TBMI26
Neural Networks and Learning Systems
Lecture 5
Convolutional Neural Networks

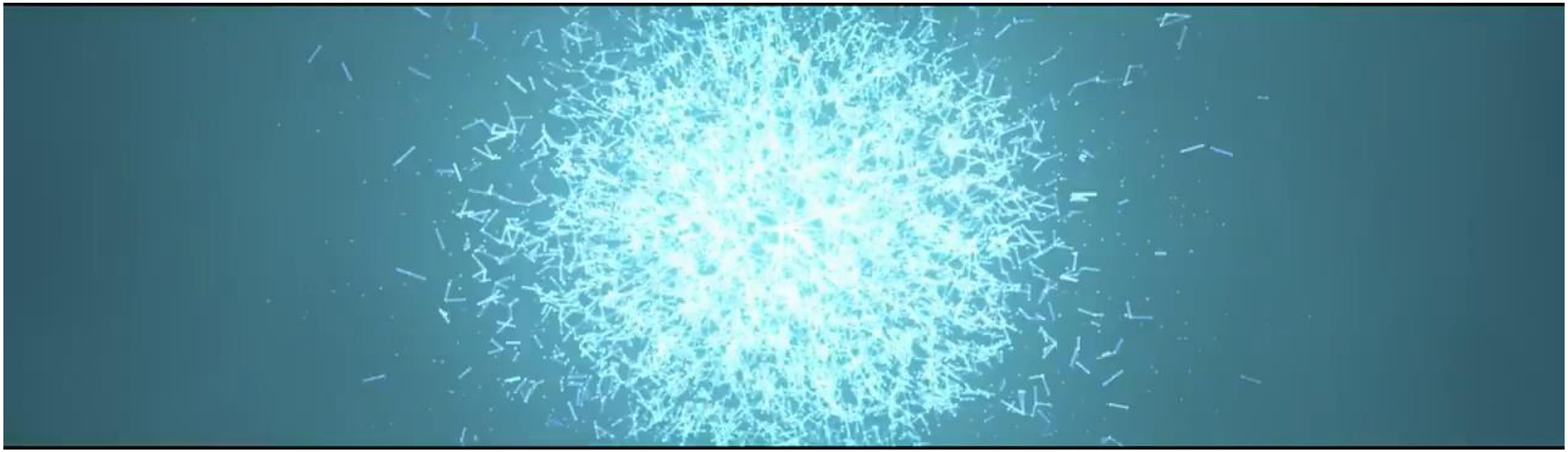
Michael Felsberg
Computer Vision Laboratory
Department of Electrical Engineering
Linköping University, Sweden

LINKÖPINGS UNIVERSITET

Introduction

LINKÖPINGS UNIVERSITET

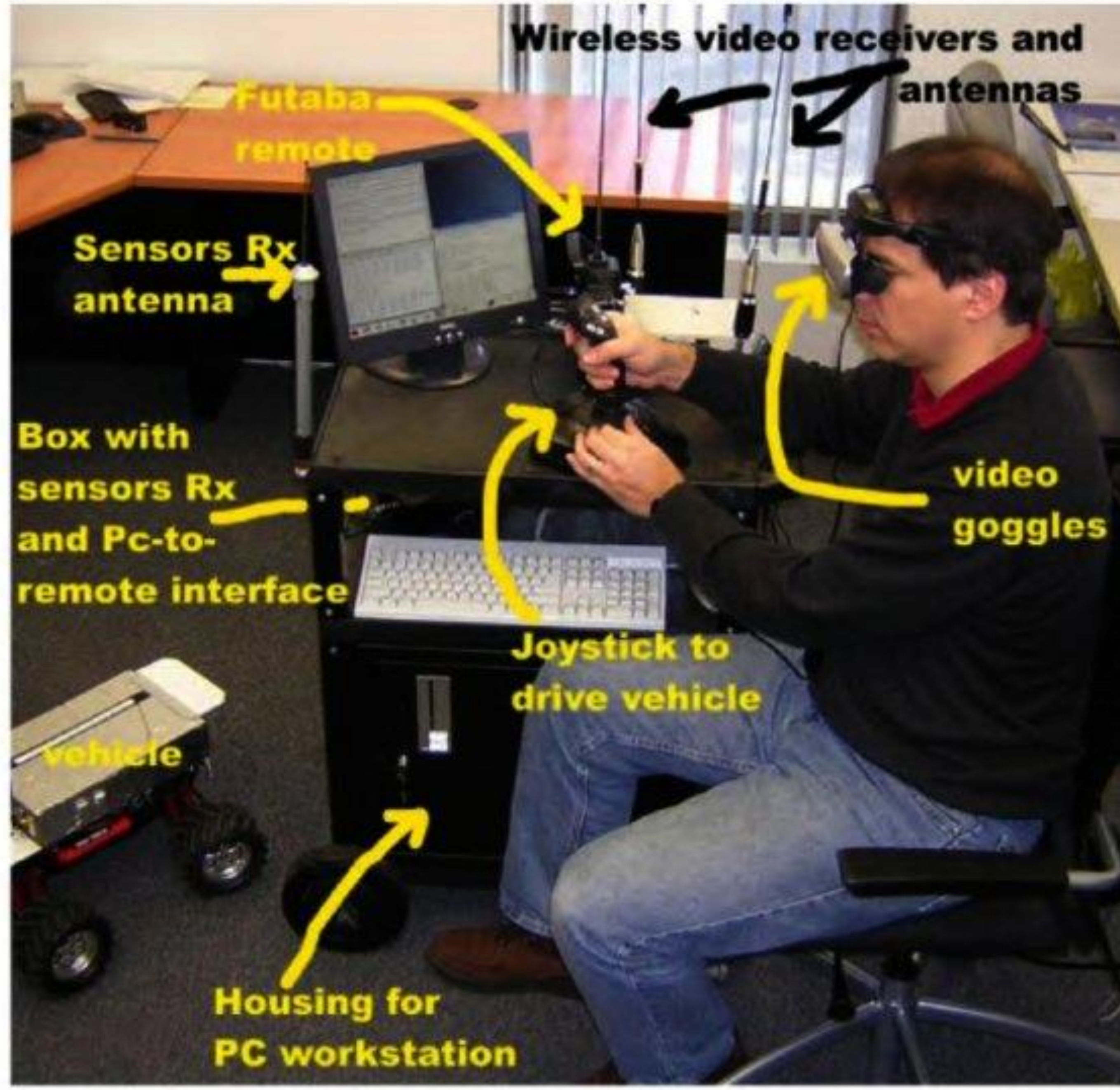
The Deep Learning Revolution



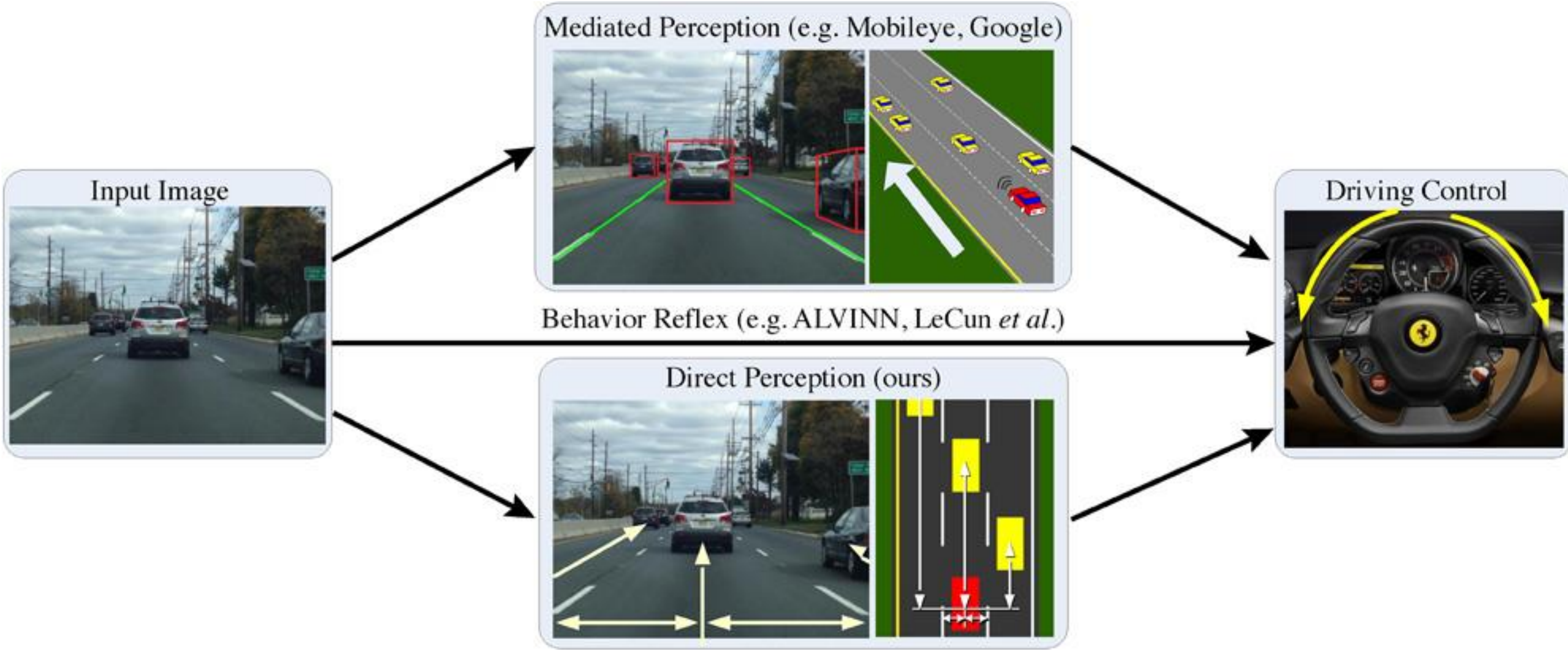
Step 1: Convolutional (Neural) Networks

DAVE [LeCun et al. 2005]

http://www.cs.nyu.edu/~yann/research/dave/



Regression learning for driving

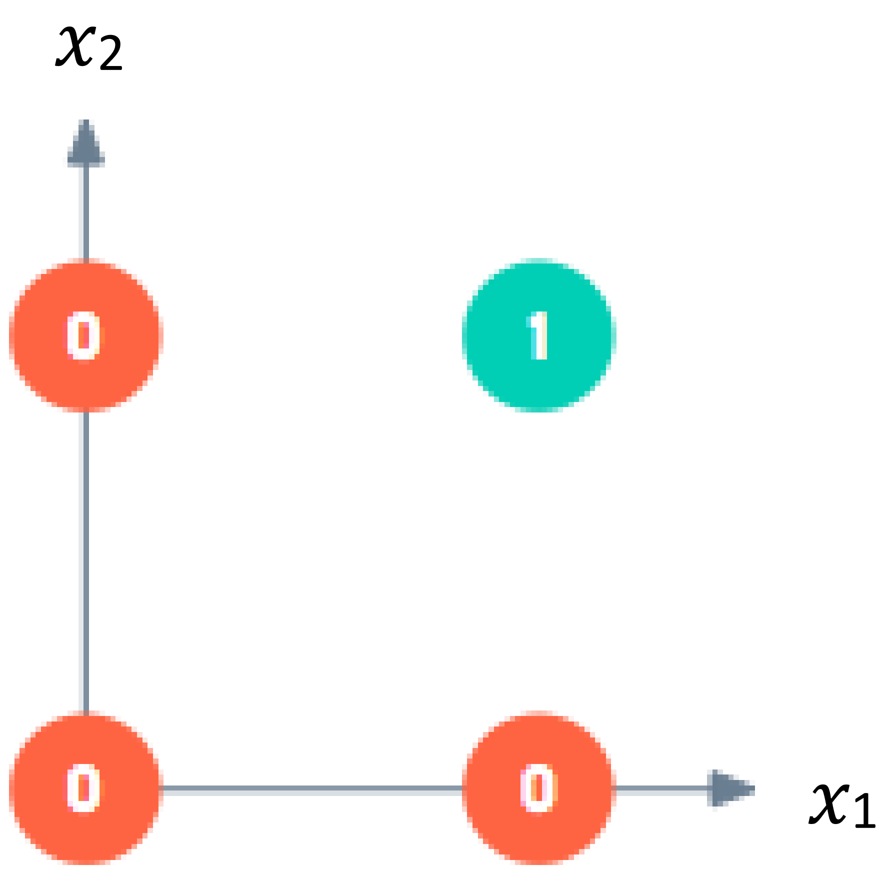


http://deepdriving.cs.princeton.edu/

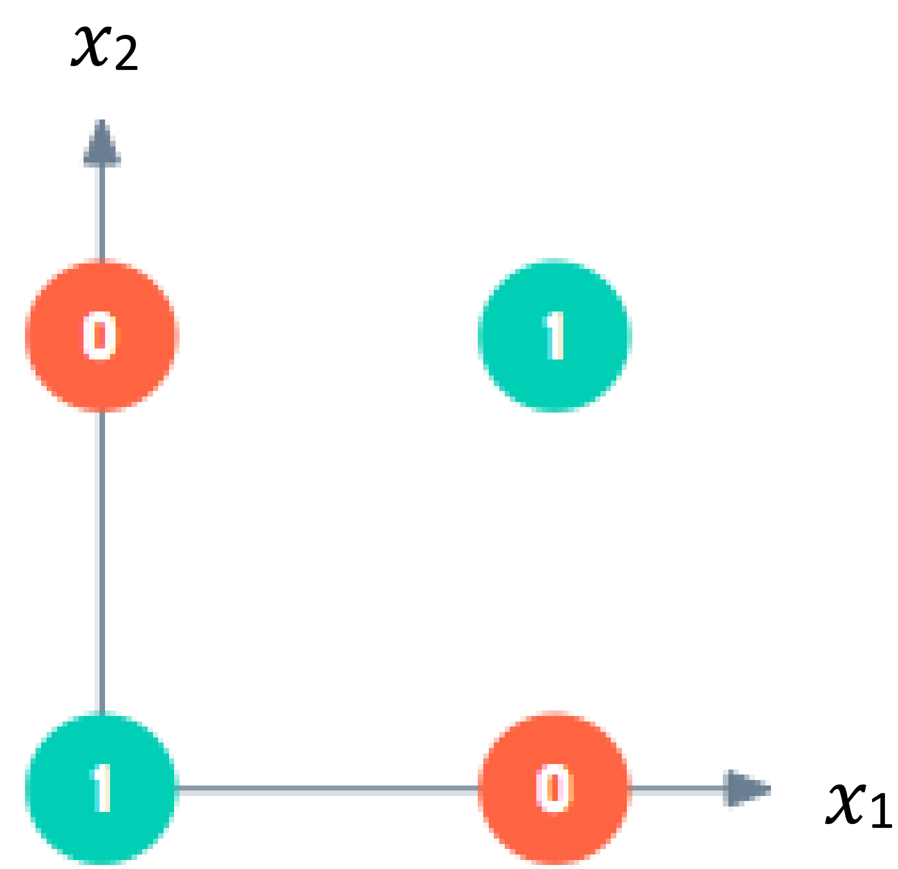
qHebb driving [Öfjäll et al. 2014]

Feedforward networks

Linear separability

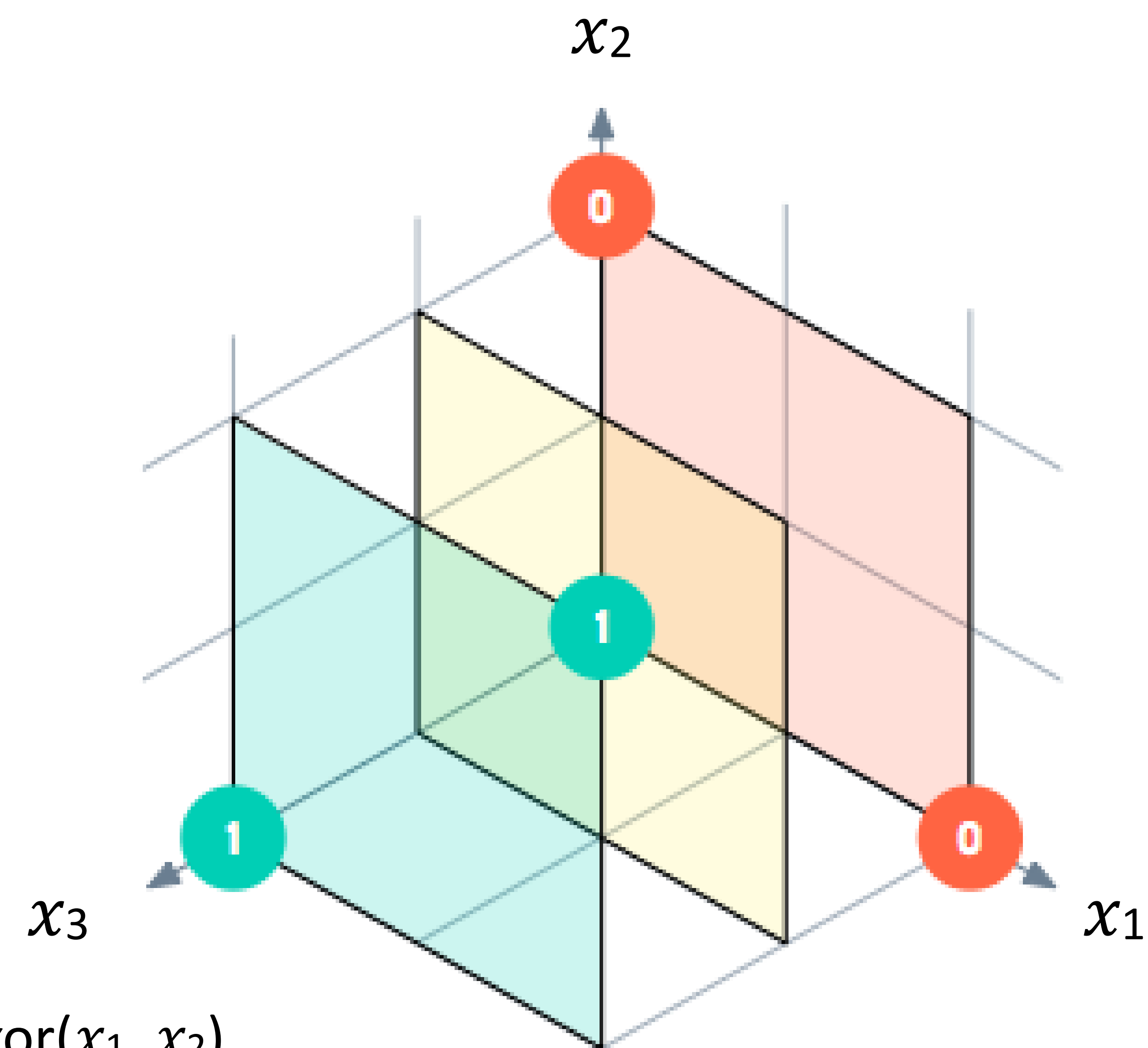


linearly separable



not linearly separable

New features to the rescue!



x_3 = xor(x_1, x_2)

How do we get new features?

We want to apply the linear model not to $\boldsymbol{x}$ directly but to a representation $\phi(\boldsymbol{x})$ of $\boldsymbol{x}$. How do we get this representation?

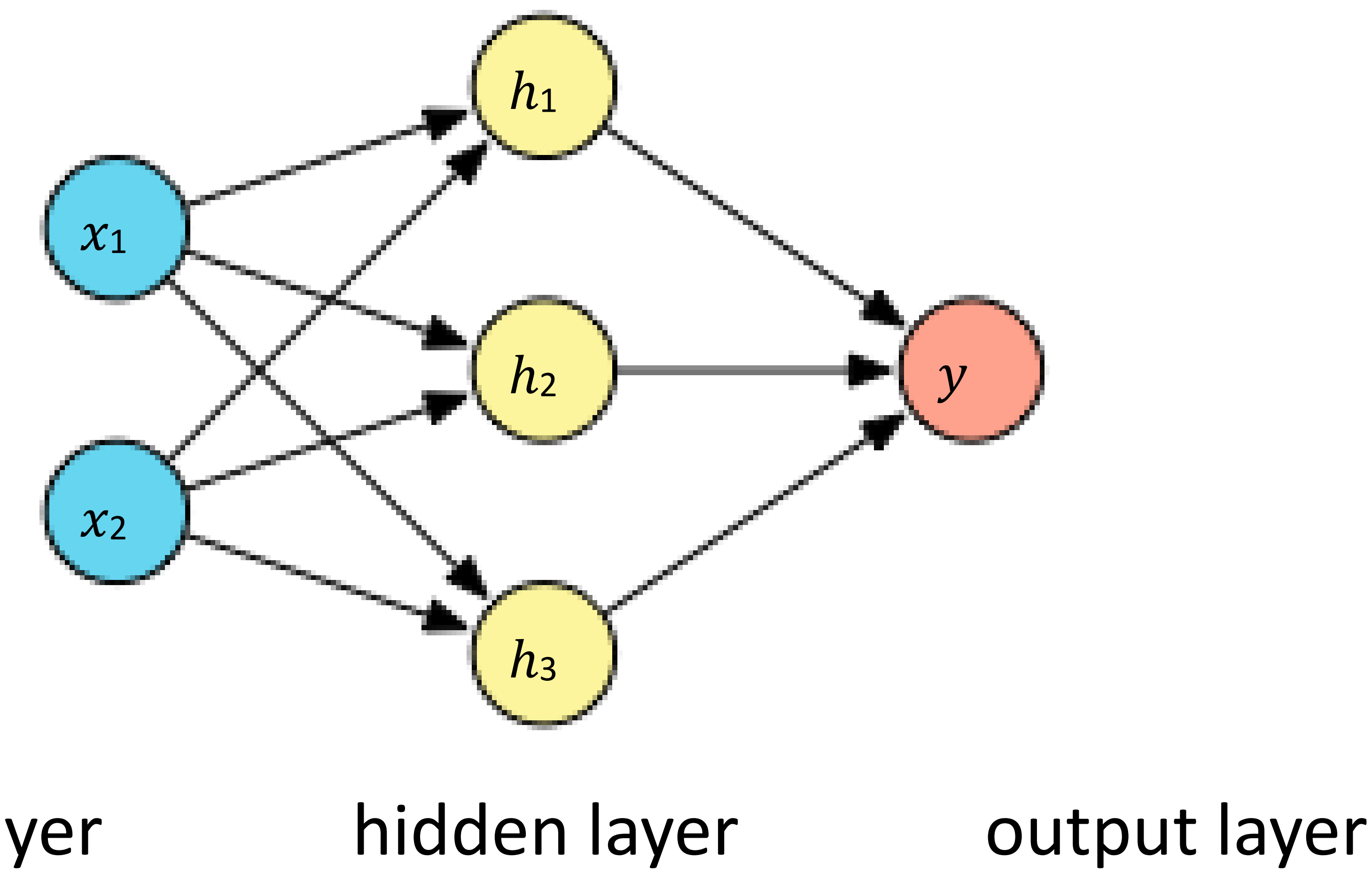
- Option 1. Manually engineer ϕ using expert knowledge.

  feature engineering

- Option 2. Make the model sensitive to parameters such that learning these parameters identifies a good representation ϕ.

  feature learning

Shapes of the parameter matrices



$\boldsymbol{H}$: (2, 3) $\boldsymbol{W}$: (3, 1)

Convolution

LINKÖPINGS UNIVERSITET

Apply networks to images

- What happens with $\mathbf{H}$ for image-sized input ?
- What happens with $\mathbf{H}$ and $\mathbf{W}$ for about same order of magnitude hidden units?
- Computational effort
- Overfitting

$$\mathbf{s} = \mathbf{H}\mathbf{x} =$$

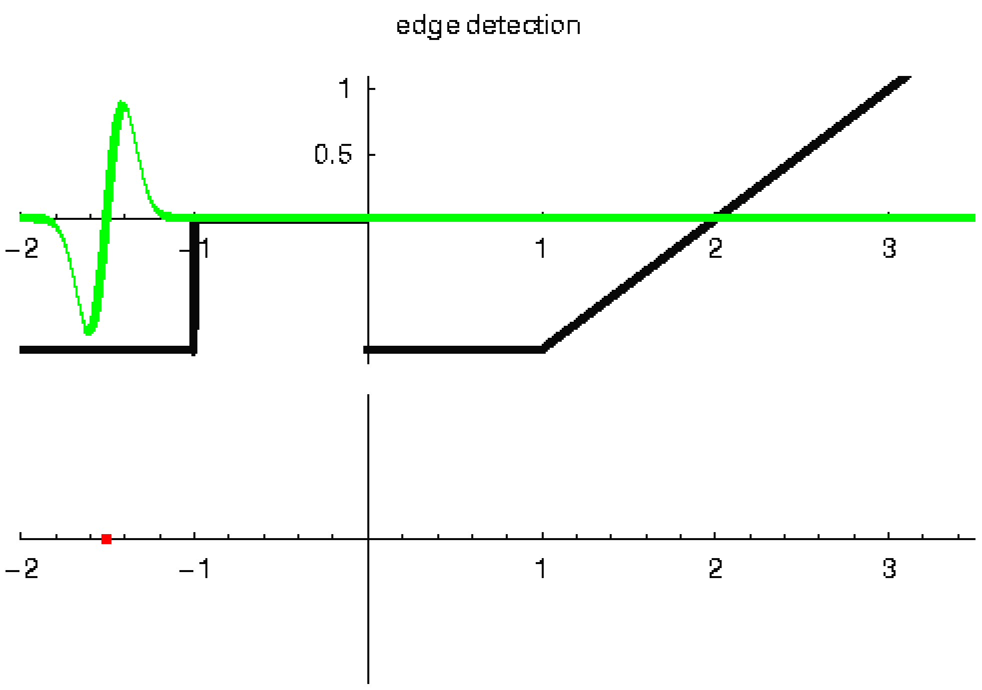
$$\begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \cdots & h_{1,n} \\ h_{2,1} & h_{2,2} & h_{2,3} & \cdots & h_{2,n} \\ h_{3,1} & h_{3,2} & h_{3,3} & \cdots & h_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{m,1} & h_{m,2} & h_{m,3} & \cdots & h_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Convolutional (neural) networks

- CNN [LeCun, 1989]
- suitable for data with known, grid-like topology
  - Time series
  - Images “tensors”
  - Medical data
- “Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.”

Convolution

- Separate lesson
- CNNs use correlation
- flipping irrelevant for learned coefficients
- 1D convolution: Toeplitz matrix
- Or circulant for periodic boundary conditions



http://bmia.bmt.tue.nl/education/courses/fev/course/notebooks/Convolution.html

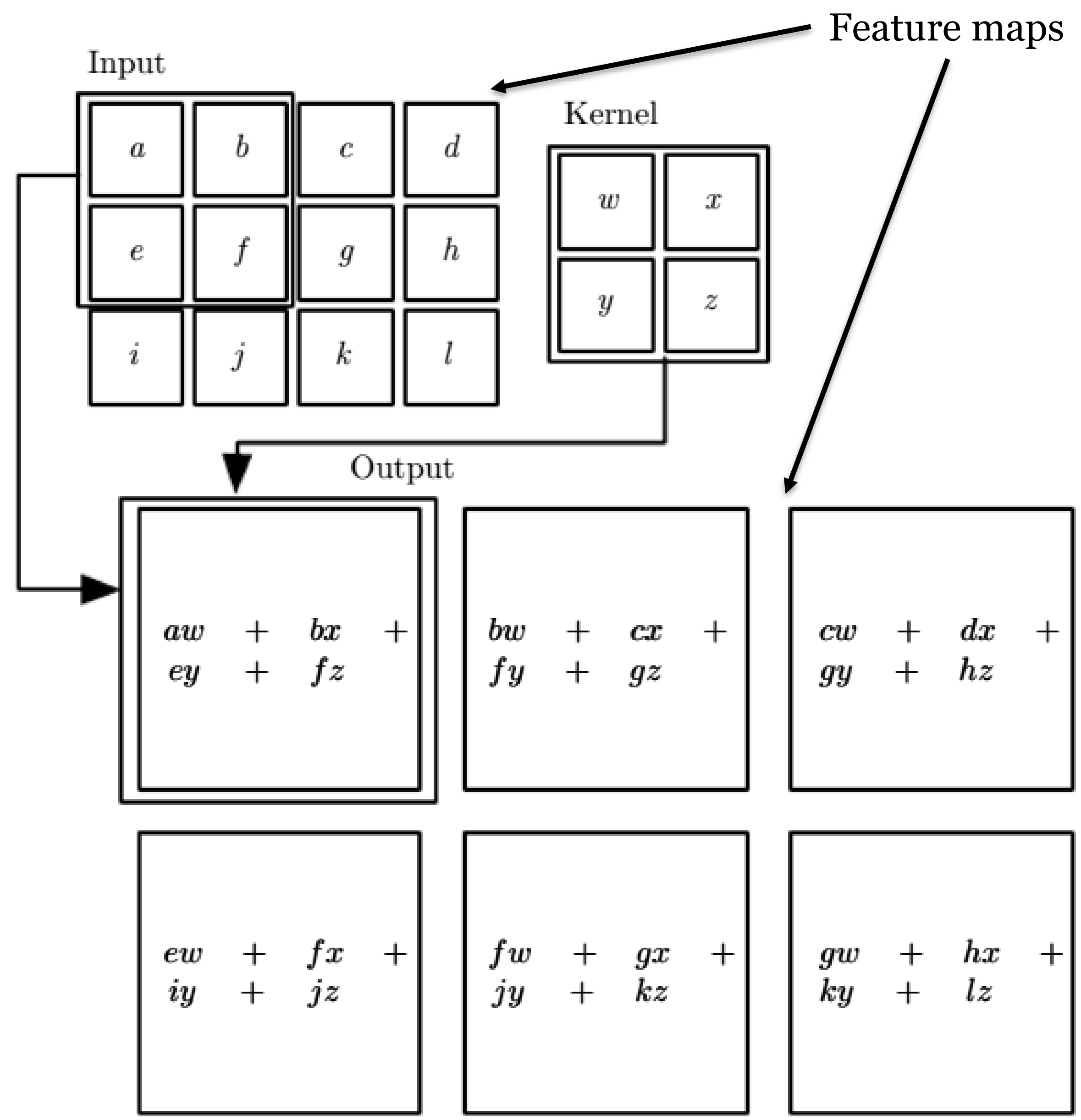
Convolution

- Separate lesson
- CNNs use correlation
- flipping irrelevant for learned coefficients
- 1D convolution: Toeplitz matrix
- Circulant if periodic boundary conditions
- Often: sparse

$$s = w * x = \begin{bmatrix} w_0 & w_{-1} & w_{-2} & 0 & 0 & \dots & 0 \\ w_1 & w_0 & w_{-1} & w_{-2} & 0 & \ddots & 0 \\ w_2 & w_1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & w_2 & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \ddots & w_{-2} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & w_{-1} \\ 0 & 0 & \dots & 0 & w_2 & w_1 & w_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Algorithmic

- 2D convolution: doubly block circulant
- Very sparse
- *Images* become *feature maps*
- If boundary conditions unknown, the feature map shrinks (‘*valid*’ in Matlab and Python



http://www.deeplearningbook.org/

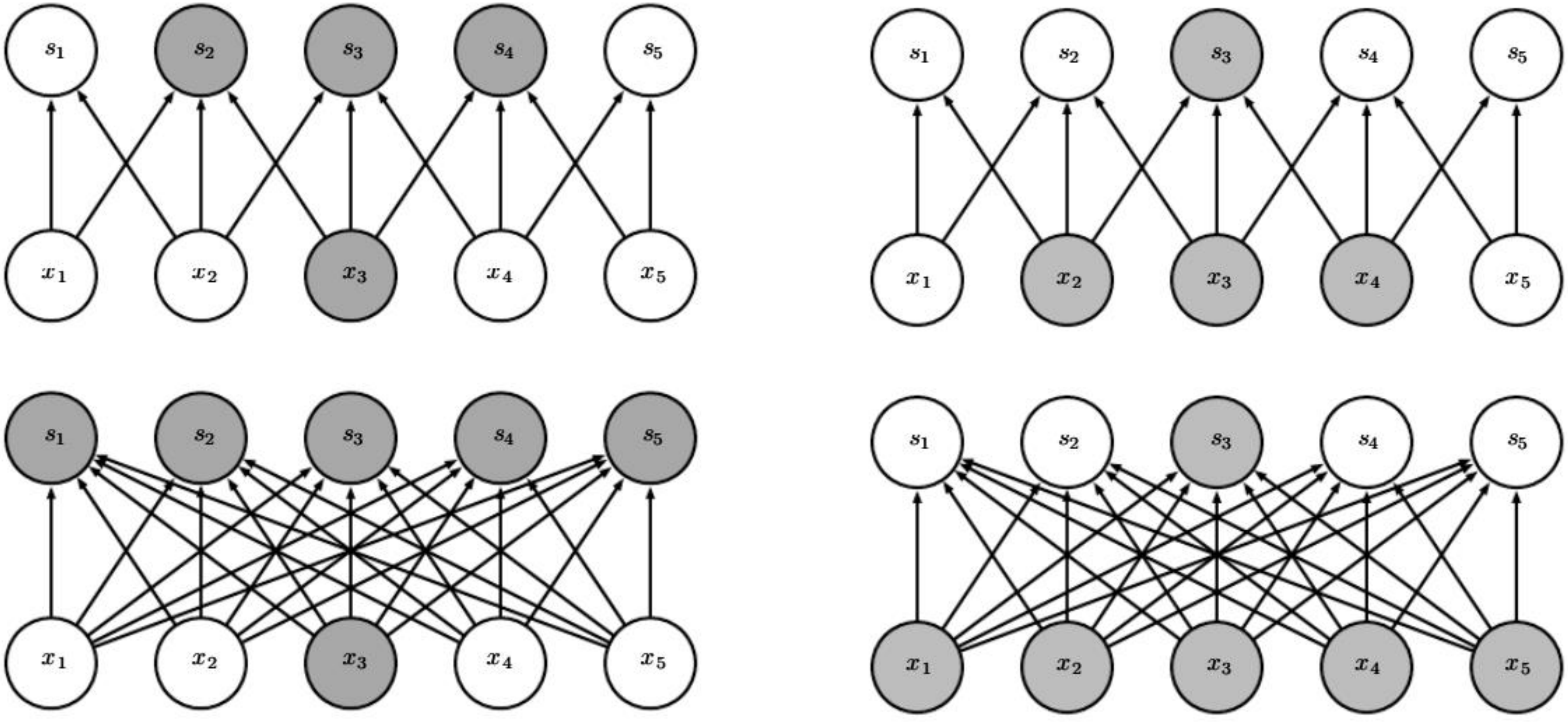
Convolutional Neural Networks

Motivation CNNs

1. sparse (and local) interaction
2. parameter sharing
3. equivariant representations

Sparse (and local) interaction

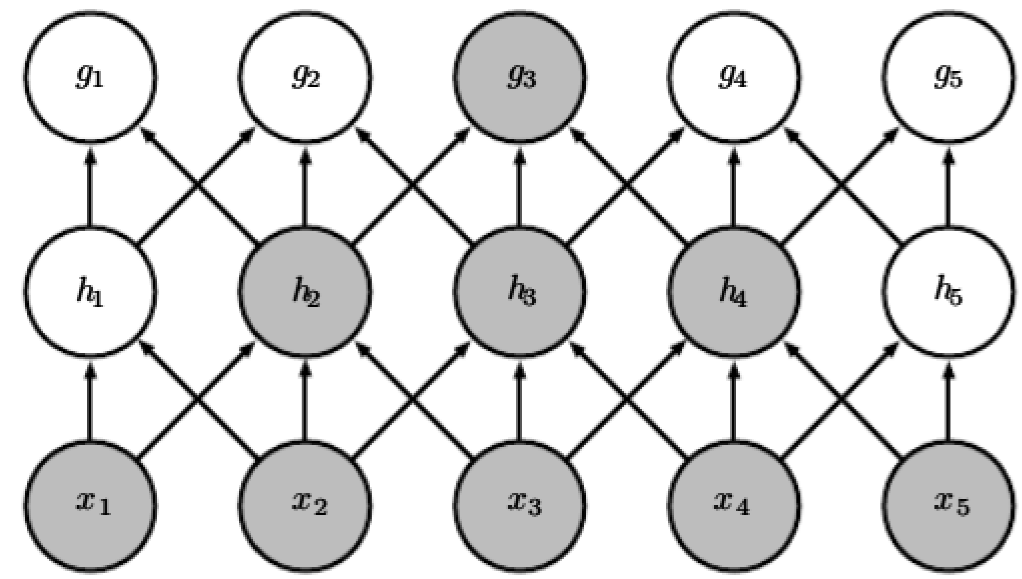
- kernel smaller than the input



http://www.deeplearningbook.org/

Sparse (and local) interaction

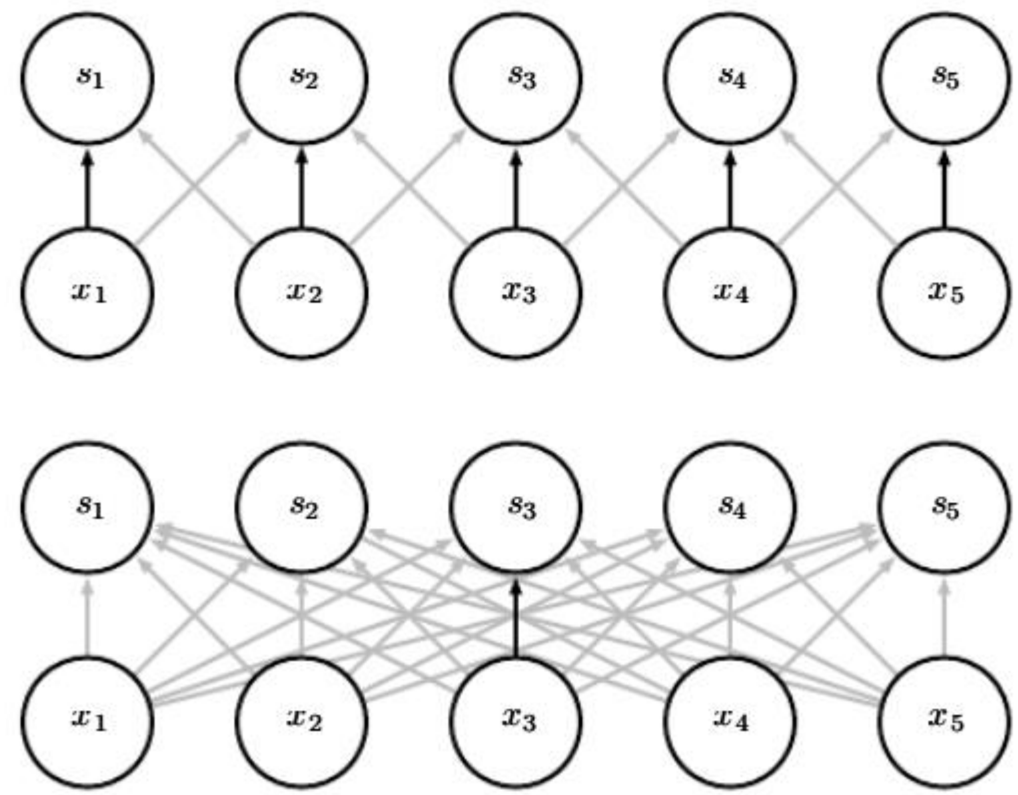
- kernel smaller than the input
  - fewer parameters
  - lower memory requirements
  - better statistical efficiency
  - fewer operations
- by increased depth indirectly connected to all input



http://www.deeplearningbook.org/

Parameter sharing

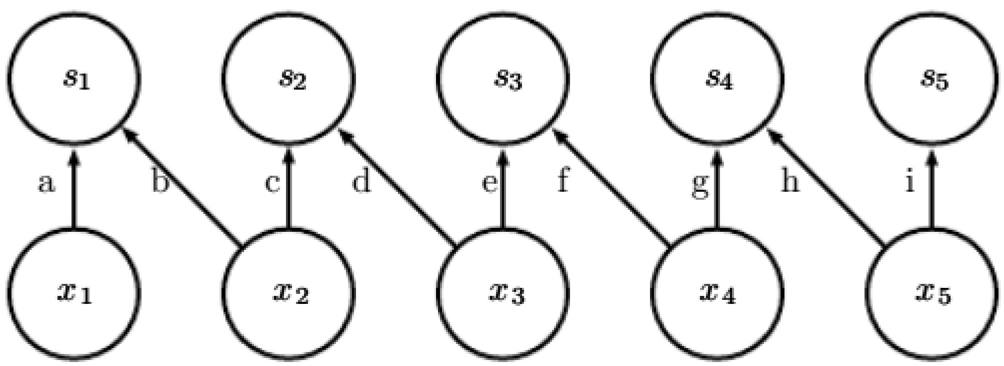
- tied weights
- reduced storage requirements
- but same time complexity
- sometimes sharing should be limited, e.g. cropped images

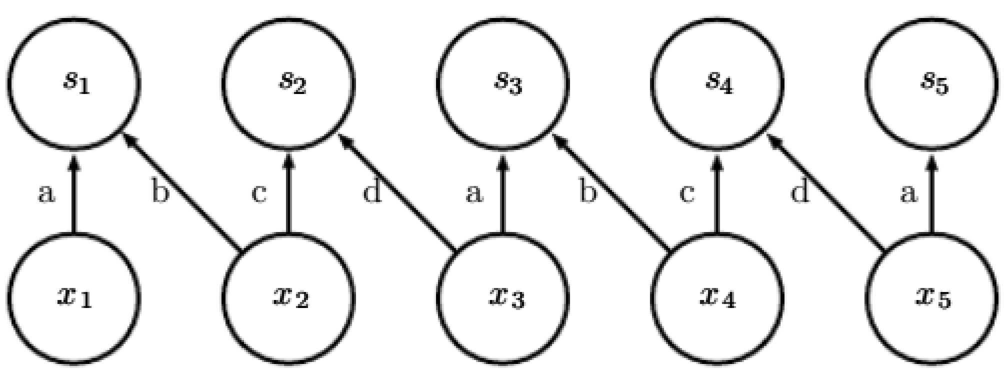


http://www.deeplearningbook.org/

Overview of options / convolution

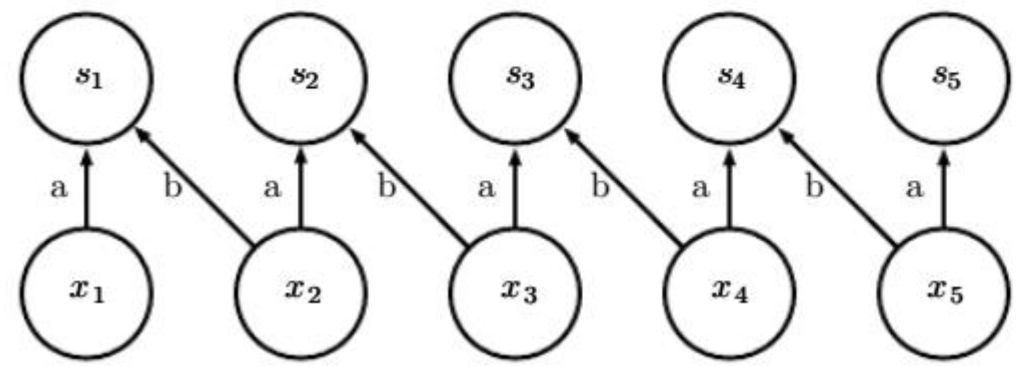
local connections unshared

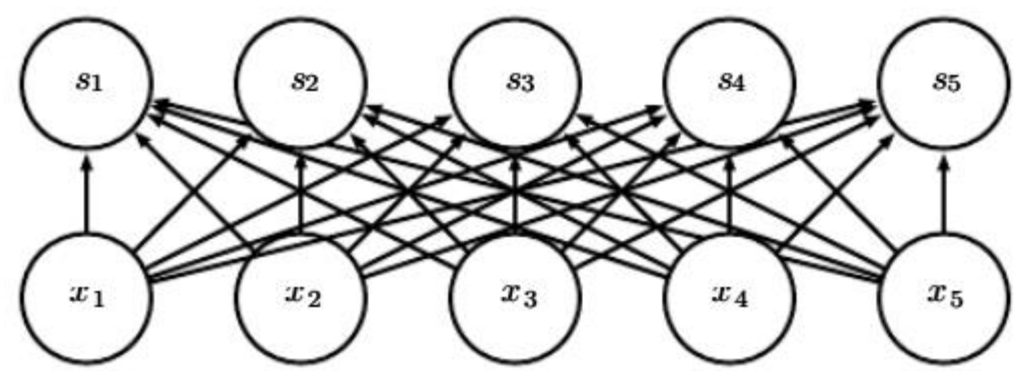




local connections tiled

local connections shared





full connections

http://www.deeplearningbook.org/

Invariance and Equivariance

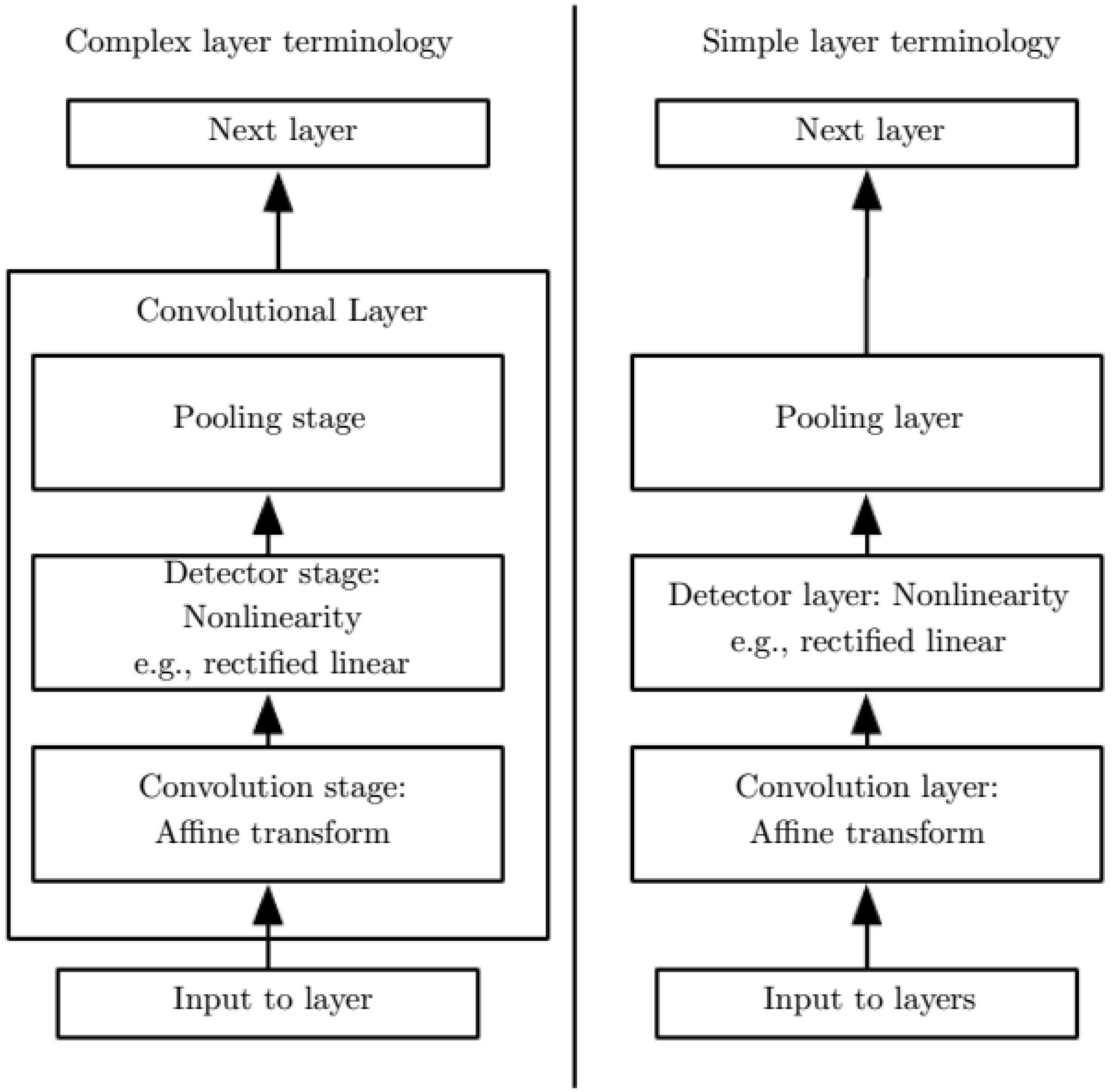
- a function f is invariant (under operation g) if
  - applying g to the input of f does not change its output
  - different inputs (modulo g) have different outputs
- a function f is equivariant (under operation g) if
  - applying g to the input of f changes its output by g'
  - different inputs have different outputs
- easy for discrete shift operations
- more tricky for rotation and scaling

Network Layers

LINKÖPINGS UNIVERSITET

Layers in CNNs

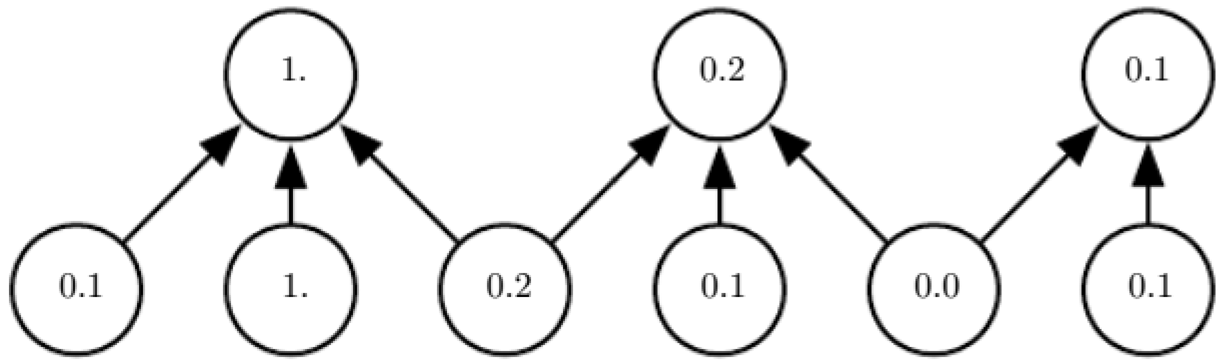
- each layer consists of three stages:
    1. (strided) convolutions to compute linear activation
    2. detector stage with activation
    3. pooling function



http://www.deeplearningbook.org/

Strides

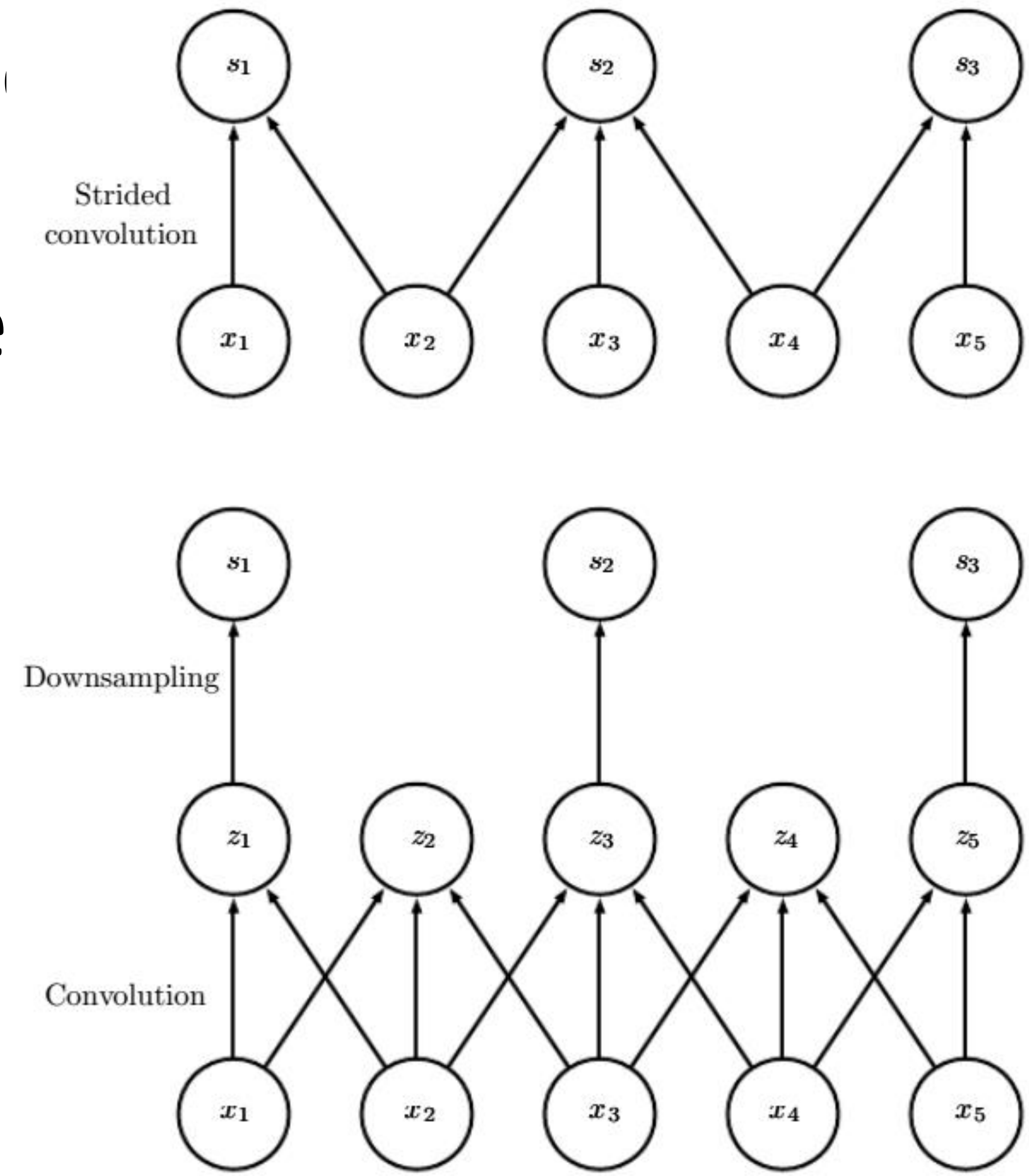
- pooling *s* pixels apart instead of every pixel (stride *s*)



http://www.deeplearningbook.org/

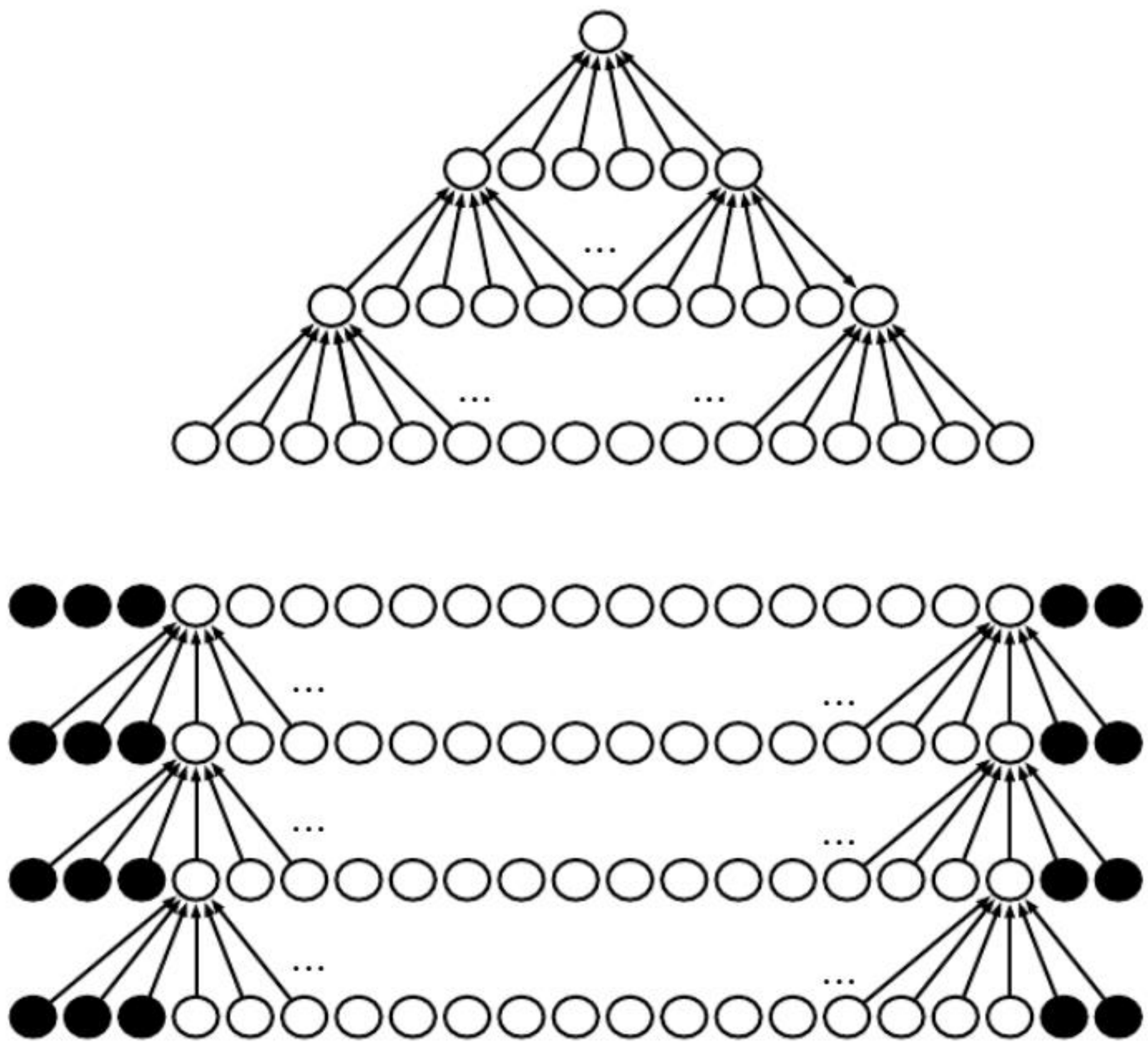
  - improved statistical efficiency
  - reduced memory requirements
  - handling inputs of varying size
  - but: pooling & strides complicate top-down processing (e.g. autoencoders)

Stride vs. sequential downsampling (cf. filterbanks/ wave



http://www.deeplearningbook.org/

Zero-padding



http://www.deeplearningbook.org/

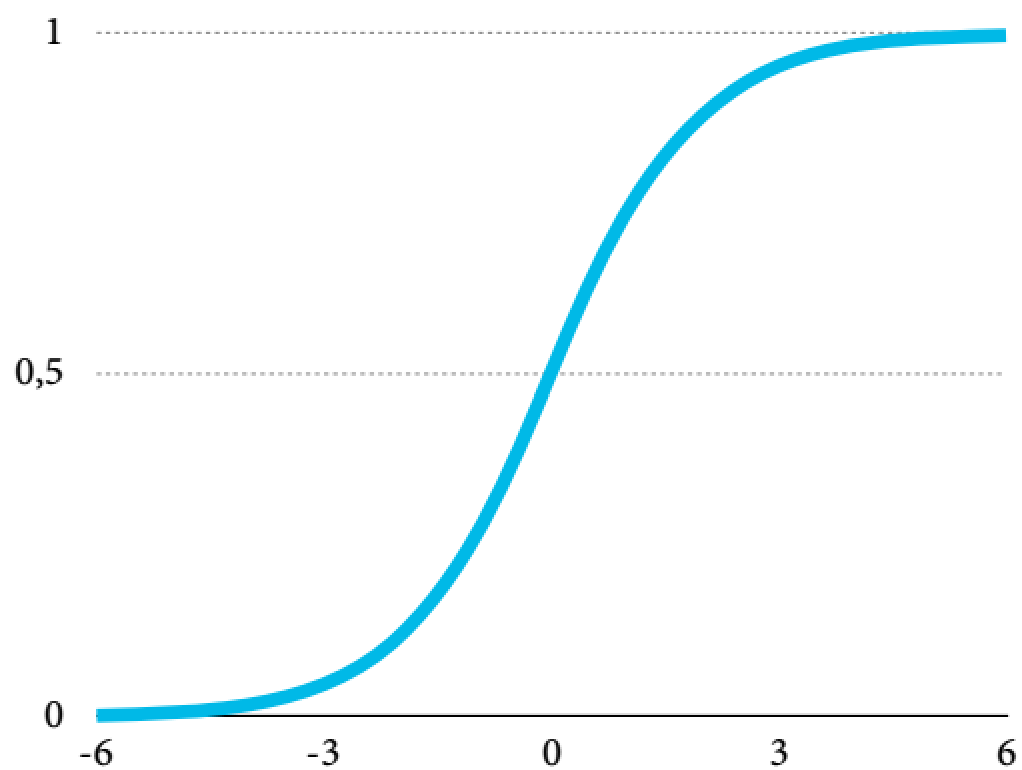
Bias terms

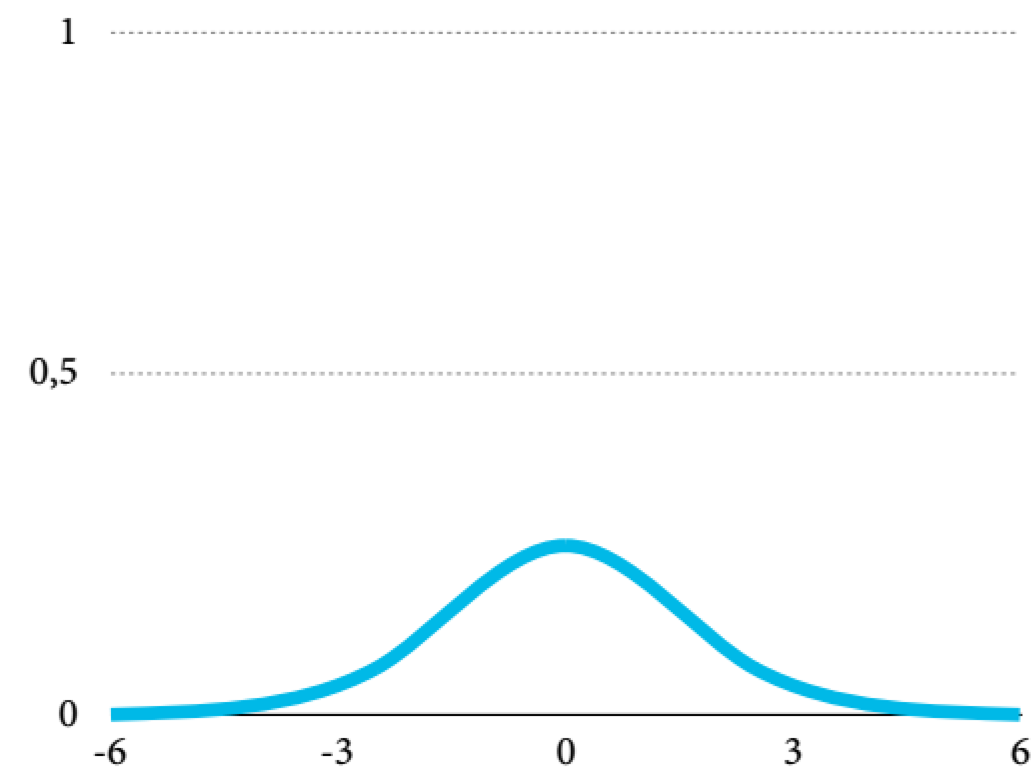
- Locally connected unshared – each unit own bias
- Tiled convolution – share biases in tiling pattern
- Shared convolution
  - share bias
  - separate bias at each location

  compensate differences in the image statistics

Activation Functions and Pooling

Logistic function



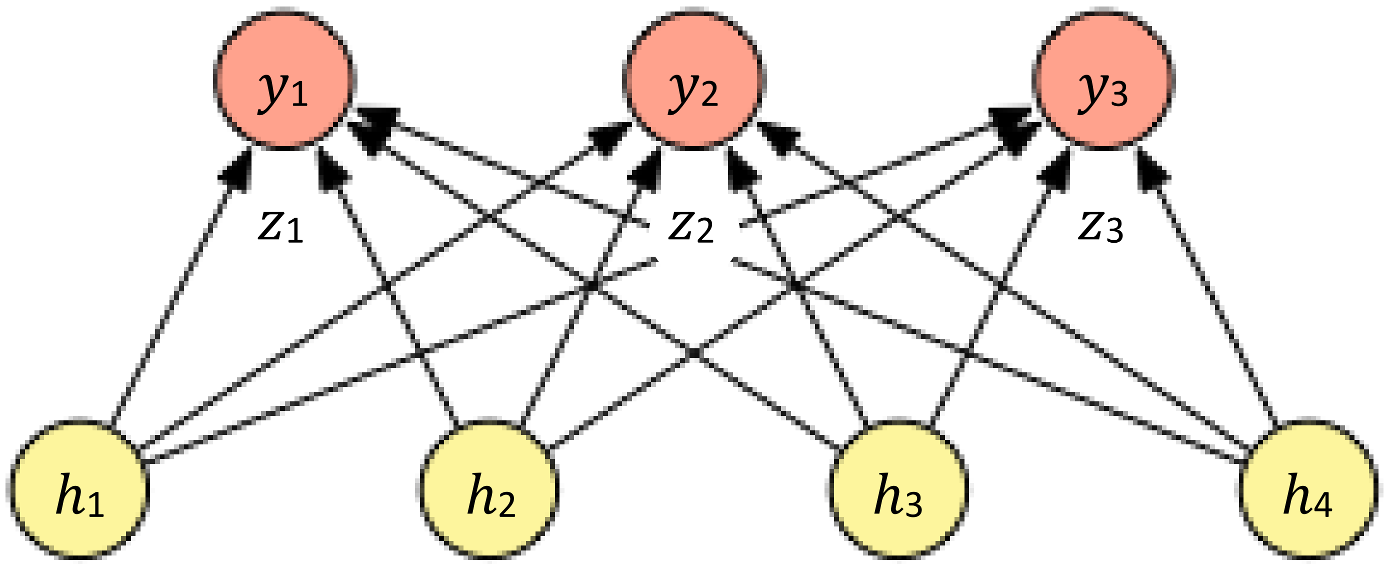


$$f(z) = \frac{1}{1 + e^{-z}}$$

$$\frac{\partial f}{\partial z} = f(z)(1 - f(z))$$

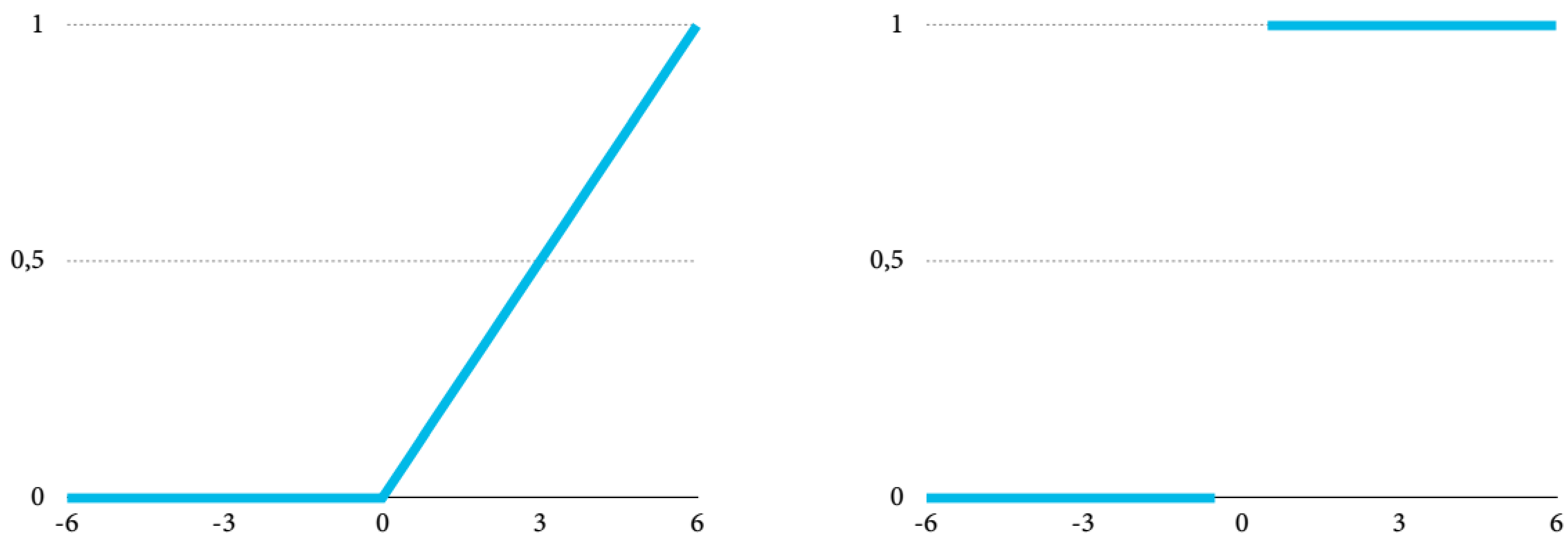
Softmax layer

$$y_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$



$$\frac{\partial y_i}{\partial z_j} = \begin{cases} y_i(1 - y_i) & i = j \\ -y_i y_j & i \neq j \end{cases}$$

Rectified linear units

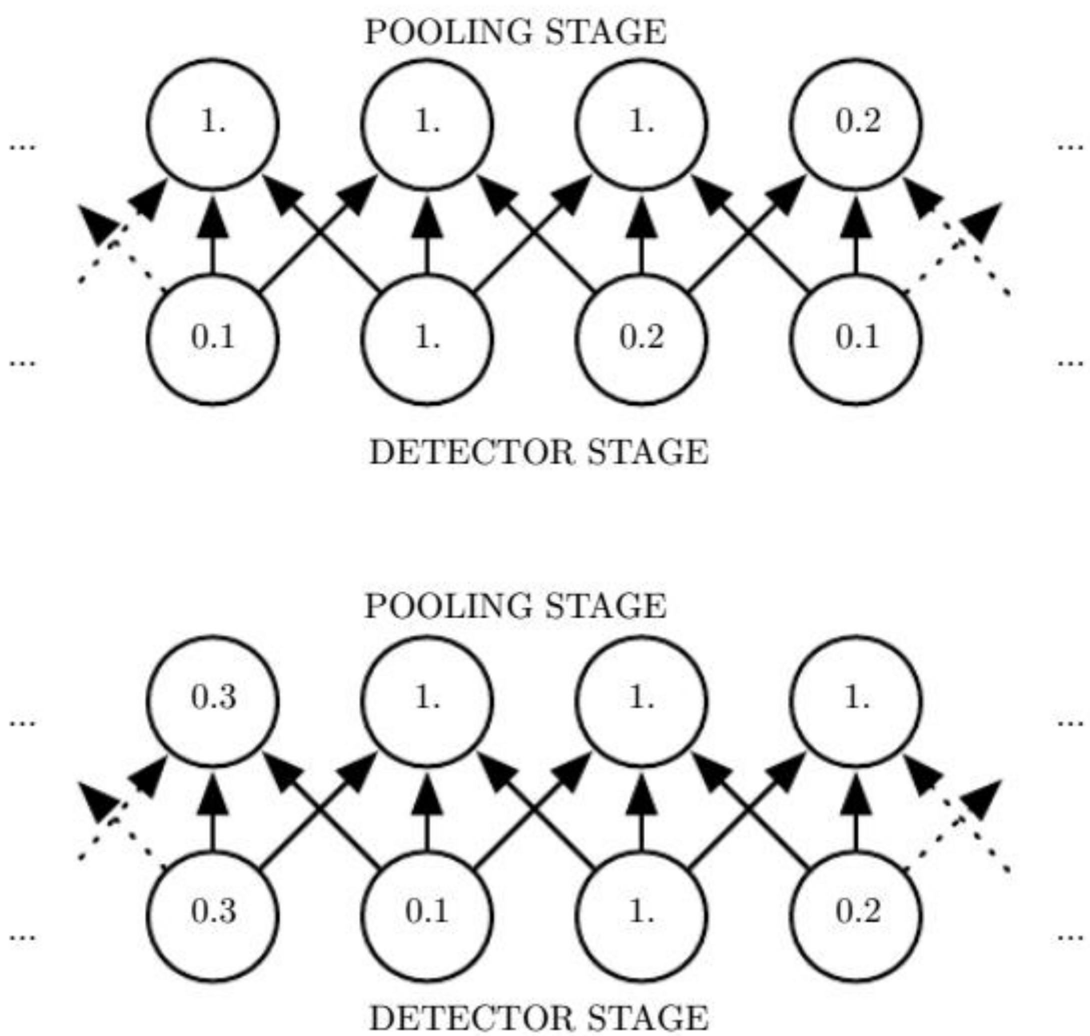


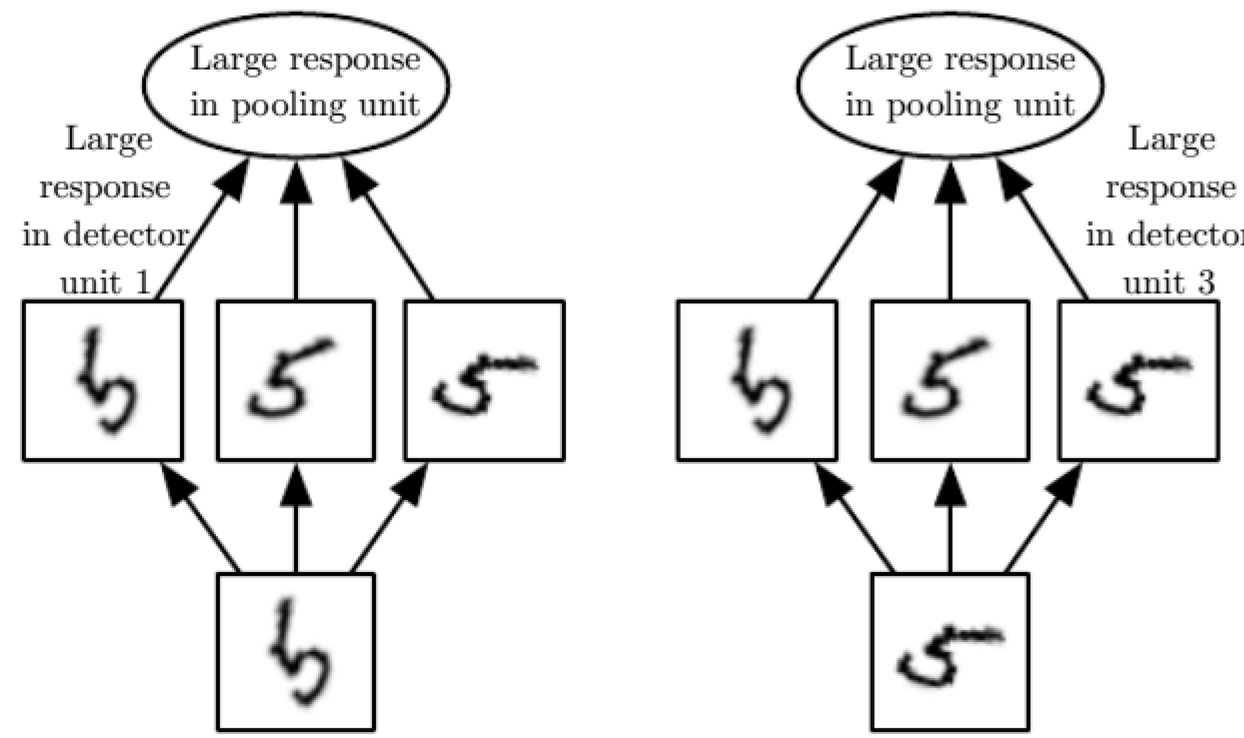
$$f(z) = \begin{cases} 0 & z \leq 0 \\ z & z > 0 \end{cases} \qquad \frac{\partial f}{\partial z} = \begin{cases} 0 & z \leq 0 \\ 1 & z > 0 \end{cases}$$

Pooling

- summary statistics of nearby outputs
  - max pooling [Zhou&Chellappa, 1988]
    maximum output in rectangular region
  - average in rectangular region
  - L2 norm of rectangular region
  - weighted average
    (based on distance from central position)
- approximately invariant to small translations

Pooling and invariance





Loss functions

Machine learning = optimization?

- Objective $J(\boldsymbol{\theta})$ / $\varepsilon(w)$ is minimized
- Expectation over some loss function L
- Ideal: expectation over *data distribution*
- Hope: empirical data (training set) gives the same parameters (empirical risk minimization)
- *Hypothesis*: test set drawn from the same distribution
- Models with high *capacity* memorize training set (overfitting)
- Optimization: direct minimization of objective

Maximum likelihood estimation

- Family of probability distributions $P(\boldsymbol{X};\boldsymbol{\theta})$ that assign a probability to any sequence $\boldsymbol{X}$ of N examples.
- The maximum likelihood estimator for $\boldsymbol{\theta}$ is defined as

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{X};\boldsymbol{\theta})$$

- Assume that the examples are mutually independent and identically distributed, this can be rewritten as

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{N} P\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right) = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log P\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right)$$

- If assuming Gaussian noise in $P()$: sum of squares

Conditional log-likelihood

- *Supervised learning*: learn a conditional probability distribution over target values $\boldsymbol{y}$, given features $\boldsymbol{x}$.
- The assumption that the samples are i.i.d. yields

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log P\left(\boldsymbol{y}^{(i)} \,\middle|\, \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$$

- Same as minimizing *cross-entropy*
- Principled way to derive the cost function (incl. L2)

$$\mathrm{cost}\left(\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}\right) = -\frac{1}{N} \sum_{i=1}^{N} \log P\left(\boldsymbol{y}^{(i)} \,\middle|\, \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$$

Surrogate loss function

- Gradient descent does not allow for discontinuous loss functions
- Test error with original loss might be lower for training with surrogate loss
- Example: 0-1 loss for class membership (one-hot) might be replaced with log-likelihood (cross-entropy)
  - Continuously differentiable
  - Continues pushing classes apart even if empirical loss on training set is zero

Cross-entropy cost function

- The output of a logistic unit can be interpreted as the conditional probability $P(y_i = 1 \mid \boldsymbol{x})$ for a binary random variable y_i.
- The natural error function for a logistic unit is the negative log probability of the correct output:

$$C = \begin{cases} -\ln h(\mathbf{x}_i) & \text{if } y_i = 1 \\ -\ln(1 - h(\mathbf{x}_i)) & \text{if } y_i = 0 \end{cases}$$

- This is usually written as

$$C = -(y_i \ln h(\mathbf{x}_i) + (1 - y_i)\ln(1 - h(\mathbf{x}_i)))$$

Cross-entropy cost function

- The output of a soft-max unit can be interpreted as the conditional probability $P(\boldsymbol{y}_i = (0, 1, 0, \ldots) \mid \boldsymbol{x})$ for an one-hot random vector $\boldsymbol{y}_i$.

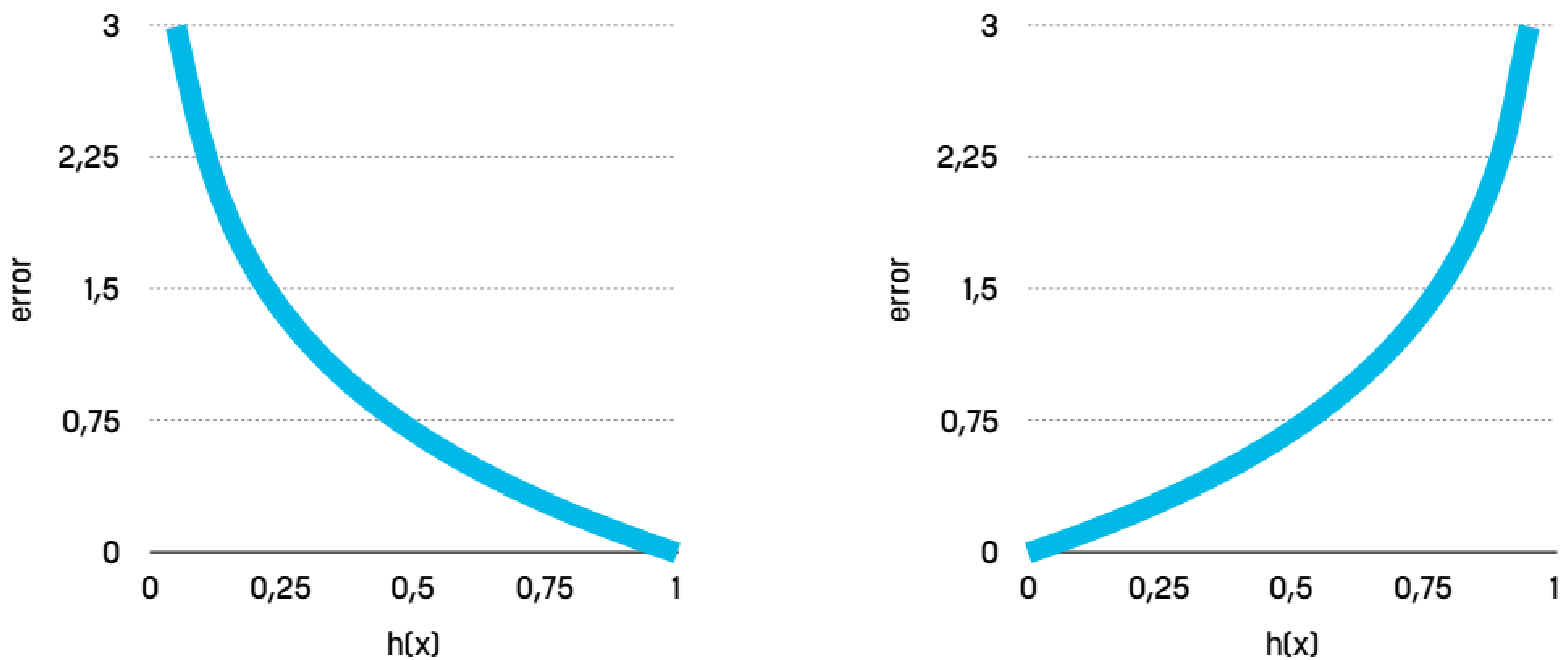
$$P(\mathbf{y}|\mathbf{x}) = \Pi_k h_k(\mathbf{x})^{y_k}$$

- The natural error function for a soft-max unit is the negative log probability (cross-entropy):

$$-\ln P(\mathbf{y}_n|\mathbf{x}_n; n = 1 \ldots N) = -\sum_n \sum_k y_{kn} \ln h_k(\mathbf{x}_n)$$

- Note that often the soft-max is considered to be the cost

Sigmoid and cross-entropy balance each other



$y = 1$ $y = 0$

$$\frac{\partial C}{\partial z} = -y_i(1 - f(z)) + (1 - y_i)f(z) = f(z) - y_i$$