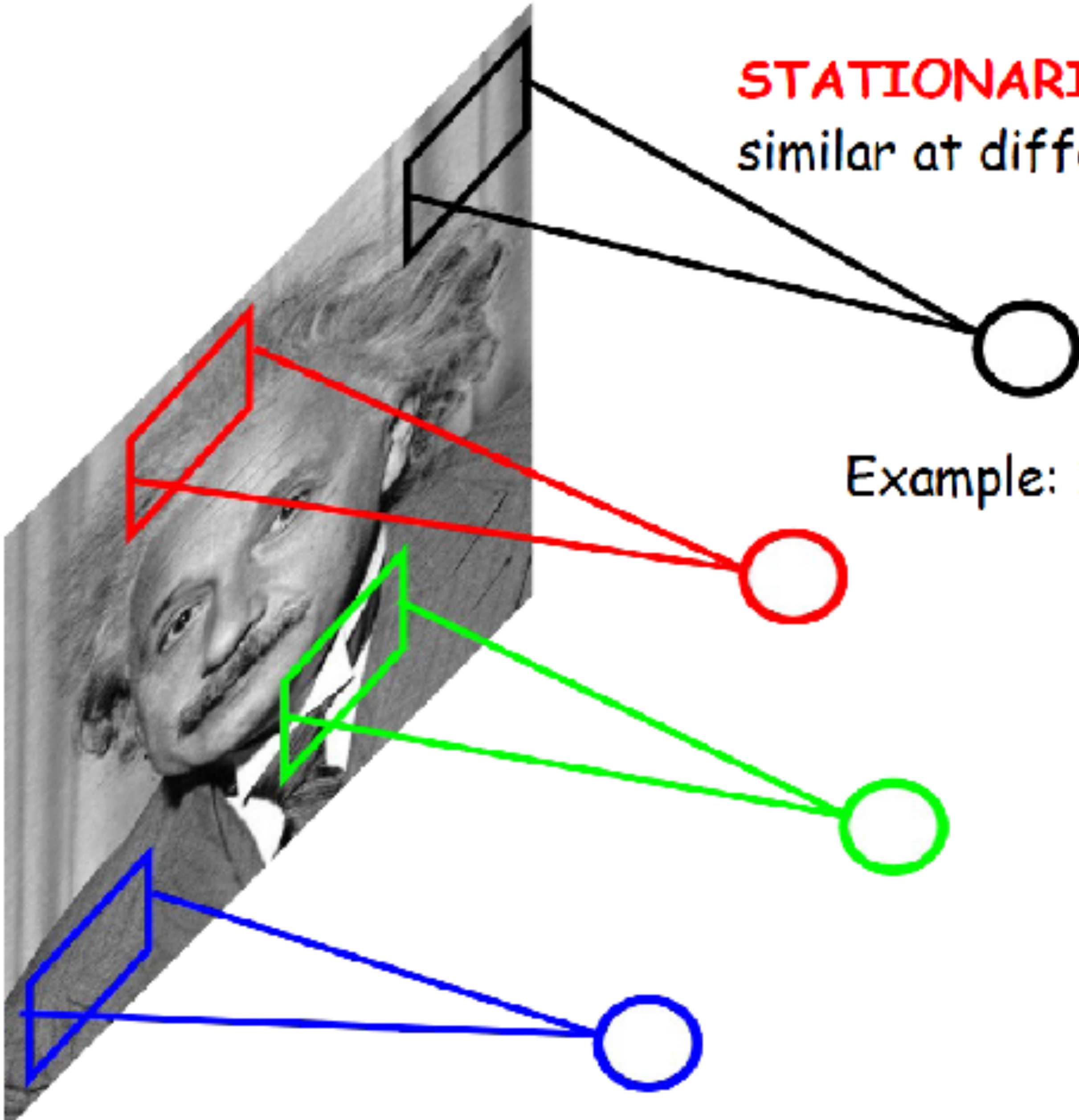
Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

LOCALLY CONNECTED NEURAL NET



Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

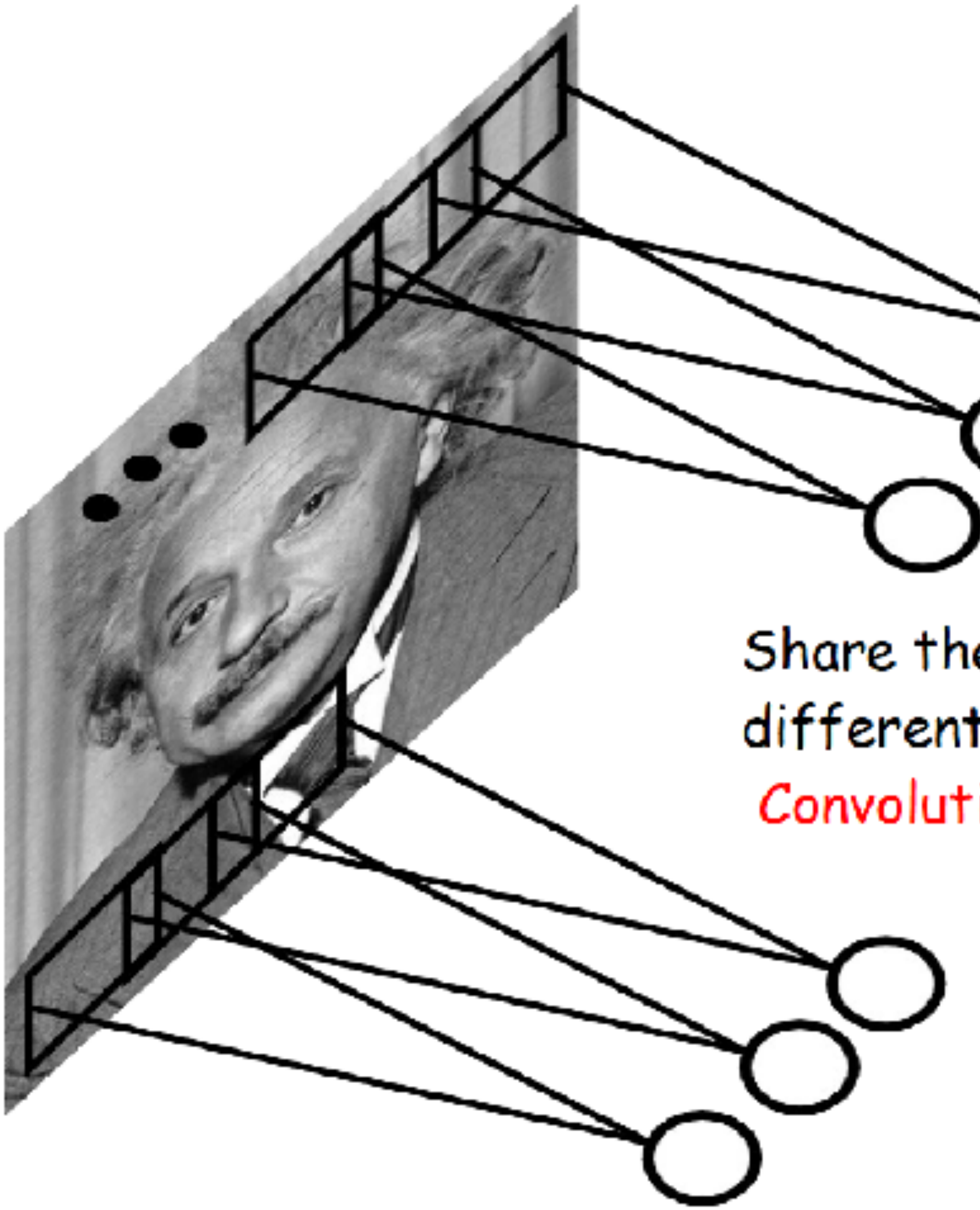
LOCALLY CONNECTED NEURAL NET



STATIONARITY? Statistics is similar at different locations

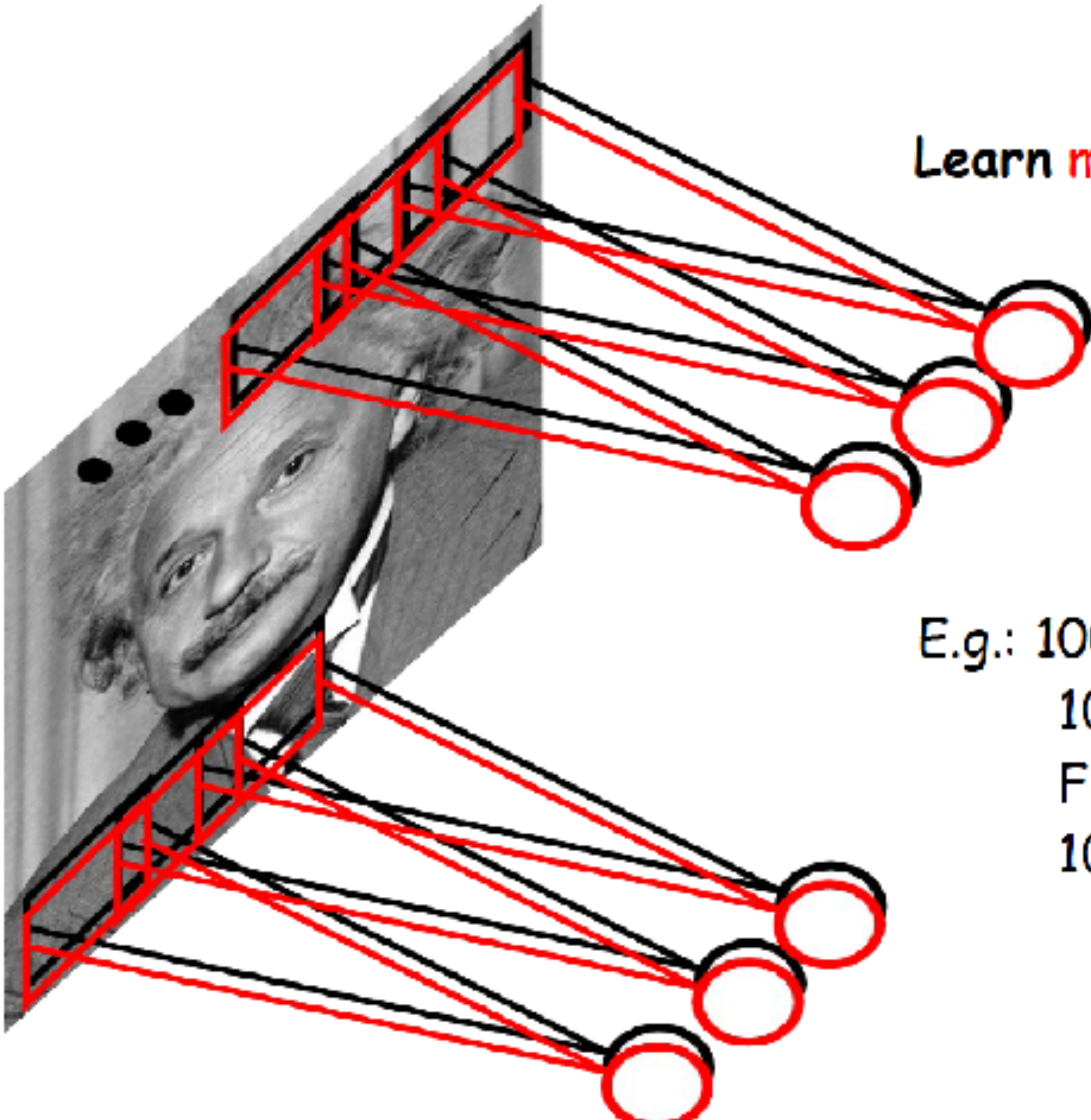
Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

CONVOLUTIONAL NET



Share the same parameters across different locations:
Convolutions with learned kernels

CONVOLUTIONAL NET



Learn multiple filters.

E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters

Ranzato

NEURAL NETS FOR VISION

A standard neural net applied to images:

- scales quadratically with the size of the input
- does not leverage stationarity

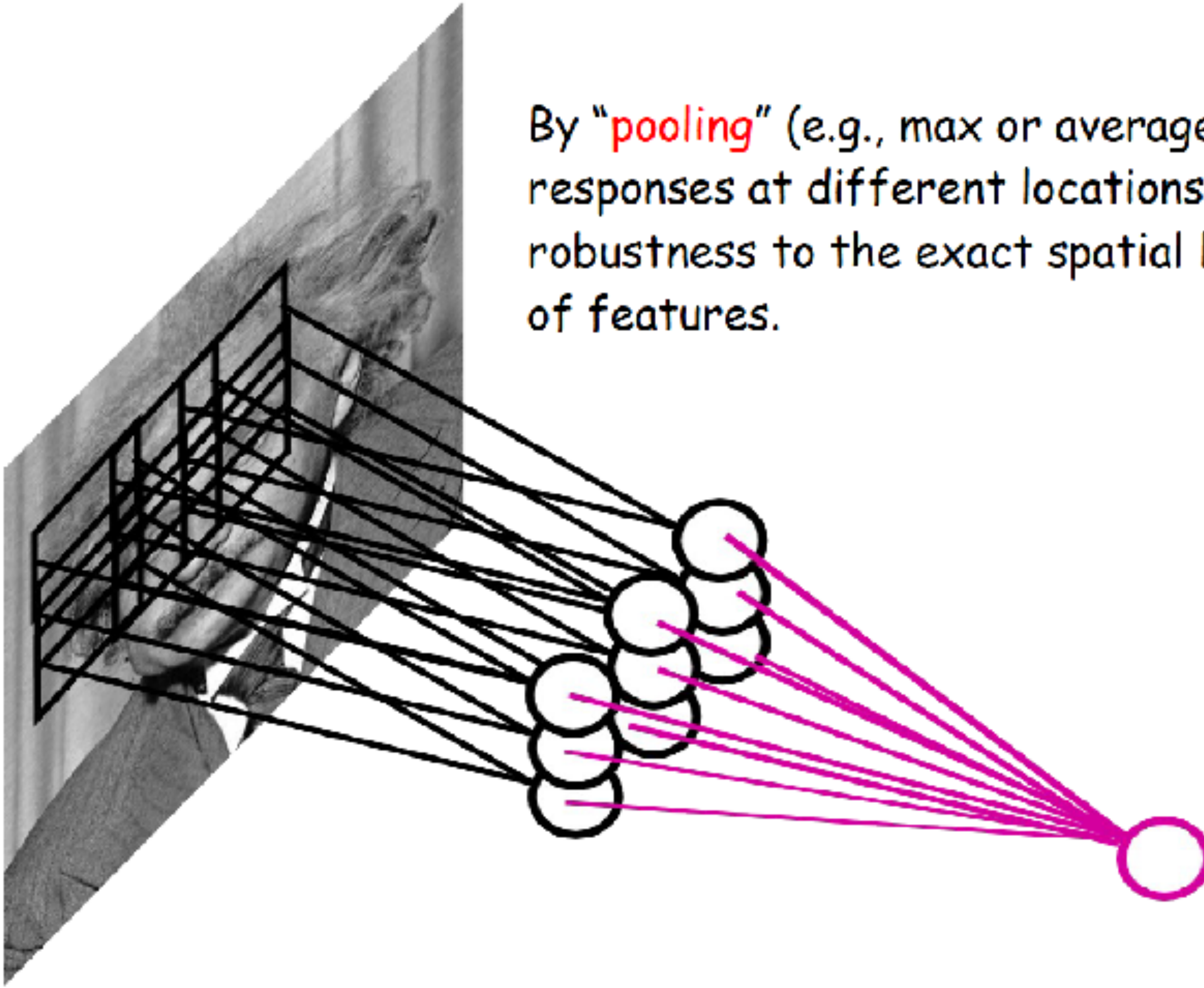
Solution:

- connect each hidden unit to a small patch of the input
- share the weight across hidden units

This is called: **convolutional network.**

LeCun et al. "Gradient-based learning applied to document recognition" IEEE 1998

CONVOLUTIONAL NET



By "pooling" (e.g., max or average) filter responses at different locations we gain robustness to the exact spatial location of features.

Ranzato