

University of Stuttgart  
Faculty of Computer Science, Electrical Engineering and Information Technology  
Institute of Signal Processing and System Theory  
Prof. Dr.-Ing. B. Yang

Slides to the lecture

## **Deep Learning**

[www.ISS.uni-stuttgart.de](http://www.ISS.uni-stuttgart.de)

# Organization of the course

## When and where

- Monday, 15:45-17:15 and Thursday, 8:00-9:30
- lecture hall V47.03

## Tools in lecture

- tablet PC + projector
- ≈ 300 prepared slides + writing on tablet PC
- no complete script, but video recording of lecture

## On ILIAS ([ilias3.uni-stuttgart.de](http://ilias3.uni-stuttgart.de)) you will find

- color slides of the lecture
- video recordings of the lecture
- animations and demonstrations
- programming homeworks
- old exam problems + solutions
- 2 books and more than 70 papers about deep learning

## Goals Learn

- machine learning basics
- theory, architectures and applications of deep neural networks
- programming in Python/Tensorflow

## Prerequisites

- The course "Advanced mathematics for signal and information processing" (AM)
  - advanced vector and matrix computations
  - probability theory (probability, random variables, stochastic processes)
  - optimizationis highly recommended.
- The course "Detection and pattern recognition" (DPR) is useful, but *not* mandatory.  
There is almost no overlap between DPR and this course.

## New concept of the course

- lecture
- no traditional calculation exercise
- programming
  - introduction into Python and Tensorflow
  - programming practice (homeworks) → minilab for everyone
- expert talks about deep learning in different applications

All topics, also the programming part and the expert talks, are *relevant* for the exam!

## Literature

Machine learning:

- [DHS04] R. O. Duda, P. E. Hart, D. G. Stork, "Pattern classification", John Wiley & Sons, 2. edition, 2004

- [B06] C. Bishop, "Pattern recognition and machine learning", Springer, 2006 (online available, google it.)

Deep learning:

- [GBC16] I. Goodfellow and Y. Bengio and A. Courville, "Deep learning", [www.deeplearningbook.org](http://www.deeplearningbook.org), 2016.

- [N18] A. Ng, "Machine Learning Yearning", [www.mlyearning.org](http://www.mlyearning.org), 2018.

- [Ilias] 2 books and > 70 papers about deep learning

Matrix computation and optimization:

- [PP08] K. B. Petersen, M. S. Pedersen, "The matrix cookbook", 2008 (online available, google it.)

# Content (1)

## 1. Introduction

- 1.1 What is machine learning?
- 1.2 What is deep learning?
- 1.3 Examples

## 2. Tools for deep learning

- 2.1 Software
- 2.2 Hardware
- 2.3 Datasets

## 3. Machine learning basics

- 3.1 Linear algebra
- 3.2 Random variable and probability distribution
- 3.3 Kullback-Leibler divergence and cross entropy
- 3.4 Probabilistic framework of machine learning

## 4. Dense neural networks

- 4.1 Neuron
- 4.2 Layer of neurons

## Content (2)

- 4.3 Feedforward neural networks
- 4.4 Activation function
- 4.5 Universal approximation
- 4.6 Loss and cost function
- 4.7 Training
- 4.8 Implementation of DNNs in Python
- 5. Advanced optimization techniques
  - 5.1 Difficulties in optimization
  - 5.2 Momentum
  - 5.3 Learning rate schedule
  - 5.4 Input and batch normalization
  - 5.5 Parameter initialization
  - 5.6 Improved model

## Content (3)

### 6. Overfitting and regularizations

- 6.1 Model capacity and overfitting
- 6.2 Weight norm penalty
- 6.3 Early stopping
- 6.4 Data augmentation
- 6.5 Ensemble learning
- 6.6 Dropout
- 6.7 Hyperparameter optimization

### 7. Convolutional neural networks

- 7.1 Convolutional layer
- 7.2 Modified convolutions
- 7.3 Polling and unpooling layer
- 7.4 Deconvolutional layer
- 7.5 Flatten layer
- 7.6 Global average pooling layer
- 7.7 Architecture of CNNs

## Content (4)

- 8. Recurrent neural networks
  - 8.1 Recurrent layer and recurrent neural network
  - 8.2 Bidirectional recurrent neural network
  - 8.3 Long short-term memory
- 9. Unsupervised and generative models
  - 9.1 Autoencoder
  - 9.2 Variational autoencoder
  - 9.3 Generative adversarial network
- 10. Popular networks and modules
  - 10.1 For image classification
  - 10.2 For object detection
  - 10.3 For image segmentation
  - 10.4 For generative tasks
- 11. Further topics and outlook
  - 11.1 Important but unaddressed topics
  - 11.2 AI vs. human intelligence

## Mathematical notations and symbols

- **scalar**:  $x, A, \alpha$

- column **vector**:  $\underline{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = [x_i] \in \mathbb{R}^N$  with element  $x_i = [\underline{x}]_i$

- **matrix**:  $\mathbf{A} = [a_{ij}]_{1 \leq i \leq M, 1 \leq j \leq N} = [a_{ij}] \in \mathbb{R}^{M \times N}$  with element  $a_{ij} = [\mathbf{A}]_{ij}$

- 3D, 4D **tensor**:  $\mathbf{A} = [a_{ijk}], \mathbf{A} = [a_{ijkl}]$

- **transpose** of vector and matrix:  $\underline{x}^T = [x_1, \dots, x_N], \mathbf{A}^T = [a_{ji}]_{ij}$

- **determinant** of a square matrix  $\mathbf{A}$ :  $|\mathbf{A}|$

- **inverse** of a square matrix  $\mathbf{A}$ :  $\mathbf{A}^{-1}$

- **trace** of a square matrix  $\mathbf{A}$ :  $\text{tr}(\mathbf{A}) = \sum_i a_{ii}$

## List of acronyms (1)

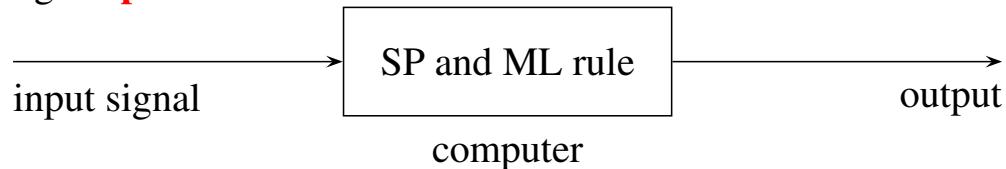
AE	<u>autoencoder</u>
AI	<u>artificial intelligence</u>
BRNN	<u>bidirectional recurrent neural network</u>
CBAM	<u>convolutional block attention module</u>
cGAN	<u>conditional generative adversarial network</u>
CL	<u>convolutional layer</u>
CNN	<u>convolutional neural network</u>
DL	<u>deep learning</u>
DNN	<u>deep neural network</u>
DRL	<u>deep reinforcement learning</u>
ELU	<u>exponential linear unit</u>
GAN	<u>generative adversarial network</u>
GAP	<u>global average pooling</u>
Grad-CAM	<u>gradient-weighted class activation mapping</u>
ILSVRC	<u>ImageNet large scale visual recognition challenge</u>
KLD	<u>Kullback-Leibler divergence</u>
LSTM	<u>long short-term memory</u>

## List of acronyms (2)

MAE	<u>m</u> ean <u>a</u> bsolute <u>e</u> rror
MSE	<u>m</u> ean <u>s</u> quare <u>e</u> rror
ML	<u>m</u> achine <u>l</u> earning
NAS	<u>n</u> eural <u>as</u> earch
PDF	<u>p</u> robability <u>d</u> ensity <u>f</u> unction
PMF	<u>p</u> robability <u>m</u> ass <u>f</u> unction
ReLU	<u>r</u> ectifier <u>l</u> inear <u>unit</u>
ResNet	<u>r</u> esidual <u>n</u> etwork
RL	<u>r</u> einforcement <u>l</u> earning
RNN	<u>r</u> ecurrent <u>n</u> eural <u>n</u> etwork
RV	<u>r</u> andom <u>v</u> ariable
SE	<u>s</u> queeze and <u>e</u> cititation
SGD	<u>s</u> tochastic <u>gd</u> escent
SP	<u>s</u> ignal <u>p</u> rocessing
t-SNE	<u>t</u> -distributed <u>s</u> tochastic <u>n</u> eighbor <u>e</u> mbedding
TTS	<u>t</u> ext-to-speech
VAE	<u>v</u> ariational <u>a</u> uto <u>e</u> ncoder
YOLO	<u>y</u> ou <u>o</u> nly <u>l</u> ook <u>o</u> nce

## Signal processing (SP) and machine learning (ML)

Process a given **input signal** according to a certain **rule** (program) and return the corresponding **output**.



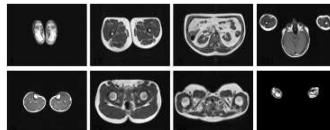
1D signal



2D signal

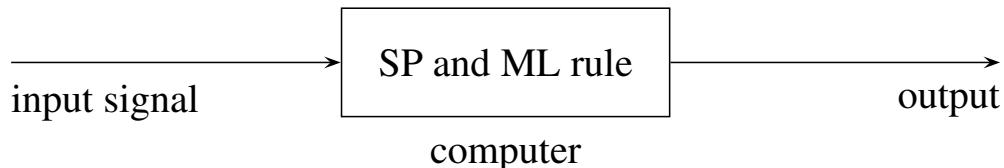


3D signal



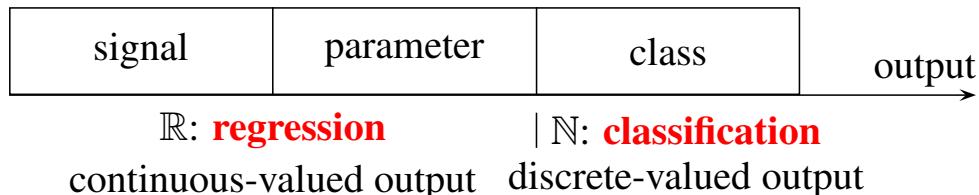
- Task of the computer: SP and ML according to the rule.
- *Your task:* Design the SP and ML rule.

## Different types of output



Depending on the application, the desired output of SP and ML, the quantity of interest, can be of different types:

- From signal to **signal** a sequence/array/tensor of numbers in  $\mathbb{R}$
- From signal to **parameter** a few numbers in  $\mathbb{R}$ , e.g. 1.23m for range
- From signal to **class** a number in  $\mathbb{N}$ , e.g. male/female



## E1.1: Different types of output

From signal to signal:

- a) Digital filter: from time-domain signal  $x(n)$  to time-domain signal  $y(n)$
- b) Fourier analysis: from time-domain signal  $x(n)$  to frequency-domain signal  $X(j\omega)$

From signal to parameter:

- c) Synchronization by correlation: position of  $x(n)$  in  $y(n)$
- d) Radar: from signal  $x(n)$  to distance/velocity/direction of target

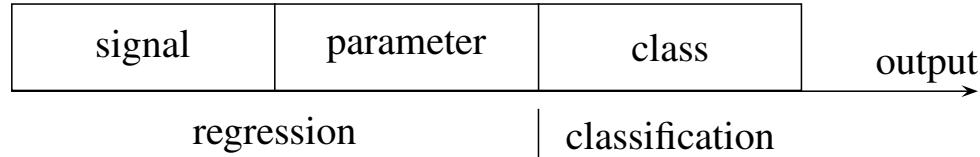
From signal to class:

- e) Radar: from signal  $x(n)$  to type of target (car, bike, pedestrian, ...)
- f) Speech recognition, speaker identification, image recognition, ...

a), b)

c), d)

e), f)



## Signal processing vs. machine learning

There are two major approaches to design the processing rule "?".



**Signal processing (SP)** is traditionally **model-based**:

There is a **signal model**, a mathematical description of the input signal as a function of the desired output. The signal processing rule is derived from the signal model.

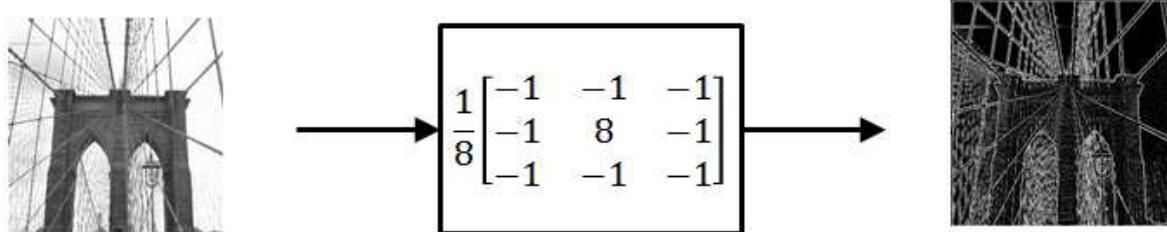
**Machine learning (ML)** is **learning-based** or **data-driven**:

There is no signal model because the relationship between the input signal and desired output is too complicated. The processing rule is learned from examples.

We humans learn to recognize, speak, walk, calculate etc. from examples.

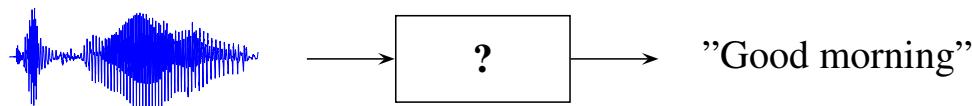
## E1.2: Model-based signal processing

- a) Delay estimation uses the signal model  $x(t) = As(t - \tau)$  between the transmitted signal  $s(t)$  and received signal  $x(t)$ .
- b) Radar distance estimation uses the signal model distance =  $2 \cdot \text{delay} \cdot \text{speed}$ .
- c) Radar velocity estimation uses the Doppler effect as signal model.
- d) Channel estimation in digital communication uses the FIR channel model  $x(n) = \sum_i h_i s(n - i) + z(n)$  between the transmitted signal  $s(n)$  and received signal  $x(n)$ .
- e) Edge detection by highpass filtering relies on the knowledge that edges correspond to high-frequency components.

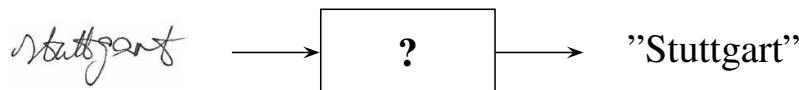


### E1.3: Machine learning (1): Classification

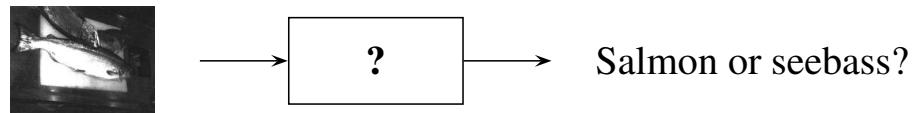
a) Speech recognition



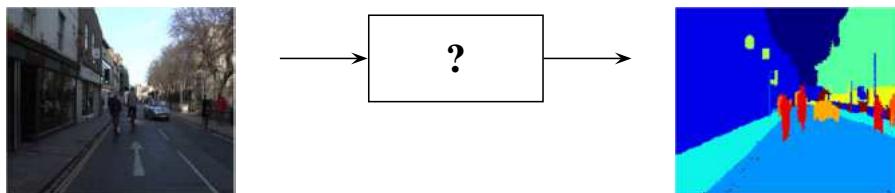
b) Optical character recognition (OCR)



c) Fish sorting in a fish-packing plant



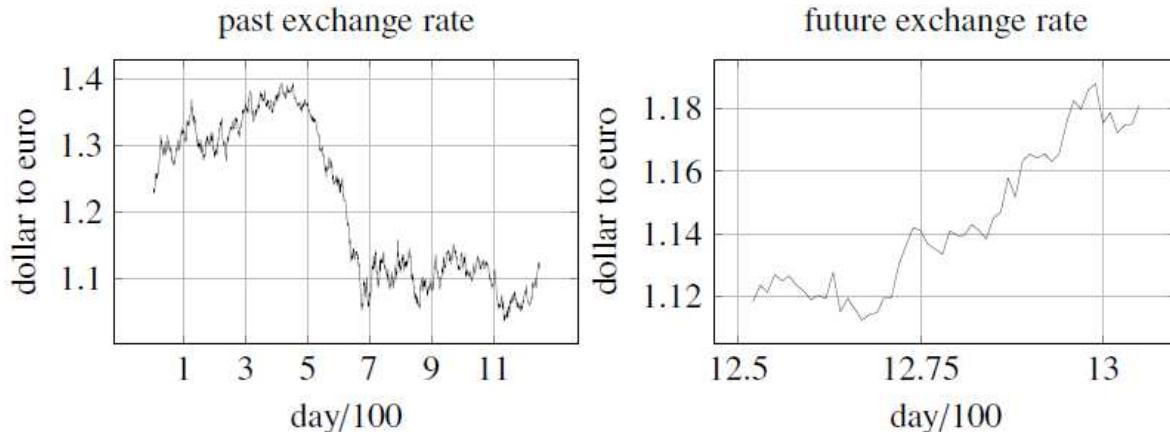
d) Semantic image segmentation (pixelwise classification) for autonomous driving



### E1.3: Machine learning (2): Regression

e) Prediction of exchange rate:

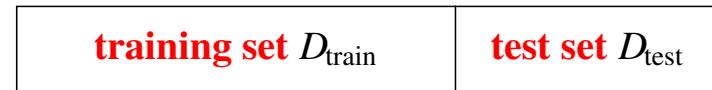
Predict the future exchange rate (time series) based on the past one (time series).



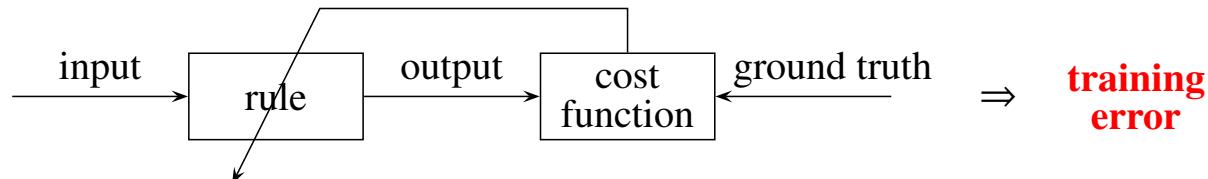
In all cases, the classification or regression rule is learned from examples due to the lack of a signal model. For this purpose, one needs to collect a **dataset** containing input signals and the corresponding desired outputs known as **ground truth** or **labels**. Machine learning learns the underlying input-output mapping from examples.

## Three steps of machine learning (1)

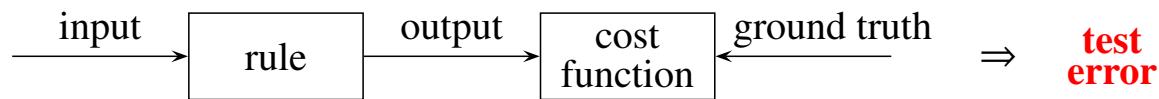
Dataset  $D = D_{\text{train}} \cup D_{\text{test}}$



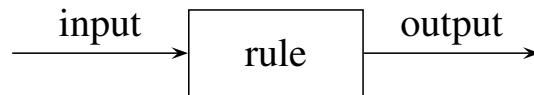
- a) **Training** using **training data** from the training set  $D_{\text{train}}$



- b) **Test** or **evaluation** using **test data** from the test set  $D_{\text{test}}$



- c) Deployment on new data



## Three steps of machine learning (2)

### Dataset

- The dataset is divided into two non-overlapping parts, a training set and a test set.  
The training set is used for learning the rule and the test set is used for its test.<sup>1</sup>

### Training

- The actual output is compared against the ground truth via a **cost function**.
- The rule is adjusted to minimize the cost function, i.e. to make the actual output as close to the ground truth as possible.

### Test

- The learned rule is applied to unseen test data to calculate the test error.
- If test error  $\gg$  training error, **overfitting** occurs. This means, instead of learning the underlying input-output relationship, the computer just memorized the training data. It has a poor generalization to new unseen data. *Overfitting has to be avoided.*

### Deployment

- The learned rule is applied to new data in practice.

---

<sup>1</sup>Late you will learn that the training set is further divided into two parts resulting in an additional validation set.

## Different machine learning tasks

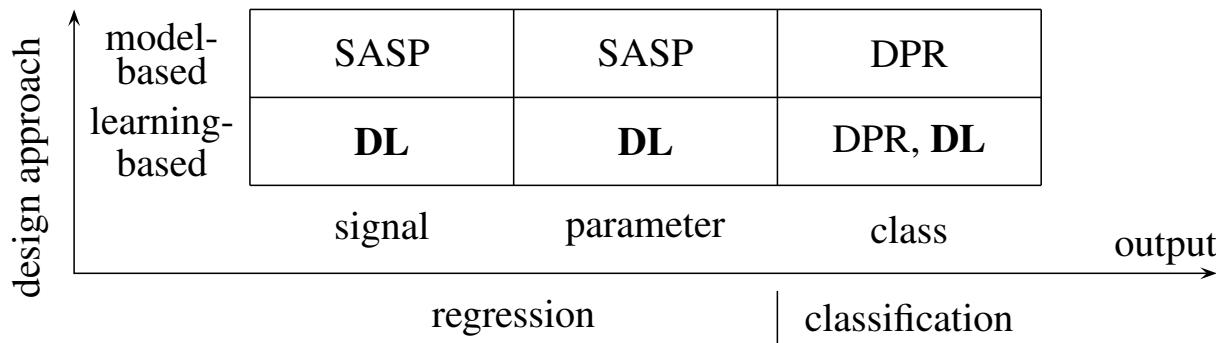
	Classification	Regression
Output	discrete-valued e.g. E1.3 a)–d)	continuous-valued e.g. E1.3 e)

Supervised learning	Unsupervised learning
For each input signal from the dataset, there is a corresponding desired output or ground truth or label. The dataset is <b>labeled</b> or annotated. The computer tries to learn the input-output mapping from examples. This can be classification or regression.	There are only input signals and no corresponding ground truth. The computer tries to find common patterns in the input data. Possible tasks are <b>clustering</b> , <b>dimension reduction</b> or <b>representation learning</b> .

At moment, deep learning (e.g. classification and regression) is mostly supervised.

## I offer four courses for SP and ML in Master

- Winter term: **Advanced mathematics for signal and information processing** (AM, 6CP), basics for all other courses
- Winter term: **Statistical and adaptive signal processing** (SASP, 6CP)
- Summer term: **Detection and pattern recognition** (DPR, 6CP)
- Summer term: **Deep learning** (DL, 6CP)



## AI vs. ML vs. DL vs. DNN

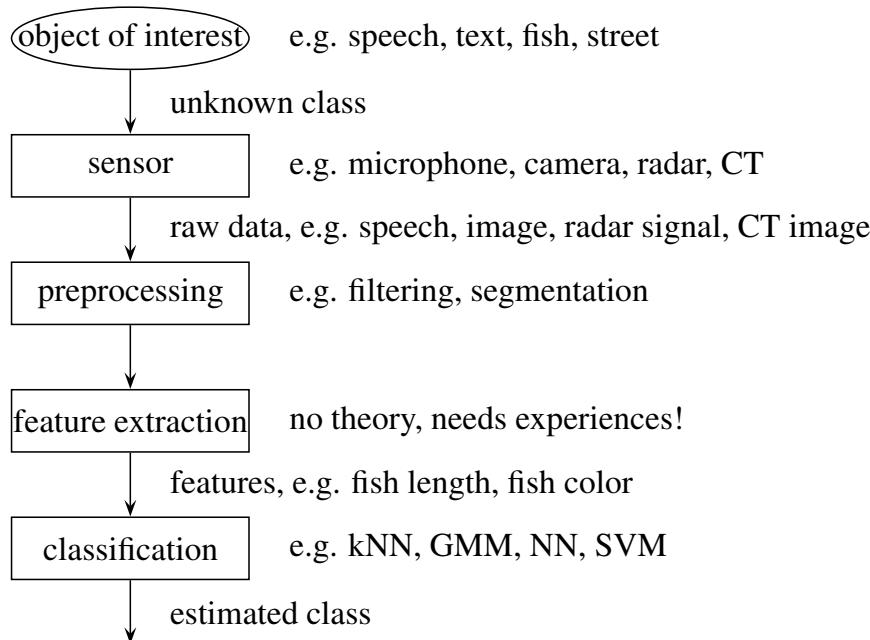
There are many similar buzzwords: **Artificial intelligence (AI)**, **machine learning (ML)**, **deep learning (DL)**, **deep neural network (DNN)**. Are they the same?

AI	also contains expert systems ("if ... then ...")
ML	also contains conventional ML
DL	
DNN	We focus on DNN in this course.

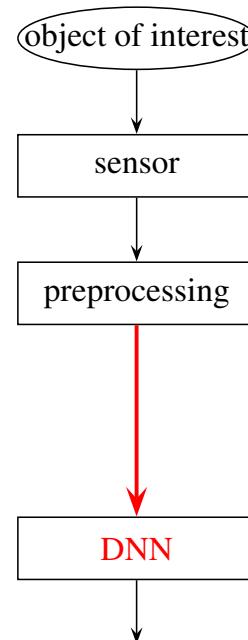
- **Conventional machine learning:** ML before DL, see course DPR.
    - handcrafted feature extraction
    - different supervised classifiers:  $k$  nearest neighbor (kNN), Gaussian mixture model (GMM), neural network (NN), support vector machine (SVM), ...
    - different unsupervised clustering algorithms: k-means, mean-shift, DBSCAN,
- ...

## Conventional machine learning vs. deep learning

### Conventional machine learning



### Deep learning



Deep learning: a) No explicit feature extraction, rather **end-to-end learning**  
 b) Use DNN

## Conventional neural network vs. deep neural network

	Conventional neural network	Deep neural network (DNN)
Depth	shallow (1~2 hidden layers)	deep (many hidden layers)
Architecture	simple	CNN, RNN, AE, GAN, ...
Activation function	sigmoid	ReLU, sigmoid, softmax, ...
Cost function	mean square error (MSE)	MSE, KLD, CE, categorical, ...
Learning	backpropagation	+ advanced techniques
Regularization	none	L1, L2, shortcuts, dropout, ...
Datasets	small	huge
Computing power	small	large

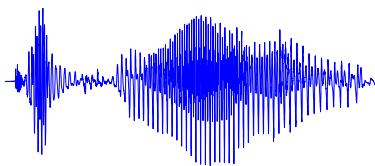
DNN is characterized by

- a deep network and
- many new optimization and regularization techniques.

## Why is deep better (1)?

Many signals have a native *hierarchical* representation.

- Speech recognition: samples → phonemes → words → sentences
- Image recognition: pixels → edges → shapes → objects
- Go: stones → eyes → local groups → global strategies

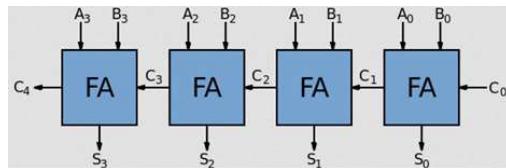
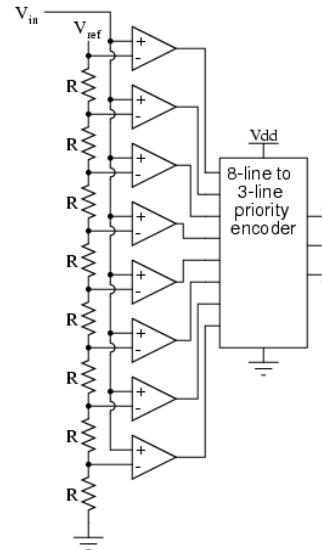


It is easier to capture this multi-level representation by using a deep neural network because each layer only needs to learn an easy mapping. This simplifies the learning. Simple features in the first layers are combined to complex features in the last layers.

## Why is deep better (2)?

A shallow architecture requires a higher complexity than a deep (hierarchical) architecture for the same task.

- $n$ -bit analog-digital-converter (ADC)
  - Deep: Serial ADC,  $n$  levels for  $n$  bits, determine one bit per level  $\rightarrow n$  comparators
  - Shallow: Flash ADC, one level for all  $n$  bits  $\rightarrow 2^n - 1$  comparators
- Addition of two  $n$ -bit numbers
  - Deep: Ripple-carry adder, calculate and propagate the carry bit for bit  $\rightarrow$  only  $n$  full adders (FA)
  - Shallow: Carry-lookahead adder, needs a much more complex logical circuit to predict all carry bits



## A short history of neural networks

- Birth (1940's–1970's): first idea (mimic brain), simple linear neuron (perceptron), could solve only simple linear tasks
- 1970's–1980's: less recognized
- Neural network era (1980's-1990's): 1. renaissance
  - feedforward multilayer network with nonlinear neurons
  - a learning algorithm called backpropagation
  - could solve simple nonlinear tasks (e.g. xor, phoneme recognition)
- 1990's–2010's: no progress and less attention
- Deep neural network era (2010's-present): 2. renaissance
  - deep networks, new architectures, new regularization techniques
  - huge datasets, significantly increased computing power
  - can solve challenging real-life problems, called AI
- *Future?*

## E1.4: Image classification (1)

### ImageNet

- a large dataset for image recognition (classification)
- over 14 million labeled images
- over 20 thousand classes
- image label often not unique



alp



grass snake



harvester



Shetland sheepdog



Eskimo dog

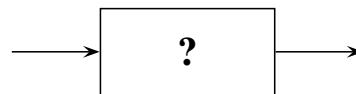


old English sheepdog

## E1.4: Image classification (2)

ImageNet large scale visual recognition challenge (**ILSVRC**)

- 1,000 classes
- 1.2 million training images and 50,000 test images
- **image classification**: assign each image to one class
- competition among different research teams from universities and companies

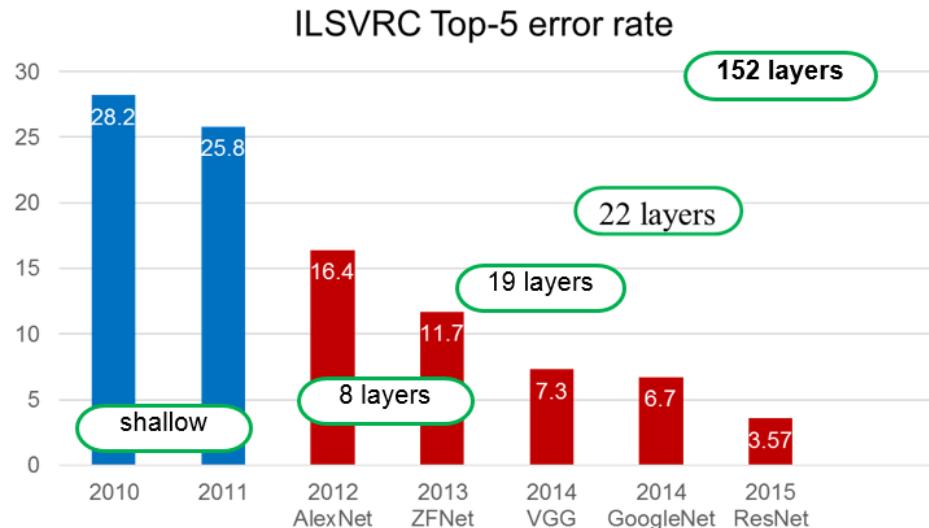


Prob(image from class  $i$ ),  
 $1 \leq i \leq 1000$

Challenges:

- high variability of images (color, shape, perspective, illumination, ...)
- image class often not unique. Hence
  - **top-1 error rate**: Classification fails if the true class is not the top result.
  - **top-5 error rate**: Classification fails if the true class is not among the top 5 results.

## E1.4: Image classification (3)

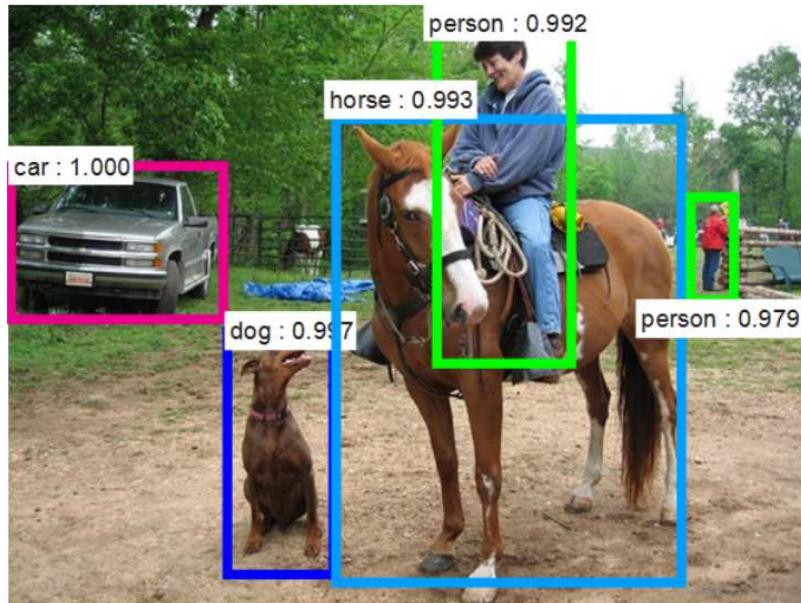


- 2010, 2011: conventional (shallow) neural networks
- since 2012: deep neural networks, see ch. 10 for more details about the networks
- increasing number of layers → increasing accuracy
- ResNet better than average human performance (5%)

*Is DL already smarter than human?* See last lecture "AI vs. human intelligence".

## E1.5: Object detection

- **Object detection:** Detection and localization of (multiple) objects in terms of bounding boxes (as well as classification). This task is more challenging than image classification.<sup>2</sup>



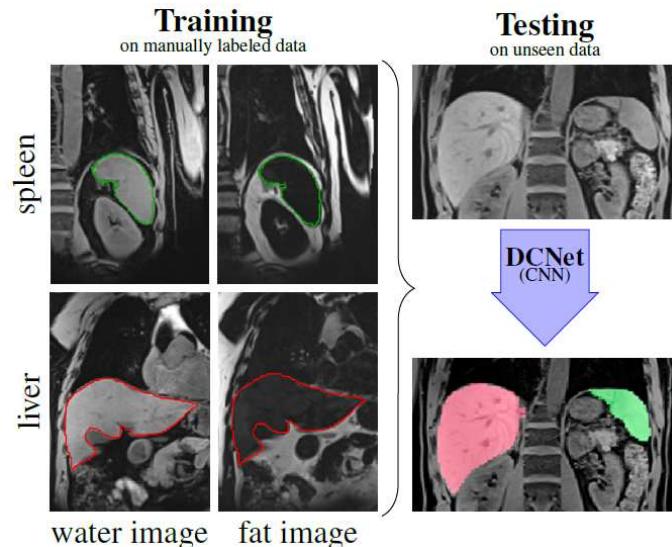
- Common Objects in Context challenge (**COCO**) for this task

<sup>2</sup>S. Ren et al, Faster R-CNN: Towards real-time object detection with region proposal networks, arXiv:1506.01497v3, 2016

## E1.6: Semantic image segmentation (1)

(Semantic) **image segmentation**: Assign each pixel to one class, i.e. pixelwise classification by learning the global visual context of a scene. It is more challenging than object detection.

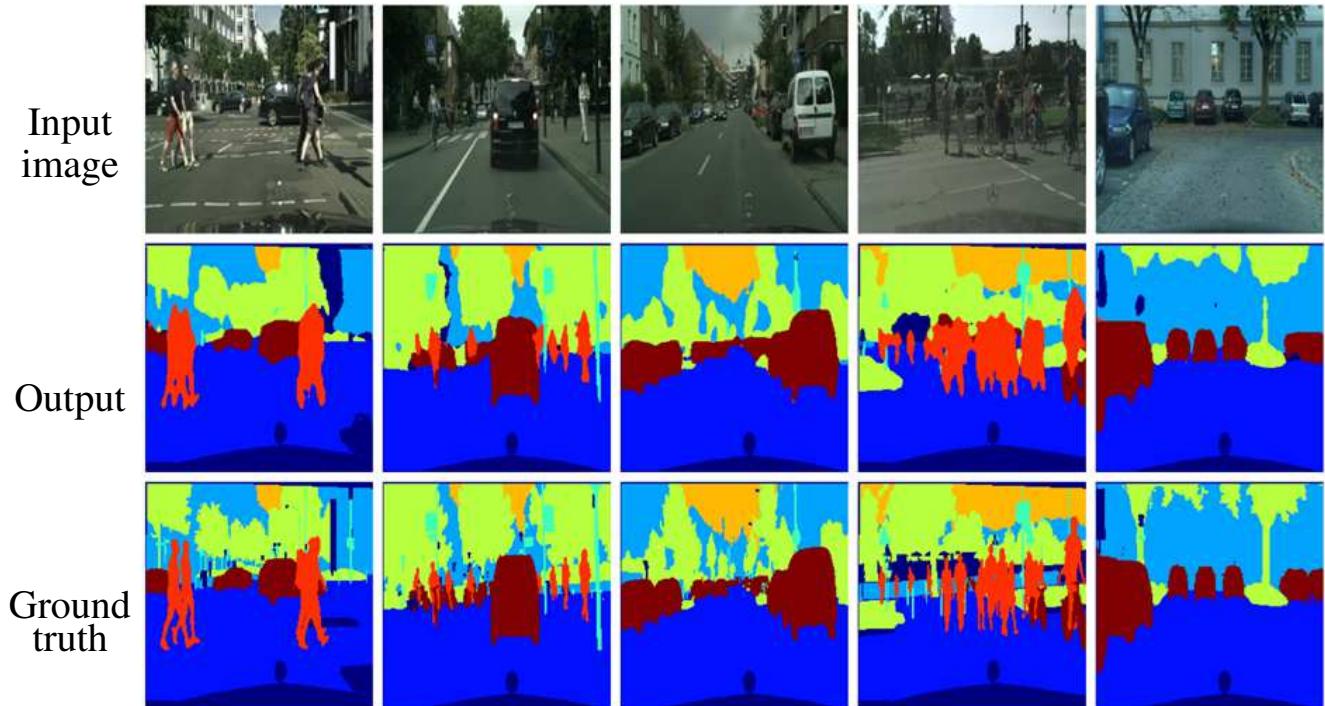
- a) Applied to magnetic resonance images (MRI) to segment organs (spleen, liver) for automated medical diagnosis<sup>3</sup>



<sup>3</sup>ISS publication: T. Küstner, S. Müller et al, Whole-body semantic segmentation of MR images, ICIP, 2018

## E1.6: Semantic image segmentation (2)

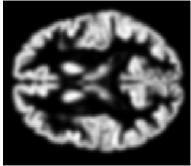
b) Applied to Cityscapes dataset to segment street images for autonomous driving<sup>4</sup>



<sup>4</sup>ISS publication: C. Wang and L. Mauch and Z. Guo and B. Yang. On semantic image segmentation using deep convolutional neural network with shortcuts and easy class extension, IPTA, 2016

### E1.7: Age estimation

Age estimation from the white matter of an MR brain image. It is a regression problem.<sup>5</sup>

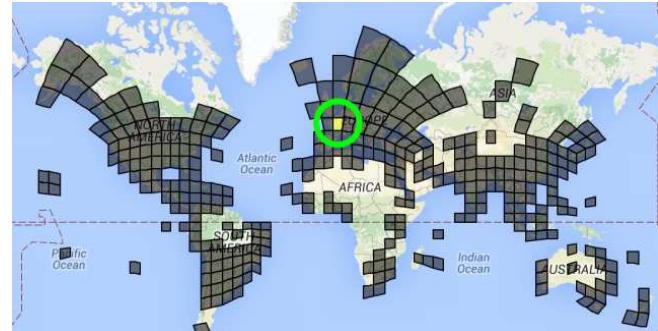
Subject		
True age	23	86
Estimated Age	24.2	88.7

Wiki: "White matter is composed of bundles, which connect various gray matter areas (the locations of nerve cell bodies) of the brain to each other, and carry nerve impulses between neurons."

<sup>5</sup>ISS publication: K. Armanoijus et al, 2020

## E1.8: Geo guessing

- **PlaNet:** Guess geo locations from image, i.e. image → location<sup>6</sup>
- a game based on this idea: [www.geoguessr.com](http://www.geoguessr.com)



*Is it a classification or regression problem?*

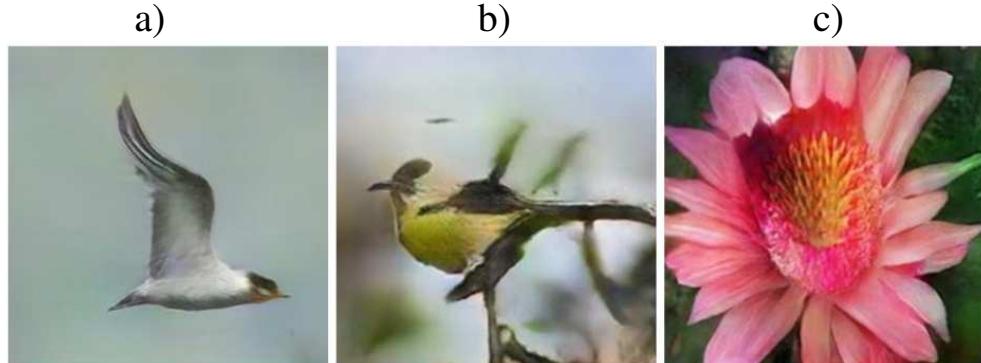
<sup>6</sup>T. Weyand et al, PlaNet - Photo geolocation with convolutional neural networks. In: B. Leibe et al (eds), Computer Vision, Springer, 2016

### E1.9: Text-to-image translation

Text → photo-realistic image using cGAN.<sup>7</sup>

- a) This bird is white with some black on its head and wings, and has a long orange beak.
- b) This bird has a yellow belly and tarsus, grey back, wings, and brown throat, nape with a black face.
- c) This flower has overlapping pink pointed petals surrounding a ring of short yellow filaments.

StackGAN  
Stage-II  
256x256  
images



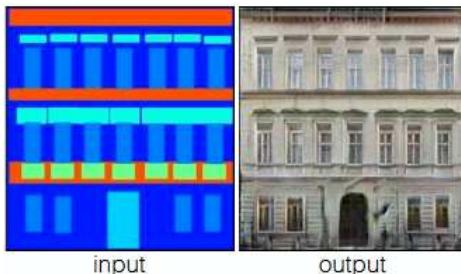
---

<sup>7</sup>H. Zhang et al, StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks, arxiv.org/abs/1612.03242, 2016

### E1.10: Image-to-image translation (1)

Image → image with the same content but in a different style using cGAN.<sup>8</sup>

facade labels → house photo



edges → bag



black/white image → color image



day photo → night photo



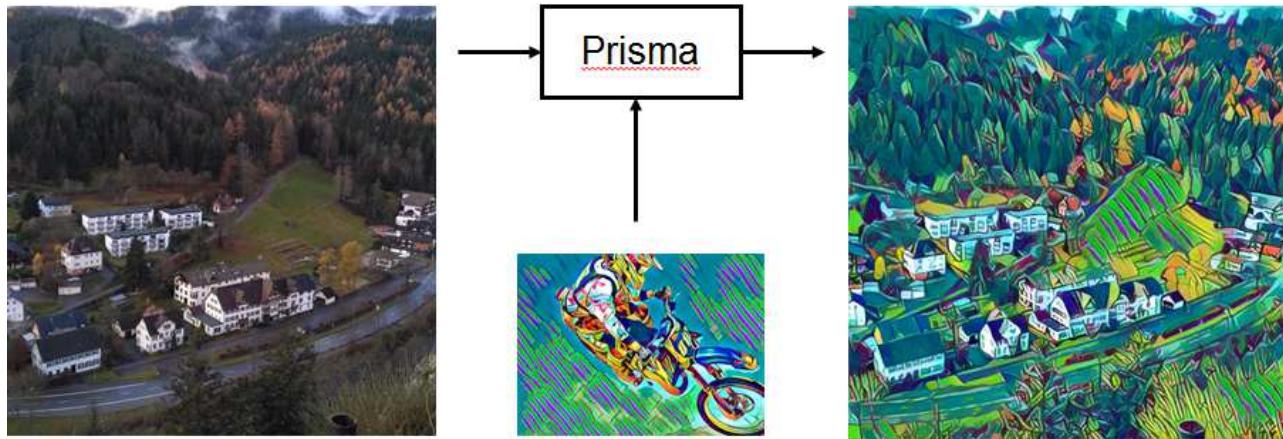
The same techniques can be used to reduce noise, distortion and for super resolution.

<sup>8</sup>P. Isola, Image-to-image translation with conditional adversarial networks, arXiv:1611.07004, 2017

## E1.10: Image-to-image translation (2)

### Style transfer<sup>9</sup>

- analyze the content of a photo
- analyze the painting style of a picture
- combine the photo content with the painting style
- available as Android App "Prisma"



<sup>9</sup>L.A. Gatys et al, A neural algorithm of artistic style, arXiv:1508.0657, 2015

## E1.11: Generative model: AI has finished Schubert's unfinished symphony

- Franz Schubert composed his Symphony No.8 in 1822, but only 2 movements.
- Huawei trained a DNN based on many pieces by Schubert.
- It generated the melody for movement 3 and 4.
- Composer Lucas Cantor orchestrated the melody.
- Feb 14, 2019: Premiere in Cadogan Hall, London
- "If I wasn't told the third and fourth movements were created by artificial intelligence, I wouldn't have known."



Ilias: MP3 of the finished symphony

## E1.12: Speech and language processing

DNN also becomes the state-of-the-art tool for speech and language processing.

- **Speech:** spoken text
  - speech recognition,<sup>10</sup> e.g. Siri
    - \* traditional: HMM (hidden Markov model) + GMM (Gaussian mixture model)
    - \* today: HMM + DNN
  - speech translation, e.g. German → English
  - speech synthesis, e.g. WaveNet (see ch. 10)
    - \* Speech 1: "This is your personal assistant, Google Home."
    - \* Speech 2: "Generative adversarial network or variational auto-encoder."
- **Language:** written text, analysis of
  - newspaper articles
  - internet messages
  - customer feedbacks

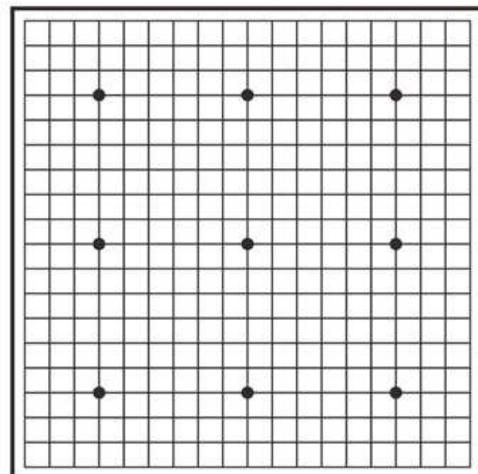
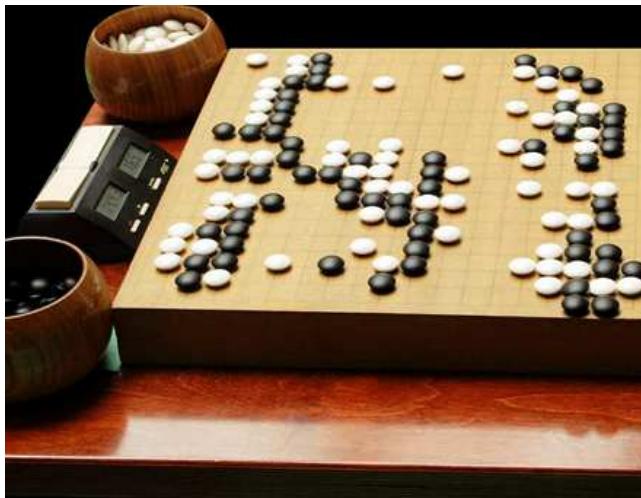
---

<sup>10</sup>G. Hinton et al, Deep neural networks for acoustic modeling in speech recognition, IEEE Signal Processing Magazine, 2012

## E1.13: AlphaGo (1)

### Go

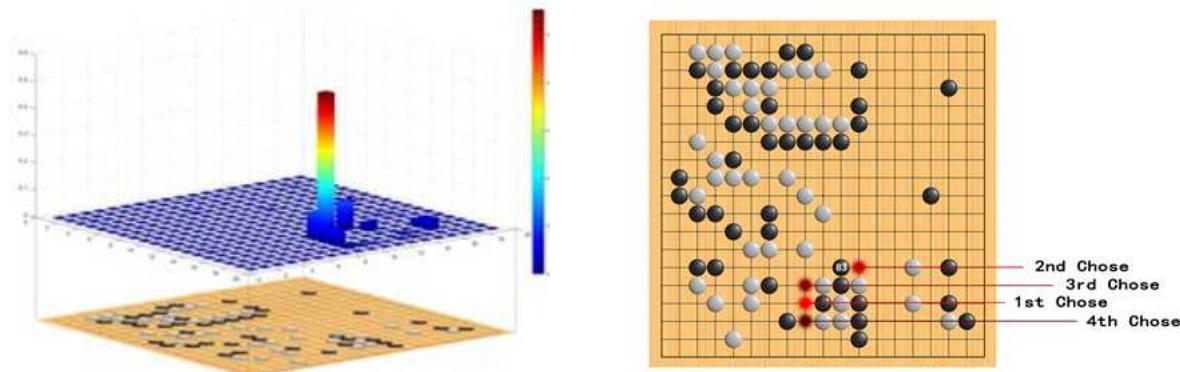
- oldest board game (over 2,500 years) from China
- 19x19 board, black and white stones
- simple rules, complex strategies
- state-space complexity  $3^{361} \approx 10^{172}$  ( $10^{84\sim89}$  atoms in cosmos)



### E1.13: AlphaGo (2)

**AlphaGo** based on deep learning from DeepMind (Google)

- learn from many human Go games
- calculate a probability map for the next best moves based on the current board situation
- 2015: AlphaGo Fan beat a human professional Go player (Fan Hui)
- 2016: AlphaGo Lee beat a 9-dan Go player (Lee Sedol) with 4:1
- 2017: AlphaGo Master beat the world No. 1 (Ke Jie) with 3:0



### E1.13: AlphaGo (3)

2017: **AlphaGo Zero**

- learn to play Go from scratch *without* knowing any human games by AlphaGo-zero against-AlphaGo reinforcement learning
- AlphaGo Zero reached the level of AlphaGo Lee in 3 days (100:0 win)
- AlphaGo Zero reached the level of AlphaGo Master in 21 days
- unreachable by any humans

One month training time beats 2,500 years of human experiences.

Now AlphaGo is retired.

## Different DL tasks

for an input image.

Task	Description	Output	Examples
classification	estimate the class for an image	class	E1.4, E1.8
segmentation	estimate the class for each pixel	[class <sub>ij</sub> ]	E1.5
object detection	localize objects	bounding box	E1.6
regression	estimate real-valued parameter	number	E1.7
translation	change one image to another	image	E1.9, E1.10
generative model	generate an image	image	E1.11

The same also applies to non-image input data.

## Introduction to Python

- easy to learn and powerful
- runs on all major operating systems
- concise syntax
- many packages for machine learning, deep learning and scientific computing
  - Scipy/Numpy (basic linear algebra and optimization)
  - Scikit-Learn (conventional machine learning)
  - Theano (basic linear algebra with automatic differentiation)
  - Lasagne (deep learning)
  - Keras (deep learning)
  - *Tensorflow* (deep learning), see "Introduction into Tensorflow" in this course
- most of the machine learning community uses Python
- you get it for free

MATLAB for signal processing, Python for deep learning.

## Data types (1)

### Built-in scalar types

- variables do not have to be declared, they can just be used
- they have a type that is implicitly assigned when defined
- variables are casted implicitly

Type	Description	Example
int	integer number	a = 100
long	large integer number	a = 1e40
float	fractional number	a = 0.125
bool	logic 0/1 truth values	a = True
None	a value representing nothing	a = None
str	a sequence of characters	a = 'hello world'

### Examples

```
a = 10 # int
b = 1.25 # float
c = a*b # returns c = 12.5 (float)
```

## Data types (2)

### Operators on scalar types

Operator	Description	Example
+	add	$c = a+b$
-	subtract	$c = a-b$
*	multiply	$c = a*b$
/	divide	$c = a/b$
//	integer division	$c = a//b$
%	modulo	$c = a \% b$
**	to the power	$c = a**2$
==	equal	$a == b$
!=	not equal	$a != b$
<	less than	$a < b$
<=	less than or equal to	$a <= b$
>	greater than	$a > b$
>=	greater than or equal to	$a >= b$
&	logic and	$(a==1) \& (b==1)$

## Data types (3)

### Built-in collection types

- collection types summarize scalar data types
- any scalar type can be an element of a collection type

Type	Description	Example
list	a read/write sequence	<code>a = [0, 1, 2, 3]</code>
tuple	a readonly sequence	<code>a = (0, 1, 2, 3)</code>
dict	a dictionary	<code>a = {'a':0, 'b':1, 'c':2}</code>

- each entry of a list/tuple/dict can be anything
- entries can have different types
- entries can also be lists/tuples/dicts

### Examples

```
a = [10, 'a', [5, ('b', 'c')]]
```

## Data types (4)

### Indexing collection types

- use square brackets for indexing: `a[i]`
- the first element has index `i=0`
- the last element has index `i=len(a)-1` or just `i=-1`
- the index can count from the first element (`i>0`) or from the last element (`i<0`)

### Examples

```
a = [[1,2],[3,4],5]
```

```
b = {'a':1, 'b':2}
```

Indexing	Returns
<code>c = a[0][1]</code>	<code>c = 2</code>
<code>c = a[-2]</code>	<code>c = [3,4]</code>
<code>c = b['a']</code>	<code>c = 1</code>

## Data types (5)

### Operators on lists

Input A	Input B	Operator	Returns
a = [1,2]	b = [3,4]	c = a+b	c = [1,2,3,4]
a = [1,2]	b = [3,4]	c = a.append(b)	c = [1,2,[3,4]]
a = [1,2]		c = len(a)	c = 2

### Operators on dictionaries

Input A	Input B	Operator	Returns
a = {'a':1,'b':2}	b = {'c':3,'d':4}	c = a.update(b)	c = {'a':1,'b':2,'c':3,'d':4}
a = {'a':1,'b':2}		del a['b']	a = {'a':1}
a = {'a':1,'b':2}		c = len(a)	c = 2

a.append(): the method "append()" of the object "a"

## Syntax (1)

- Python is case sensitive
- use # for comment

```
# This is a comment line.
```

- code is structured by indentation

### If statements

```
if a == 1:  
    do something  
elif a == 2:  
    do something else  
else:  
    do something else
```

## Syntax (2)

### Loops

```
a = 0
while a < 10:
    print(a)
    a += 1

for a in range(0, 10):
# range returns an iterator over the numbers 0,...,9
    print(a)
```

### Iterating over lists/tuples

```
a = [1, 2, 3, 4]
for ai in a:
    print(ai)
```

# Functions

## Definition

```
def myFunction(a, b=0):  
    c = a + b  
    return c  
  
# call a function (parameters by order)  
result = myFunction(10, 20)  
  
# call a function (parameters by name)  
result = myFunction(10, b=20)  
  
# call a function (with default parameters)  
result = myFunction(10)
```

- you can split your code into meaningful functions
- parameters with a default value (e.g. b=0) may or may not be passed
- parameter assignment by order or name

## Classes and Methods (1)

### Definition

```
class Test:  
    def __init__(self, value, purpose=None):  
        self.value = value  
        self.purpose = purpose  
  
    def get_purpose():  
        return self.purpose  
  
    def store_something(self, x):  
        self.value = x
```

- `__init__(self, ...)` is the constructor of the class
- `self` is the reference to the object with various methods

## Classes and Methods (2)

### Instantiation

```
test = Test(10)
test = Test(10, purpose='nothing')
test = Test(value=10, purpose='nothing')
a = test.get_purpose()
```

### Changing the internal state of objects

```
test = Test(10, purpose='nothing')
test.purpose = 'anything'
```

- `test.value`: value of object test
- `test.method()`: method of object test
- It is allowed but not recommended to change the internal state of an object.

## Packages

- classes and methods can be wrapped up in packages
- you can import packages for use
- you can define a short name for an imported package

```
import MyPackage as mp  
test = mp.Test(10)
```

- you can also import parts of the package

```
from MyPackage import Test as Test  
test = Test(10)
```

## The Numpy package (1)

### What is Numpy?

- basic linear algebra and additional data types (arrays)
- almost the same syntax as MATLAB

### Matrices and vectors

```
import numpy as np
# convert a list of lists to a 2x3 matrix
a = np.array([[1,2,3],[4,5,6]],dtype=np.float32)
# create a 2x3 matrix filled with zeros
a = np.zeros((2,3))
# create a 2x3 matrix filled with ones
a = np.ones((2,3))
# create a 3x3 identity matrix
a = np.eye(3)
```

## The Numpy package (2)

### Indexing and slicing

- $a[i,:]$  for  $i + 1$ -th row of  $a$
- $a[:,j]$  for  $j + 1$ -th column of  $a$

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]],dtype=np.float32)
# assign the first element of a to b
b = a[0,0]
# assign the first column of a to b
b = a[:,0]
# assign all columns of a except for the first one to b
b = a[:,1:]
```

## The Numpy package (3)

### Operators on arrays

```
a = b = np.array([[1,2,3],[4,5,6]],dtype=np.float32)
```

Description	Example
elementwise product	c = a * b
transpose	c = a.T
inner product	c = np.dot(a.T,b)
trace	c = np.trace(a)
diagonal elements	c = np.diag(a)
change dimensions	c = a.reshape(3,2)

### Note

```
a.reshape(3,2) = array([[1,2],[3,4],[5,6]],dtype=float32)
```

## The Numpy package (4)

### Broadcasting

```
import numpy as np
# a 2x3 matrix
a = np.array([[1,2,3],[4,5,6]],dtype=np.float32)
# a 1x3 matrix
b = np.array([[1,2,3]],dtype=np.float32)
# if a and b have the same last dimension (number of columns),
# elementwise operators broadcast over all preceding dimensions,
# i.e. b is added to each row of a
c = a + b
# but broadcast impossible if last dimensions differ
c = a+b.T # results in an error
```

## Hardware for deep learning

### Research

- **CPU**: Central processing unit
- **GPU**: Graphics processing unit (Nvidia)
- enough memory for large datasets

GPU is much faster than CPU in computer graphics, image processing and deep learning due to pipelined matrix-vector calculations and parallel structures.

### Embedded systems

- **$\mu$ C**: Microcontroller
- special accelerator like **TPU**: Tensor processing unit (Google)
- **FPGA**: Field programmable gate arrays
- **ASIC**: Application-specific integrated circuit

### ISS

- Teaching: Google Colab (as in programming homeworks)
- Research:  $\approx 30$  GPUs at ISS

## MNIST dataset<sup>1</sup>

- a small labeled dataset for handwritten digit recognition
- 70,000 gray images of pre-segmented handwritten digits
- 60,000 for training and 10,000 for test
- fixed image size 28x28
- [yann.lecun.com/exdb/mnist/](http://yann.lecun.com/exdb/mnist/)

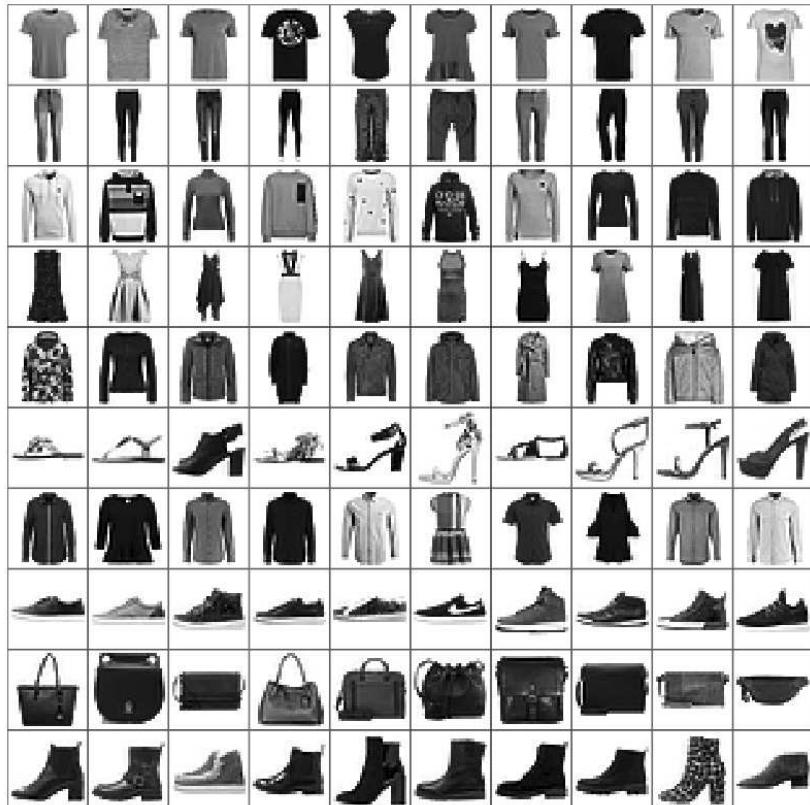


(myselph.de/neuralNet.html)

<sup>1</sup>"NIST" is the National Institute of Standards and Technology, the agency that originally collected this dataset. "M" means a "modified" dataset for easier use in machine learning.

## Fashion MNIST dataset<sup>2</sup>

- a dataset of Zalando's article images as a direct replacement of MNIST
- 10 classes: T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot
- 70,000 pre-segmented gray images, 60,000 for training and 10,000 for test
- fixed image size 28x28



<sup>2</sup><https://research.zalando.com/welcome/mission/research-projects/fashion-mnist>

## SVHN dataset

- a small labeled dataset for street view house number (SVHN) recognition<sup>3</sup>
- color images of street view house numbers
- 73,257 for training and 26,032 for test
- fixed image size 32x32
- large variety of colors and brightness, more challenging than MNIST

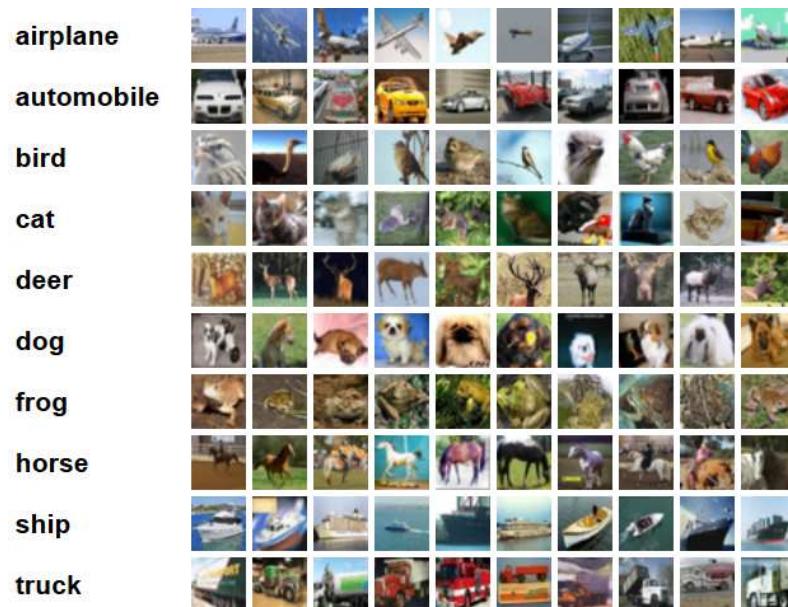


(agi.io/2018/01/31/getting-started-street-view-house-numbers-svhn-dataset/)

<sup>3</sup>Y. Netzer, T. Wang, Coates A., A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

## CIFAR-10 and CIFAR-100 datasets

- small labeled datasets for visual object recognition
- 60,000 color images of size 32x32, 50,000 for training and 10,000 for test
- CIFAR-10: 10 object classes, 6000 images per class
- CIFAR-100: 100 object classes, 600 images per class
- [www.cs.toronto.edu/~kriz/cifar.html](http://www.cs.toronto.edu/~kriz/cifar.html)



## ImageNet dataset

- a large labeled dataset for visual object recognition
- over 14 million color images of objects
- over 20,000 classes
- varying image sizes (average image size 469x387)
- [www.image-net.org](http://www.image-net.org)



(karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/)

## Tiny ImageNet dataset

- a miniature of ImageNet for low computing power
- 120,000 color images of size 64x64
- 200 classes
- 500/50/50 for training/validation/test images per class

## Cityscapes dataset

- a large labeled dataset of street images for semantic image segmentation
- 50 cities and 30 object classes from 8 object groups (flat, human, vehicle, construction, object, nature, sky, void)
- 5,000/20,000 labeled images with fine/coarse annotations
- large image size
- [www.cityscapes-dataset.com](http://www.cityscapes-dataset.com)

Stuttgart (2040x1016 pixels)



## Which hardware for which datasets?

- MNIST, Fashion MNIST, SVHN, CIFAR-10, CIFAR-100, tiny ImageNet: CPU is sufficient.
- ImageNet, Cityscapes: You need a GPU.
- The Python code is the same for CPU or CPU+GPU except for an one-line change in a configuration file.

## Vector and matrix

- $x \in \mathbb{R}$ : **scalar**
- $\underline{x} \in \mathbb{R}^M = \mathbb{R}^{M \times 1}$ : column **vector** of length  $M$ . The notation  $\underline{x} = [x_i]$  means that  $\underline{x}$  consists of the elements  $x_i, 1 \leq i \leq M$ .
- $\mathbf{X} \in \mathbb{R}^{M \times N}$ : **matrix** of dimensions  $M \times N$ . The notation  $\mathbf{X} = [x_{ij}]$  means that  $\mathbf{X}$  consists of the elements  $x_{ij}, 1 \leq i \leq M, 1 \leq j \leq N$ .
- $\underline{x}^T$  or  $\mathbf{X}^T$ : **transpose** of  $\underline{x}$  or  $\mathbf{X}$
- $\mathbf{X} \odot \mathbf{Y} = [x_{ij}y_{ij}]$  is the **elementwise multiplication (Hadamard product)** of two matrices  $\mathbf{X}$  and  $\mathbf{Y}$  of the same dimensions.
- The **vectorization** operator  $\text{vec}$  stacks all columns of a matrix  $\mathbf{X}$  into a single column vector:

$$\text{vec}(\mathbf{X}) = \begin{bmatrix} \text{first column of } \mathbf{X} \\ \vdots \\ \text{last column of } \mathbf{X} \end{bmatrix}.$$

## Tensor

Vector and matrix are one- or two-dimensional arrays of numbers. An extension to multi-dimensional arrays is called **tensor**. The dimensionality is called the **order** of the tensor:

 $x$ 

scalar or 0th-order tensor

 $\underline{x} = [x_i] \in \mathbb{R}^M$ 

vector or 1st-order tensor with the dimension  $M$

 $\mathbf{X} = [x_{ij}] \in \mathbb{R}^{M \times N}$ 

matrix or 2nd-order tensor with the dimensions  $M, N$

 $\mathbf{X} = [x_{ijk}] \in \mathbb{R}^{L \times M \times N}$ 

3rd-order tensor with the dimensions  $L, M, N$

 $\dots$  $\dots$ 

Tensors are frequently used in convolutional neural networks, see ch. 7.

## Trace

Given a square matrix  $\mathbf{X} = [x_{ij}] \in \mathbb{R}^{M \times M}$ . Its **trace** is the sum of all diagonal elements

$$\text{tr}(\mathbf{X}) = \sum_{i=1}^M x_{ii} \in \mathbb{R}.$$

Properties of trace:

- P1)  $\text{tr}(\mathbf{I}) = M$
- P2)  $\text{tr}()$  is a linear operator:  $\text{tr}(a\mathbf{X} + b\mathbf{Y}) = a\text{tr}(\mathbf{X}) + b\text{tr}(\mathbf{Y})$
- P3)  $\text{tr}(\mathbf{X}^T) = \text{tr}(\mathbf{X})$
- P4)  $\text{tr}(\mathbf{XY}) = \text{tr}(\mathbf{YX})$ ,     $\text{tr}(\mathbf{XYZ}) = \text{tr}(\mathbf{YZX}) = \text{tr}(\mathbf{ZXY})$

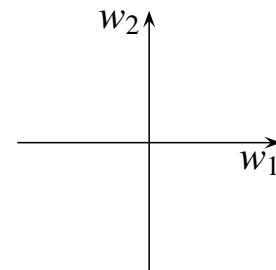
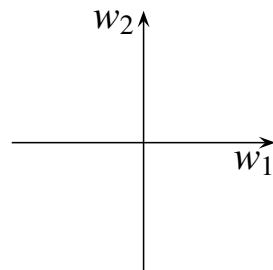
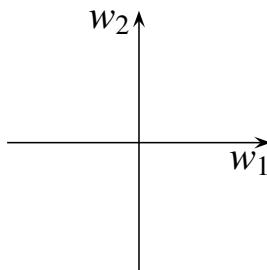
## Vector norms

Given a vector  $\underline{x} = [x_i] \in \mathbb{R}^M$ . There are different definitions for the vector norm:

- **2-norm** or  $l_2$ -norm or **Euclidean norm**:  $\|\underline{x}\|_2 = \sqrt{\sum_{i=1}^M x_i^2} = \sqrt{\underline{x}^T \underline{x}}$
- **1-norm** or  $l_1$ -norm:  $\|\underline{x}\|_1 = \sum_{i=1}^M |x_i|$
- **0-norm** or  $l_0$ -norm:  $\|\underline{x}\|_0$  = number of non-zero elements in  $\underline{x}$

Comments:

- $\|\underline{x}\|_2^2$  represents the energy of  $\underline{x}$ .
- $\|\underline{x}\|_0$  measures the sparsity of  $\underline{x}$ .
- Different vector norms have different unit-norm contour lines  $\{\underline{x} \mid \|\underline{x}\|_p = 1\}$ .



## Derivative

Let  $\underline{y} = [y_i] \in \mathbb{R}^M$  be a vector function of  $\underline{x} = [x_i] \in \mathbb{R}^N$ . The derivative of  $\underline{y}$  w.r.t.  $\underline{x}$  is defined as

$$\mathbf{J} = \frac{\partial \underline{y}}{\partial \underline{x}} = \left[ \frac{\partial y_i}{\partial x_j} \right]_{ij} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_M}{\partial x_1} & \cdots & \frac{\partial y_M}{\partial x_N} \end{bmatrix} \in \mathbb{R}^{M \times N}.$$

It is called the **Jacobi matrix**. Clearly, the derivative of a vector  $\underline{y} \in \mathbb{R}^M$  w.r.t. a scalar number  $x$  is a column vector

$$\frac{\partial \underline{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \vdots \\ \frac{\partial y_M}{\partial x} \end{bmatrix}.$$

But the derivative of a scalar function  $y(x)$  w.r.t. a vector  $\underline{x} \in \mathbb{R}^N$  is a row vector  $\left[ \frac{\partial y}{\partial x_1}, \dots, \frac{\partial y}{\partial x_N} \right]$ . It is the transpose of the **gradient vector** of  $y(\underline{x})$

$$\nabla y = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \vdots \\ \frac{\partial y}{\partial x_N} \end{bmatrix} = \left( \frac{\partial y}{\partial \underline{x}} \right)^T.$$

## Chain rule and product rule of derivative

**Chain rule:** Let  $L(\underline{x}(\underline{y}(\theta))) \in \mathbb{R}$  be a function of  $\underline{x}$  that is a function of  $\underline{y}$  that is a function of  $\theta \in \mathbb{R}$ . Then

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \underline{x}} \cdot \frac{\partial \underline{x}}{\partial \underline{y}} \cdot \frac{\partial \underline{y}}{\partial \theta}$$

It plays an important role in backpropagation for the training of a neural network, see ch. 4.

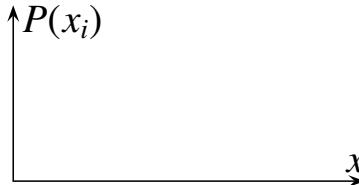
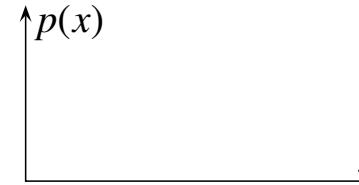
**Product rule:**

$$\frac{\partial x(\theta)y(\theta)}{\partial \theta} = \frac{\partial x(\theta)}{\partial \theta}y(\theta) + x(\theta)\frac{\partial y(\theta)}{\partial \theta}$$

This will be used in backpropagation through time, see ch. 8.

## One random vector

Let  $\underline{X} = [X_i]$  be a real-valued random vector containing  $d$  **random variables (RV)**. The **distribution** (PMF or PDF) describes fully the statistical properties of  $\underline{X}$ .

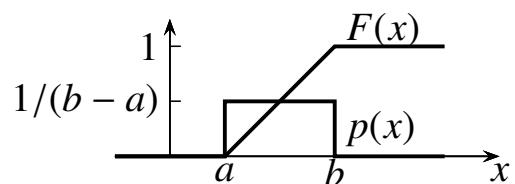
	discrete-valued	continuous-valued
value	$\underline{X} \in \{\underline{x}_1, \underline{x}_2, \dots\}$	$\underline{X} \in \mathbb{R}^d$
distribution	<b>probability mass function (PMF)</b> $\underline{X} \sim P(\underline{X} = \underline{x}_i) = P(\underline{x}_i) = p_i \geq 0$	<b>probability density function (PDF)</b> $\underline{X} \sim p(\underline{x}) = \lim_{\Delta \underline{x} \rightarrow 0} \frac{P(\underline{x} < \underline{X} \leq \underline{x} + \Delta \underline{x})}{ \Delta \underline{x} } \geq 0$
normalization	$\sum_i p_i = 1$	$\int p(\underline{x}) d\underline{x} = 1$
notation	$P(\cdot)$	$p(\cdot)$
		

### E3.1: PDF and CDF

a) **Uniform distribution** in  $[a, b]$ :

$$p(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b \\ 0 & \text{for others} \end{cases},$$

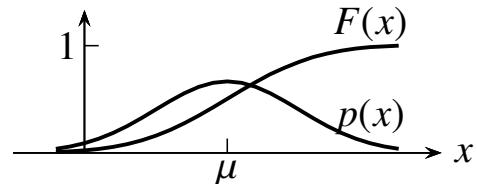
$$F(x) = \int_{-\infty}^x p(z)dz = \begin{cases} 0 & \text{for } x < a \\ \frac{x-a}{b-a} & \text{for } a \leq x \leq b \\ 1 & \text{for } b < x \end{cases}.$$



b) **Normal (Gaussian) distribution**  $N(\mu, \sigma^2)$ :

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

$$F(x) = \int_{-\infty}^x p(z)dz.$$



## Moments of one random vector

- **mean** of  $\underline{X}$

$$\mu = E(\underline{X}) = \int \underline{x} p(\underline{x}) d\underline{x} \stackrel{\text{d.v.}}{=} \sum_i \underline{x}_i P(\underline{x}_i) \in \mathbb{R}^d$$

$E(\cdot)$  is the **expectation**, a statistical average over all random realizations.

- **correlation matrix** of  $\underline{X}$

$$\mathbf{R} = E(\underline{X} \underline{X}^T) = \int \underline{x} \underline{x}^T p(\underline{x}) d\underline{x} \stackrel{\text{d.v.}}{=} \sum_i \underline{x}_i \underline{x}_i^T P(\underline{x}_i) \in \mathbb{R}^{d \times d}$$

- **covariance matrix** of  $\underline{X}$

$$\begin{aligned} \mathbf{C} &= E((\underline{X} - \underline{\mu})(\underline{X} - \underline{\mu})^T) = \int (\underline{x} - \underline{\mu})(\underline{x} - \underline{\mu})^T p(\underline{x}) d\underline{x} \\ &\stackrel{\text{d.v.}}{=} \sum_i (\underline{x}_i - \underline{\mu})(\underline{x}_i - \underline{\mu})^T P(\underline{x}_i) = \mathbf{R} - \underline{\mu} \underline{\mu}^T \in \mathbb{R}^{d \times d} \end{aligned}$$

In general, **moments** are only a partial description of the statistical properties of  $\underline{X}$ .

## Multivariate normal (Gaussian) distribution

### PDF

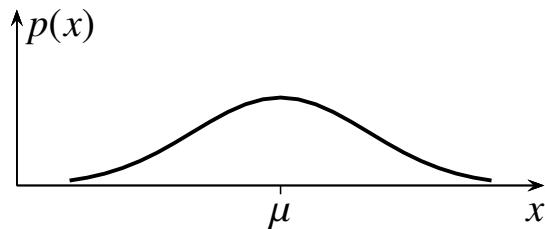
$$\underline{X} \in \mathbb{R}^d \sim N(\underline{\mu}, \mathbf{C})$$

$$p(\underline{x}) = \frac{1}{(2\pi)^{d/2} |\mathbf{C}|^{1/2}} e^{-\frac{1}{2}(\underline{x}-\underline{\mu})^T \mathbf{C}^{-1} (\underline{x}-\underline{\mu})}$$

$$\ln(p(\underline{x})) = -\frac{d}{2} \ln(2\pi) - \frac{1}{2} \ln(|\mathbf{C}|) - \frac{1}{2}(\underline{x} - \underline{\mu})^T \mathbf{C}^{-1} (\underline{x} - \underline{\mu})$$

$N(\underline{0}, \mathbf{I})$  is called the **standard normal distribution**.

### 1D-Visualization



### Moments

$$\mathbb{E}(\underline{X}) = \underline{\mu}, \quad \text{Cov}(\underline{X}) = \mathbf{C}$$

## Laplace distribution

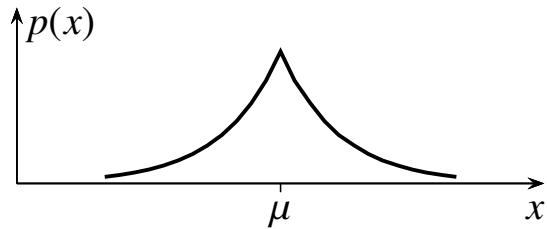
### PDF

$$X \in \mathbb{R} \sim \text{Laplace}(\mu, b), \quad b > 0$$

$$p(x) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}}$$

$$\ln(p(x)) = -\ln(2b) - \frac{1}{b}|x - \mu|$$

### Visualization



### Moments

$$\text{E}(X) = \mu, \quad \text{Var}(X) = 2b^2$$

## Bernoulli distribution

for a binary random variable, e.g. flip a coin.



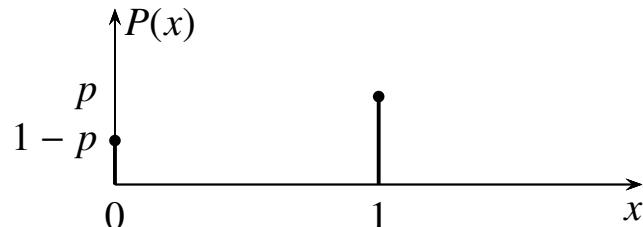
### PMF

$$X \in \{0, 1\} \sim \text{Bernoulli}(p), \quad p = P(X = 1) > 0$$

$$P(x) = \begin{cases} 1 - p & \text{for } x = 0 \\ p & \text{for } x = 1 \end{cases} = p^x(1 - p)^{1-x}$$

$$\ln(P(x)) = x \ln(p) + (1 - x) \ln(1 - p)$$

### Visualization



### Moments

$$\text{E}(X) = p, \quad \text{Var}(X) = p(1 - p)$$

## Categorical distribution or multinoulli distribution

for a discrete multi-state random variable, e.g. roll a die.

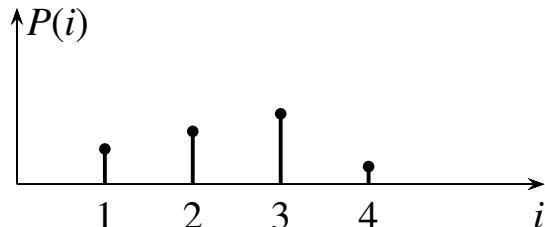


### PMF

$$X \in \{1, 2, \dots, c\} \sim \text{Cat}(\underline{p}), \quad P(X = i) = P(i) = p_i, 1 \leq i \leq c$$

$$\underline{p} = [p_1, p_2, \dots, p_c]^T, \quad \sum_{i=1}^c p_i = \underline{1}^T \underline{p} = 1$$

### Visualization



### Moments

$$\text{E}(X) = \sum_{i=1}^c ip_i, \quad \text{Var}(X) = \text{E}(X^2) - (\text{E}X)^2 = \sum_{i=1}^c i^2 p_i - (\text{E}X)^2$$

## One-hot coding

**One-hot coding** is a frequently used coding for a group of bits when testing digital circuits. It is defined such that all bits are "0" except for a single "1", e.g. 0100. A similar coding is called **one-cold coding**: all bits are "1" except for a single "0", e.g. 1011.

The one-hot coding is used to denote the class label in DNN. For a  $c$ -class classification task, there are two ways to represent the class label:

- Scalar integer coding: The class label is a random variable  $y \in \{1, 2, \dots, c\}$ .
- Vectorial one-hot coding: The class label is a random vector  $\underline{y} \in \{\underline{e}_1, \underline{e}_2, \dots, \underline{e}_c\}$  where  $\underline{e}_i$  is the  $i$ -th unit vector, i.e. the  $i$ -th column of the  $c \times c$  identity matrix  $\mathbf{I} = [\underline{e}_1, \underline{e}_2, \dots, \underline{e}_c]$ . This means, all elements of  $\underline{y}$  are "0" except for a single "1". The position of "1" is the class index.

### E3.2: One-hot coding

Consider the digit recognition task (MNIST). There are ten classes, the digits  $0, 1, \dots, 9$ . The one-hot coding of these 10 classes consists of the following 10 unit vectors:

$$\underline{y} = \underline{e}_1 = [1, 0, 0, \dots, 0, 0]^T \quad \text{for digit "0"}$$

$$\underline{y} = \underline{e}_2 = [0, 1, 0, \dots, 0, 0]^T \quad \text{for digit "1"}$$

$$\vdots \quad \vdots$$

$$\underline{y} = \underline{e}_{10} = [0, 0, 0, \dots, 0, 1]^T \quad \text{for digit "9"}$$

The one-hot encoding seems to be more complicated than the normal integer encoding, but it enables an elegant expression for the PMF, see below.

## Different distributions of $\underline{X}$ and $\underline{Y}$

Continuous-valued RVs, PDF	Discrete-valued RVs, PMF
<b>Joint distribution</b>	
$p(\underline{x}, \underline{y}) = p\left(\begin{bmatrix} \underline{x} \\ \underline{y} \end{bmatrix}\right)$	$P(\underline{x}_i, \underline{y}_j) = P\left(\begin{bmatrix} \underline{x}_i \\ \underline{y}_j \end{bmatrix}\right)$
<b>Marginal distribution</b>	
$p(\underline{x}) = \int p(\underline{x}, \underline{y}) d\underline{y}$	$P(\underline{x}_i) = \sum_j P(\underline{x}_i, \underline{y}_j)$
$p(\underline{y}) = \int p(\underline{x}, \underline{y}) d\underline{x}$	$P(\underline{y}_j) = \sum_i P(\underline{x}_i, \underline{y}_j)$
<b>Conditional distribution</b>	
$p(\underline{x} \underline{y}) = \frac{p(\underline{x}, \underline{y})}{p(\underline{y})}$	$P(\underline{x}_i \underline{y}_j) = \frac{P(\underline{x}_i, \underline{y}_j)}{P(\underline{y}_j)}$
$p(\underline{y} \underline{x}) = \frac{p(\underline{x}, \underline{y})}{p(\underline{x})}$	$P(\underline{y}_j \underline{x}_i) = \frac{P(\underline{x}_i, \underline{y}_j)}{P(\underline{x}_i)}$

## Chain rule of probability

The **chain rule of probability** formulates a joint distribution in terms of a number of conditional distributions. It is the generalization of the product rule:

$$p(\underline{x}_1, \dots, \underline{x}_N) = \left( \prod_{i=1}^{N-1} p(\underline{x}_i | \underline{x}_{i+1}, \dots, \underline{x}_N) \right) p(\underline{x}_N).$$

Proof:

$$p(\underline{x}_1, \dots, \underline{x}_N) = p(\underline{x}_1 | \underline{x}_2, \dots, \underline{x}_N) p(\underline{x}_2, \dots, \underline{x}_N),$$

$$p(\underline{x}_2, \dots, \underline{x}_N) = p(\underline{x}_2 | \underline{x}_3, \dots, \underline{x}_N) p(\underline{x}_3, \dots, \underline{x}_N),$$

⋮

$$p(\underline{x}_{N-1}, \underline{x}_N) = p(\underline{x}_{N-1} | \underline{x}_N) p(\underline{x}_N).$$

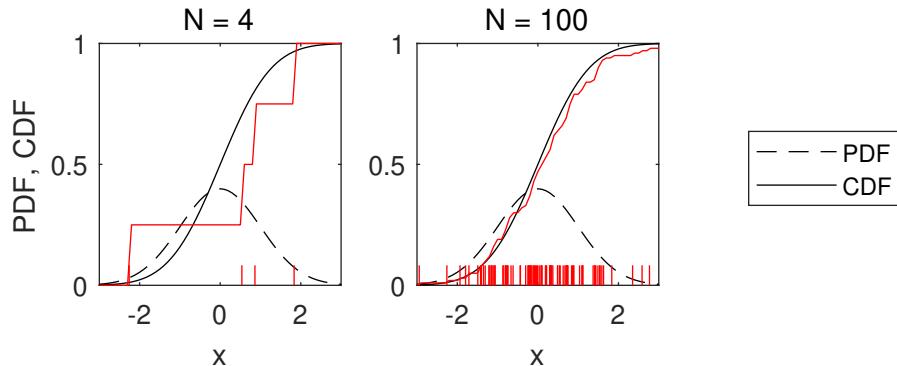
Hence

$$p(\underline{x}_1, \dots, \underline{x}_N) = p(\underline{x}_1 | \underline{x}_2, \dots, \underline{x}_N) p(\underline{x}_2 | \underline{x}_3, \dots, \underline{x}_N) \dots p(\underline{x}_{N-1} | \underline{x}_N) p(\underline{x}_N).$$

### E3.3: Estimate of PDF and CDF

$x(n), 1 \leq n \leq N$  are i.i.d. samples drawn from  $p(x) \sim N(0, 1)$ . They are used to calculate the empirical PDF  $\hat{p}(x) = \frac{1}{N} \sum_{n=1}^N \delta(x - x(n))$  and the empirical CDF  $\hat{F}(x) = \int_{-\infty}^x \hat{p}(z) dz = \frac{1}{N} \sum_{n=1}^N u(x - x(n))$ .  $u(x)$  is the unit step function.

$$u(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$



- The PDFs of the true and empirical distribution look quite different, but the CDFs agree well for large  $N$ .
- We do not need a smooth PDF estimate (using e.g. the Gaussian kernel). Later in this chapter, the empirical distribution is used to approximate the unknown true distribution  $p(x)$ .

## Proof of $D_{\text{KL}}(p\|q) \geq 0 \ \forall p, q^{\star}$ <sup>1</sup>

Let  $f(\underline{x})$  be a **convex function**. According to the **Jensen's inequality**,

$$f\left(\sum_{i=1}^N \lambda_i \underline{x}_i\right) \leq \sum_{i=1}^N \lambda_i f(\underline{x}_i)$$

$\forall \underline{x}_i$  and  $\lambda_i \geq 0$  with  $\sum_{i=1}^N \lambda_i = 1$ . This means, the curve of the function  $f\left(\sum_{i=1}^N \lambda_i \underline{x}_i\right)$  is below the secant line  $\sum_{i=1}^N \lambda_i f(\underline{x}_i)$ , see course AM.

$-\ln(z)$  is a convex function. Hence<sup>2</sup>

$$\begin{aligned} D_{\text{KL}}(P\|Q) &= \sum_i P(\underline{x}_i) \ln\left(\frac{P(\underline{x}_i)}{Q(\underline{x}_i)}\right) = \sum_i \underbrace{P(\underline{x}_i)}_{\lambda_i} \underbrace{-\ln\left(\frac{Q(\underline{x}_i)}{\underbrace{P(\underline{x}_i)}_{\underline{x}_i}}\right)}_{f} \\ &\geq -\ln\left(\sum_i P(\underline{x}_i) \frac{Q(\underline{x}_i)}{P(\underline{x}_i)}\right) = -\ln(1) = 0. \end{aligned}$$

+

---

<sup>1</sup>If a topic is marked with a star \*, it is an advanced topic and not relevant for the exam.

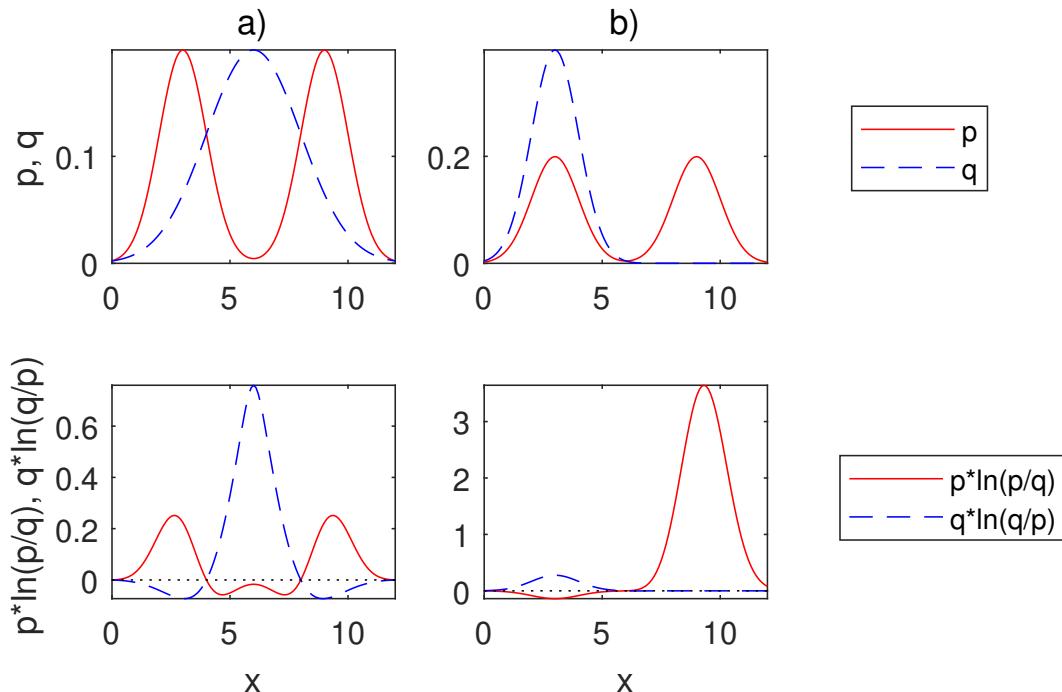
<sup>2</sup>We show here the proof for discrete random variables with the PMFs  $P(\underline{x})$  and  $Q(\underline{x})$ . The proof for continuous random variables with the PDFs  $p(\underline{x})$  and  $q(\underline{x})$  is similar.

## Forward vs. backward KL divergence

	Forward KL divergence $D_{\text{KL}}(p\ q)$	Backward KL divergence $D_{\text{KL}}(q\ p)$
$p > q$	$p(\underline{x}) \ln \left( \frac{p(\underline{x})}{q(\underline{x})} \right) > 0$	$q(\underline{x}) \ln \left( \frac{q(\underline{x})}{p(\underline{x})} \right) < 0$
$q \rightarrow 0$	$\rightarrow \infty$	$\rightarrow 0$
$p \rightarrow 0$	$\rightarrow 0$	$\rightarrow \infty$
	Minimize $D_{\text{KL}}(p\ q)$	Minimize $D_{\text{KL}}(q\ p)$
	$p = 0$ : doesn't care about $q$ $p > 0$ : make $q$ close to $p$	$q = 0$ : doesn't care about $p$ $q > 0$ : make $q$ close to $p$
	"zero avoiding" strategy for $q$ : $q > 0$ if $p > 0$ i.e. makes $q(\underline{x})$ broader than $p(\underline{x})$	"zero forcing" strategy for $q$ : $q = 0$ if $p = 0$ i.e. makes $q(\underline{x})$ narrower than $p(\underline{x})$

Note:  $\lim_{x \rightarrow 0} x \ln(x) = 0$ .

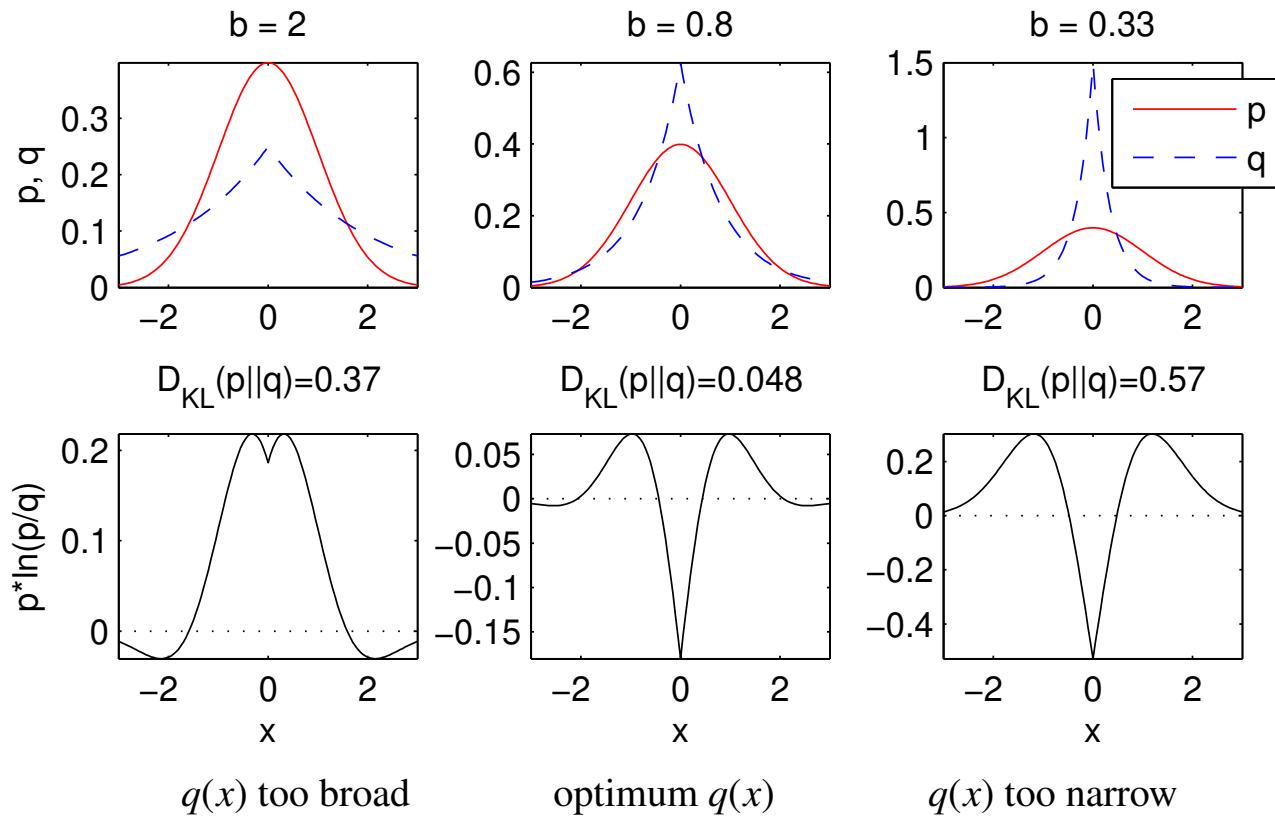
### E3.4: Forward vs. backward KL divergence



- When minimizing  $D_{\text{KL}}(p||q)$ , a) is better than b) because  $q(x)$  is broad.
- When minimizing  $D_{\text{KL}}(q||p)$ , b) is better than a) because  $q(x)$  is narrow.

### E3.5: KL divergence between normal and Laplace distribution

$$p(x) \sim N(0, 1), \quad q(x) \sim \text{Laplace}(0, b)$$



## Proof of additivity

If  $\underline{x} = \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \end{bmatrix}$  with independent  $\underline{x}_1$  and  $\underline{x}_2$ , i.e.

$$p(\underline{x}) = p_1(\underline{x}_1)p_2(\underline{x}_2) \quad \text{and} \quad q(\underline{x}) = q_1(\underline{x}_1)q_2(\underline{x}_2),$$

then

$$\begin{aligned} D_{\text{KL}}(p\|q) &= \int p(\underline{x}) \ln \left( \frac{p(\underline{x})}{q(\underline{x})} \right) d\underline{x} \\ &= \iint p_1(\underline{x}_1)p_2(\underline{x}_2) \ln \left( \frac{p_1(\underline{x}_1)p_2(\underline{x}_2)}{q_1(\underline{x}_1)q_2(\underline{x}_2)} \right) d\underline{x}_1 d\underline{x}_2 \\ &= \iint p_1(\underline{x}_1)p_2(\underline{x}_2) \ln \left( \frac{p_1(\underline{x}_1)}{q_1(\underline{x}_1)} \right) d\underline{x}_1 d\underline{x}_2 + \iint p_1(\underline{x}_1)p_2(\underline{x}_2) \ln \left( \frac{p_2(\underline{x}_2)}{q_2(\underline{x}_2)} \right) d\underline{x}_1 d\underline{x}_2 \\ &= \int p_1(\underline{x}_1) \ln \left( \frac{p_1(\underline{x}_1)}{q_1(\underline{x}_1)} \right) d\underline{x}_1 + \int p_2(\underline{x}_2) \ln \left( \frac{p_2(\underline{x}_2)}{q_2(\underline{x}_2)} \right) d\underline{x}_2 \\ &= D_{\text{KL}}(p_1\|q_1) + D_{\text{KL}}(p_2\|q_2). \end{aligned}$$

This implies later in ch. 4 that for independent samples  $\underline{x}(n)$ , we can simply add the KLD of  $\underline{x}(n)$ .

## Entropy and cross entropy

In information theory, the **entropy**  $H(X)$  of a random variable  $X$  is a measure for its average information in the sense of uncertainty. The larger the uncertainty, the larger the information content.

If  $X \sim p_i$  is a discrete random variable with the possible values  $x_i, 1 \leq i \leq M$  and the PMF  $P(X = x_i) = p_i$ , its entropy is<sup>3</sup>

$$H(X) = H(P) = -E(\ln(P(X))) = -\sum_{i=1}^M p_i \ln(p_i) \text{ [nats]} \geq 0.$$

If  $X \sim p(x)$  is a continuous random variable with the PDF  $p(x)$ , its (differential) entropy is

$$H(X) = H(p) = -E(\ln(p(X))) = -\int_{-\infty}^{\infty} p(x) \ln(p(x)) dx \text{ [nats]} \geq 0.$$

For two different distributions  $X \sim p(x)$  and  $Y \sim q(y)$ , the **cross entropy (CE)** between them is defined as

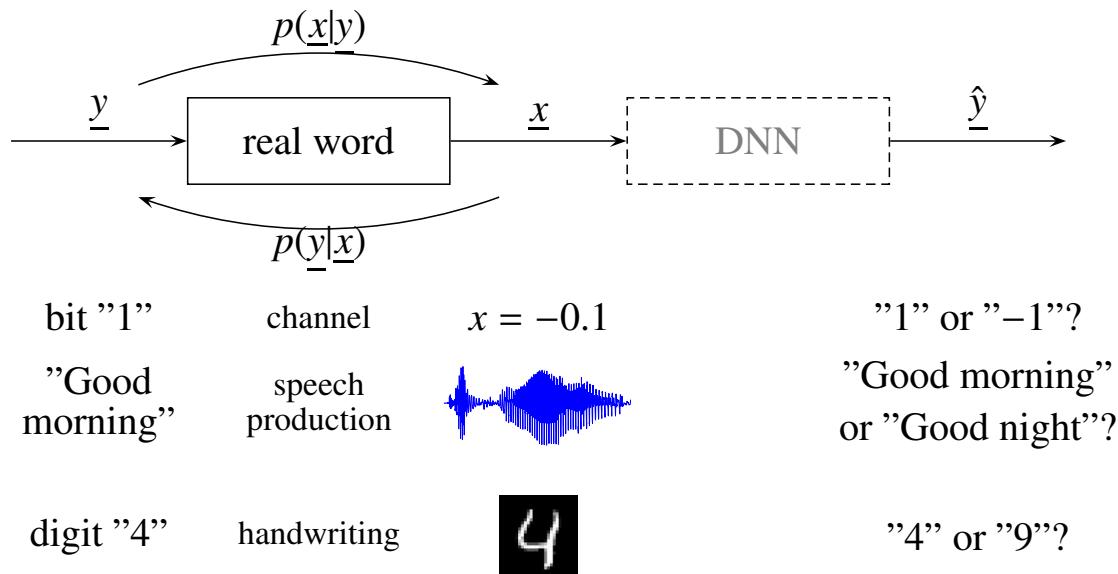
$$H(X, Y) = H(p, q) = -E_{X \sim p} \ln(q(X)) = -\int_{-\infty}^{\infty} p(x) \ln(q(x)) dx \geq 0.$$

+

---

<sup>3</sup>In information theory, the base-2 logarithm  $\log_2$  is used resulting in the entropy unit "bits". In machine learning, the natural logarithm  $\ln = \log_e$  is preferred leading to the unit "nats".

## Probabilistic framework of supervised learning



- $\underline{y}$ : **latent variable**, hidden, not directly measurable, quantity of interest
- $\underline{x}$ : **observed variable**, measurement, input for DNN
- $\hat{\underline{y}}$ : output of DNN as estimate for  $\underline{y}$
- real world: describes how is  $\underline{x}$  generated from  $\underline{y}$
- DNN: describes how to estimate  $\underline{y}$  from  $\underline{x}$

## The data generating distribution

Both  $\underline{y}$  and  $\underline{x}$  are modeled as random variables. They are described by the joint **data generating distribution**

$$p(\underline{x}, \underline{y}) = p(\underline{x}|\underline{y})p(\underline{y}) = p(\underline{y}|\underline{x})p(\underline{x}).$$

$p(\underline{y})$  prior PDF of  $\underline{y}$  or **prior**, available before any measurement of  $\underline{x}$

$p(\underline{x}|\underline{y})$  **likelihood**. It describes the real word, the generation of  $\underline{x}$  from  $\underline{y}$ .  
It is a kind of channel-sensor model, e.g.

- bit: communication channel + receiver
- speech: speech production system + microphone
- digit: handwriting + camera

$p(\underline{x})$  prior PDF of  $\underline{x}$ , also called **evidence**

$p(\underline{y}|\underline{x})$  posterior PDF of  $\underline{y}$  after a measurement  $\underline{x}$  or **posterior**

## Bayesian decision/estimation theory

In the model-based **Bayesian decision/estimation theory**, we assume to know the

- measurement  $\underline{x}$ ,
- likelihood  $p(\underline{x}|\underline{y})$ ,
- prior  $p(\underline{y})$ ,
- loss values  $l(\underline{y}, \hat{\underline{y}}(\underline{x}))$  for a correct decision  $\underline{y} = \hat{\underline{y}}$  or a wrong one  $\underline{y} \neq \hat{\underline{y}}$ .

There are different ways to estimate  $\underline{y}$  from  $\underline{x}$ :

a) **Maximum a posterior (MAP)** inference:

$$\max_{\underline{y}} p(\underline{y}|\underline{x}) = p(\underline{x}|\underline{y}) \frac{p(\underline{y})}{p(\underline{x})}$$

b) **Minimum Bayesian risk (MBR)** inference:

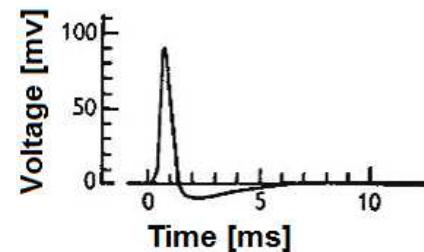
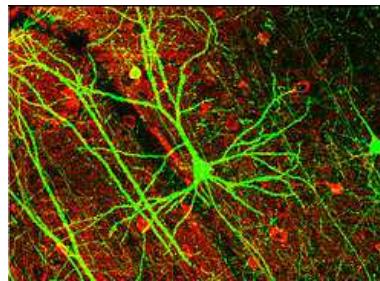
$$\min_{\hat{\underline{y}}} E_{\underline{X}, \underline{Y}}[l(\underline{Y}, \hat{\underline{y}}(\underline{X}))] = \int l(\underline{y}, \hat{\underline{y}}(\underline{x})) p(\underline{x}, \underline{y}) d\underline{x} d\underline{y}$$

see course DPR and SASP for more details.

## Why can we human learn so effectively?

Because we have a powerful brain.

- It is a biochemical and electrical network.
- It consists of a huge number of neurons (about 100 billions).
- The neurons are massively connected (one to several thousand in average).
- Each neuron shows a nonlinear input-output behavior: The neuron fires only if the input excitation is beyond a certain level.<sup>1</sup>



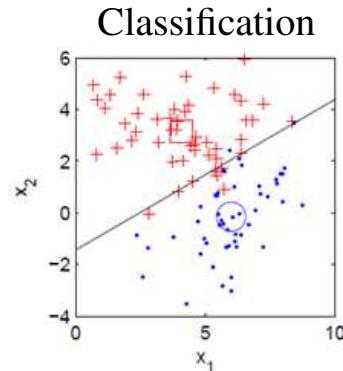
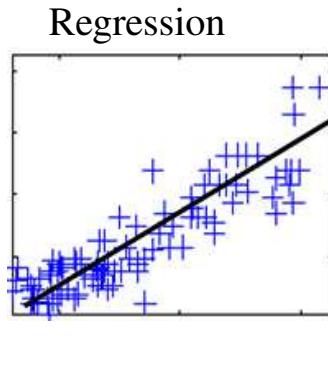
Our brain is able to learn, generalize and adapt, though we haven't fully understood its mechanisms yet.

<sup>1</sup>[www.wikipedia.org](http://www.wikipedia.org)

## Why nonlinear behavior of a neuron?

**Linear tasks:** The desired input-output mapping can be well described by a **linear function**  $y = \underline{\mathbf{A}}\underline{x}$  or an **affine function**  $y = \underline{\mathbf{A}}\underline{x} + \underline{b}$ .

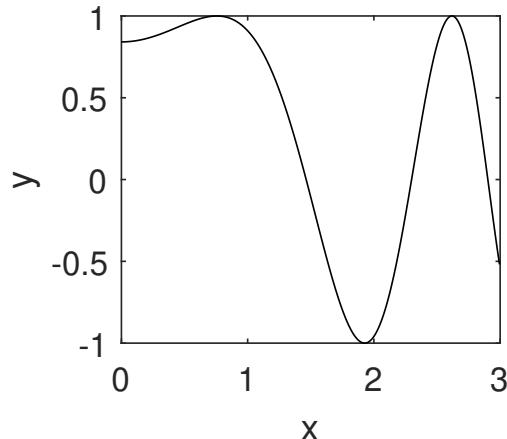
- Regression: The output is an affine function of the input.
- Classification: The decision boundary is described by an affine function (hyperplane).



In practice, most tasks are **nonlinear**.

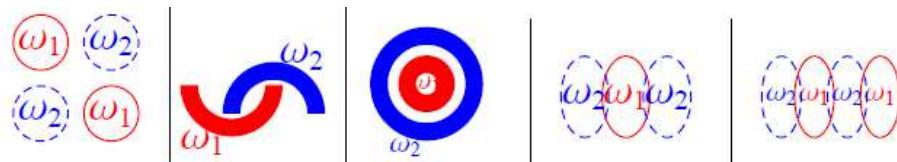
## E4.1: Nonlinear tasks

### a) Nonlinear regression

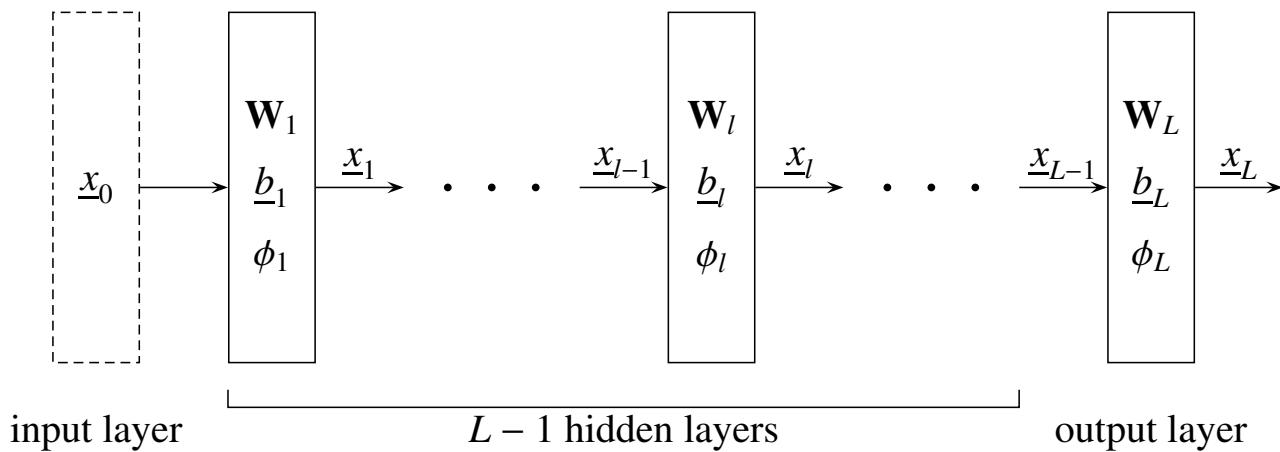


### b) Nonlinear classification

By drawing a straight line (linear decision boundary) in a two-dimensional feature space, it is impossible to separate both classes  $\omega_1$  and  $\omega_2$  in the following tasks.



## A feedforward multilayer neural network



- The input layer is not really a layer. It just provides the input  $x_0$  without any calculations.
- $L$  is the number of layers. The number of hidden layers is  $L - 1$ .
- Each dense layer  $l$  is characterized by its weight matrix  $\mathbf{W}_l$ , bias vector  $\underline{b}_l$  and activation function  $\phi_l$ ,  $1 \leq l \leq L$ .

## Perceptron vs. feedforward multilayer neural network

A feedforward multilayer neural network is an extension of the (linear) perceptron by using

- nonlinear activation functions and
- hidden layers.

The joint use of nonlinear activation functions and hidden layers is important:

- If  $\phi(a) = a$  for all neurons, each layer output  $\underline{x}_l$  is an affine function  $\mathbf{W}_l \underline{x}_{l-1} + \underline{b}_l$  of the layer input  $\underline{x}_{l-1}$ . As a consequence, the network output  $\underline{x}_L$  is an affine function of the network input  $\underline{x}_0$  and all hidden layers are useless and can be removed. The neural network simplifies to a layer of perceptrons.
- If there are no hidden layers, the network output is  $\underline{x}_1 = \phi(\mathbf{W}_1 \underline{x}_0 + \underline{b}_1)$ . Even if  $\phi(a)$  is nonlinear, this network is not able to approximate a general nonlinear input-output mapping. Often  $\phi(a)$  is monoton increasing and is useless for maximizing the network output in classification.

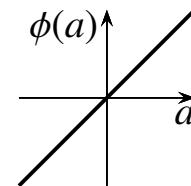
## Different activation functions

### 1) Linear or identity

$\phi(a) = a$ , i.e. no activation function.

It is used in the output layer for regression.

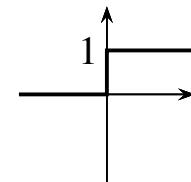
Such a neuron is often called a **linear neuron**.



### 2) Unit step

$$\phi(a) = u(a) = \begin{cases} 1 & a > 0 \\ 0 & a \leq 0 \end{cases}$$

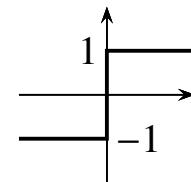
The neuron fires or not.



### 3) Sign function

$$\phi(a) = \text{sign}(a) = \begin{cases} 1 & \text{for } a > 0 \\ -1 & \text{for } a \leq 0 \end{cases} = 2u(a) - 1.$$

As 2), just another output range.

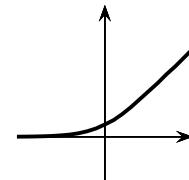


## Variants of ReLU

### 7) Softplus

$$\phi(a) = \ln(1 + e^a)$$

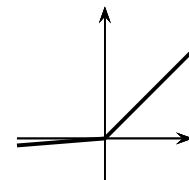
A smooth approximation to ReLU.



### 8) Leaky ReLU

$$\phi(a) = \begin{cases} a & \text{for } a \geq 0 \\ 0.01a & \text{for } a < 0 \end{cases}.$$

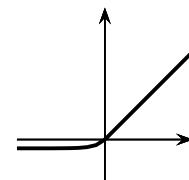
Nonzero gradient for  $a < 0$ .



### 9) Exponential linear unit (ELU)

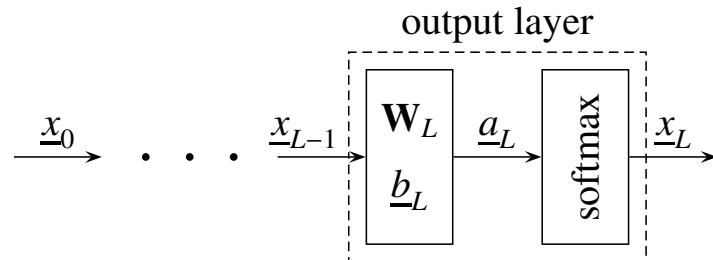
$$\phi(a) = \begin{cases} a & \text{for } a \geq 0 \\ \alpha(e^a - 1) & \text{for } a < 0 \end{cases}.$$

Nonzero gradient for  $a < 0$ .



## E4.2: Softmax activation function

The softmax activation function is used in the output layer of a DNN for classification:  $\underline{x}_L = \phi(\underline{a}_L) = \text{softmax}(\underline{a}_L)$ . It is used to transform  $\underline{a}_L \in \mathbb{R}^c$  to  $c$  probability values for  $c$  classes.



The definition of softmax is  $\phi(\underline{a}) = [\phi_i(\underline{a})]$  with

$$\phi_i(\underline{a}) = \frac{e^{a_i}}{\sum_{j=1}^c e^{a_j}}, \quad 1 \leq i \leq c.$$

Given  $\underline{a}_L = [1.8, -2.7, 3.1, 0.5]^T$  for a 4-class classification task, then  $\text{softmax}(\underline{a}_L) = [0.2019, 0.0022, 0.7408, 0.0550]^T$ . The class  $i = 3$  with the max. probability 74.08% is the predicted class.

## Activation function: When, where, which?

- when: regression or classification?
- where: hidden or output layer?
- which: which activation function?

	Regression	Classification
Hidden layers	any nonlinear activation function like ReLU, sigmoid, softmax etc	as left
Output layer	identity	softmax

- Softmax in the output layer is necessary for classification in order to transform any  $\underline{a} \in \mathbb{R}^c$  into  $c$  probability values.
- Nonlinear transform is not necessary in the output layer for regression.
- Nonlinear activation function in the hidden layers is fundamental for both regression and classification.

## Universal approximation theorem

The **universal approximation theorem** states that a feedforward neural network with a linear output layer ( $\phi_L(a) = a$ ) and

- *at least one hidden layer* with
- a *nonlinear* activation function

can approximate any continuous (nonlinear) function  $y(\underline{x}_0)$  (on compact input sets) to arbitrary accuracy.

Comments:

- arbitrary accuracy: with an increasing number of hidden neurons.
- valid for a wide range of nonlinear activation functions, but excluding polynomials.
- minimum requirement for universal approximation:  $\mathbf{W}_2\phi_1(\mathbf{W}_1\underline{x}_0 + \underline{b}_1) + \underline{b}_2$ .

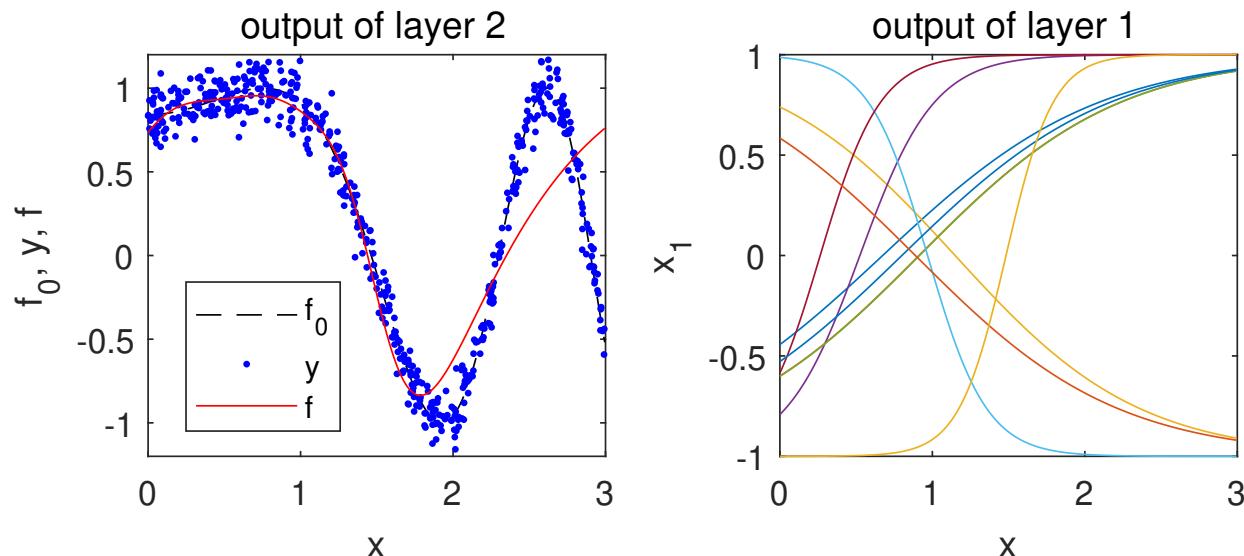
For learning  $\mathbf{W}_l$  and  $\underline{b}_l$  from  $D_{\text{train}}$ :

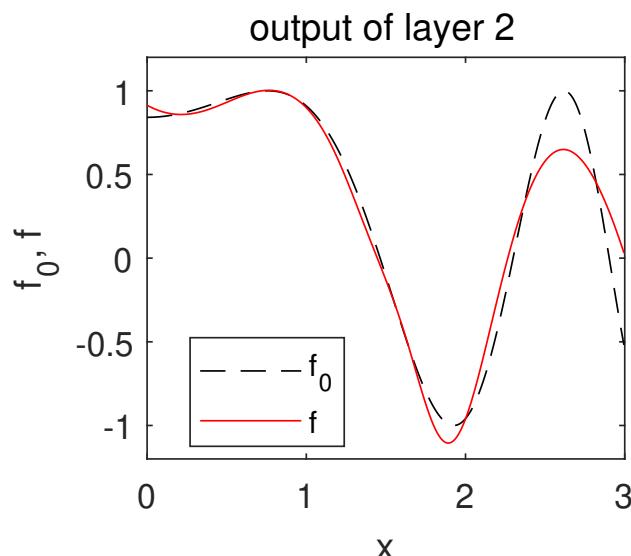
- deep networks are better than shallow ones,
- some activation functions are better than others.

**E4.3: Regression with 1 hidden layer (1)**

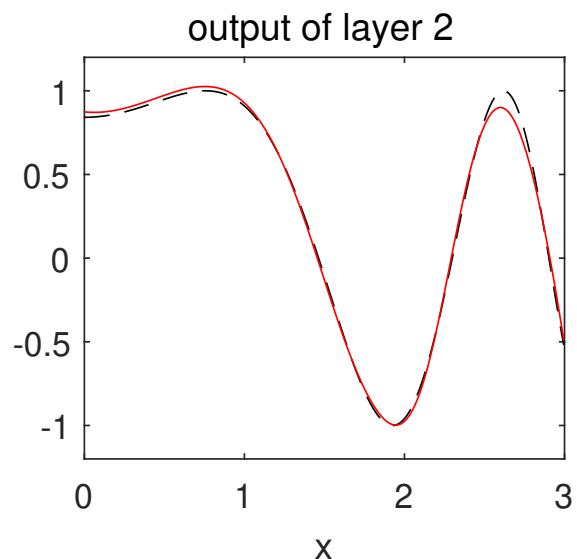
- true function  $f_0(x) = \sin(1 + x^2)$
- $N = 600$  noisy samples  $y(n) = f_0(x(n)) + N(0, 0.1^2)$  for  $0 \leq x(n) \leq 3$
- network output  $f(x)$  as approximation for  $f_0(x)$

a) 1 hidden layer with  $M_0 = M_2 = 1$  and  $M_1 = 10$  hidden neurons



**E4.3: Regression with 1 hidden layer (2)**b)  $M_1 = 100$  hidden neuronsc)  $M_1 = 500$  hidden neurons

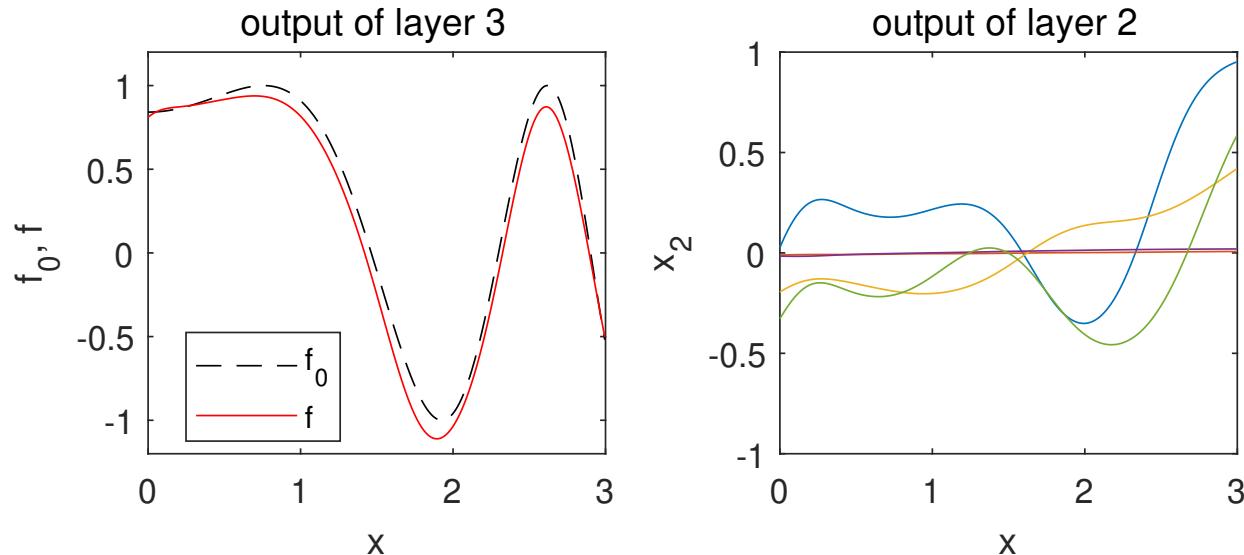
$$\sum_{i=1}^2 M_i(M_{i-1} + 1) = 1501 \text{ parameters}$$



The larger the number of hidden neurons, the better the approximation.

### E4.4: Regression with 2 hidden layers

- 2 hidden layers with  $M_0 = 1, M_1 = 100, M_2 = 5, M_3 = 1$
- $M_1 + M_2 = 105$  hidden neurons and  $\sum_{i=1}^3 M_i(M_{i-1} + 1) = 711$  parameters



This 3-layer network is as good as the previous 2-layer network, but has less hidden neurons and parameters.

## Case C: Laplace distribution and $l_1$ -loss

We assume a scalar  $x$  and  $y = f(x; \underline{\theta}) + z$ .

Case C:  $Z \sim \text{Laplace}(0, b)$

Then  $Y \sim \text{Laplace}(f(x; \underline{\theta}), b)$ , i.e.

$$q(y|x; \underline{\theta}) = \frac{1}{2b} e^{-\frac{1}{b}|y - f(x; \underline{\theta})|},$$

$$l(x, y; \underline{\theta}) = -\ln(q(y|x; \underline{\theta})) = \text{const} + \frac{1}{b}|y - f(x; \underline{\theta})|,$$

$$L(\underline{\theta}) = \frac{1}{N} \sum_{n=1}^N l(x(n), y(n); \underline{\theta}) = \text{const} + \frac{1}{Nb} \sum_{n=1}^N |y(n) - f(x(n); \underline{\theta})|.$$

This is the **mean absolute error (MAE) loss** or  $l_1$ -loss.

Note:

- The cost function  $L(\underline{\theta})$  depends on the underlying distribution of  $y - f(x; \underline{\theta})$ .
- We do not know the true distribution. You have to make an assumption.
- In all cases,  $y - f(x; \underline{\theta})$  is minimized in a certain sense.

## Output layer, loss and cost function

- The **loss**  $l(\underline{x}, \underline{y}; \underline{\theta})$  measures the quality of prediction for one pair  $(\underline{x}, \underline{y})$ .
- The **cost function**  $L(\underline{\theta}) = \frac{1}{N} \sum_{n=1}^N l(\underline{x}(n), \underline{y}(n); \underline{\theta})$  is the average loss over all  $N$  training samples.<sup>2</sup>

	Regression	Classification
Number of output neurons $M_L$	length of $\underline{y}$	number of classes $c$
Activation function $\phi_L$	identity	softmax
Loss $l(\underline{x}, \underline{y}; \underline{\theta})$	a) $l_2$ -loss or MSE-loss $\ \underline{y} - f(\underline{x}; \underline{\theta})\ ^2$	categorical loss $-\underline{y}^T \ln f(\underline{x}; \underline{\theta})$
	b) weighted $l_2$ -loss $(\underline{y} - f(\underline{x}; \underline{\theta}))^T \mathbf{C}^{-1} (\underline{y} - f(\underline{x}; \underline{\theta}))$	
	c) $l_1$ -loss or MAE-loss $ \underline{y} - f(\underline{x}; \underline{\theta}) $	

They are called **distribution-based loss** because they are derived from the distribution.

---

<sup>2</sup>In many text books, both terms loss and cost function are used synonymously.

## The probabilistic way toward the cost functions

- $p(\underline{x}, \underline{y})$ : true but unknown data generating distribution, application specific
- $D_{\text{train}} = \{\underline{x}(n), \underline{y}(n), 1 \leq n \leq N\}$ : training set, i.i.d. samples of  $p(\underline{x}, \underline{y})$
- $p(\underline{y}|\underline{x})$ : true posterior describing the desired inference  $\underline{x} \rightarrow \underline{y}$
- $q(\underline{y}|\underline{x}; \underline{\theta})$ : a parametric model (DNN) to approximate  $p(\underline{y}|\underline{x})$

min. forward KL divergence  $D_{\text{KL}}(p(\underline{x}, \underline{y}) \| q(\underline{x}, \underline{y}; \underline{\theta}))$

↓ ch. 3.3:  $p$  fixed

min. cross entropy  $H(p, q) = -\mathbb{E}_{\underline{X}, \underline{Y} \sim p(\underline{x}, \underline{y})} \ln(q(\underline{Y}|\underline{X}; \underline{\theta}))$

↓ ch. 3.4: use empirical distribution  $\hat{p}(\underline{x}, \underline{y})$

min. cross entropy  $H(\hat{p}, q) = -\mathbb{E}_{\underline{X}, \underline{Y} \sim \hat{p}(\underline{x}, \underline{y})} \ln(q(\underline{Y}|\underline{X}; \underline{\theta}))$

↓ ignore constant term

min. cost function  $L(\underline{\theta}) = \frac{1}{N} \sum_{n=1}^N [-\ln(q(\underline{y}(n)|\underline{x}(n); \underline{\theta}))]$

↓ ch. 4.6:  $q(\underline{y}|\underline{x}; \underline{\theta})$  for regression and classification?

$l_2$ -loss or  $l_1$ -loss or categorical loss

## Other losses for classification ★

a) Why not  $l_2$ -loss  $\|\underline{y} - f(\underline{x}; \underline{\theta})\|^2$  for classification?

Theoretically possible, but

- no probabilistic interpretation
- large deviations  $|y_i - f_i(\underline{x})|$  (outliers) heavily penalized due to  $(\cdot)^2$
- slow convergence during training

b) Why not 0/1-loss for classification?

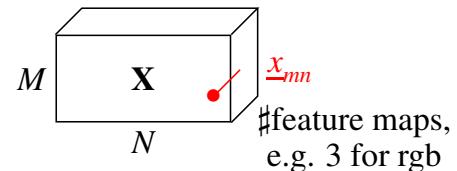
The **0/1-loss** is defined as

$$l(\underline{x}, y) = \begin{cases} 0 & \text{for } y = \text{estimated class label } f(\underline{x}) \\ 1 & \text{for } y \neq \text{estimated class label } f(\underline{x}) \end{cases}$$

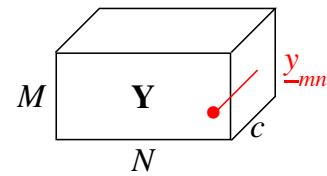
Then the cost function  $L(\underline{\theta})$  is the error rate of the classifier. This is an objective performance metric for classification, but is not differentiable for training.

## Image segmentation

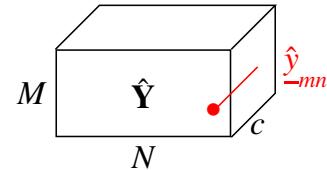
- 2D input image of size  $M \times N$ , 3D input tensor  $\mathbf{X} = [\underline{x}_{mn}]_{mn}$ . Assign each pixel  $\underline{x}_{mn}$  to one of  $c$  classes.



- class label tensor  $\mathbf{Y} = [\underline{y}_{mn}]_{mn}$ ,  
one-hot coding  $\underline{y}_{mn} \in \{\underline{e}_1, \dots, \underline{e}_c\}$  for pixel  $(m, n)$



- DNN output tensor  $\hat{\mathbf{Y}} = [\hat{y}_{mn}]_{mn} = f(\mathbf{X}; \underline{\theta})$ ,  
 $c$ -class softmax output  $\hat{y}_{mn} \in \mathbb{R}^c$  for pixel  $(m, n)$



How to define the loss  $l(\mathbf{X}, \mathbf{Y}; \underline{\theta})$  for image segmentation?

## Jaccard index and Dice coefficient (1)

Given two sets  $A$  and  $B$ , e.g. the sets of true and estimated object pixels in a binary segmentation, i.e. object vs. background. The **Jaccard index** or **intersection-over-union (IoU)** is defined as

$$0 \leq J = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \leq 1.$$

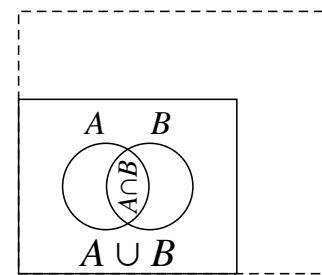
$A \cap B$  and  $A \cup B$  denote the **intersection** and **union** of  $A$  and  $B$ , respectively.  $|A|$  is the **cardinality** of  $A$ , i.e. the number of elements in  $A$ .<sup>3</sup>

The **Dice coefficient** has a slightly different definition

$$0 \leq D = \frac{2|A \cap B|}{|A| + |B|} \leq 1.$$

They are two similar region overlap metrics related by  $J = \frac{D}{2-D} \leq D$ . If  $A \cap B = \emptyset$ ,  $J = D = 0$ . If  $A = B$ ,  $J = D = 1$ .

In contrast to the categorical CE loss, there is no problem of imbalanced pixel classes.  $J$  and  $D$  are independent of the region size of the background.



<sup>3</sup>  $J$  is undefined if  $A = B = \emptyset$ . The normal convention is  $J = 1$  for this case.

## Jaccard index and Dice coefficient (2)

The previous Jaccard index and Dice coefficient are defined for one object class only. In a multi-class segmentation, they can be calculated for each of the  $c$  classes. Let  $A_i$  and  $B_i$  denote the sets of true and estimated pixels for class  $1 \leq i \leq c$  (e.g. human, vehicle, street, ...), respectively. Then

$$0 \leq J_i = \frac{|A_i \cap B_i|}{|A_i \cup B_i|} \leq 1$$

is the Jaccard index for class  $i$  and

$$J = \frac{1}{c} \sum_{i=1}^c J_i$$

is the **mean Jaccard index** or **mean IoU**. The Dice coefficient is defined in a similar way.

The above metrics are used for evaluation of the results of semantic segmentation. They are not suitable for training.

## Soft Jaccard and Dice loss

Let again  $\mathbf{Y} = [\underline{y}_{mn}]_{mn}$  and  $\hat{\mathbf{Y}} = [\hat{\underline{y}}_{mn}]_{mn}$ .  $\underline{y}_{mn} \in \{\underline{e}_1, \dots, \underline{e}_c\}$  is the one-hot coding and  $\hat{\underline{y}}_{mn} \in \mathbb{R}^c$  is the  $c$ -class softmax output for the pixel  $(m, n)$ . The **soft Jaccard loss** to be minimized in training for image segmentation is

$$\begin{aligned}\underline{\alpha} &= \sum_{m=1}^M \sum_{n=1}^N \underline{y}_{mn} \odot \hat{\underline{y}}_{mn} = [\alpha_i] \in \mathbb{R}^c, \\ \underline{\beta} &= \sum_{m=1}^M \sum_{n=1}^N (\underline{y}_{mn} + \hat{\underline{y}}_{mn}) = [\beta_i] \in \mathbb{R}^c, \\ J_i &= \frac{\alpha_i + \epsilon}{\beta_i - \alpha_i + \epsilon}, \\ l(\mathbf{X}, \mathbf{Y}; \underline{\theta}) &= 1 - \frac{1}{c} \sum_{i=1}^c J_i\end{aligned}$$

$\odot$  is the elementwise multiplication.  $\alpha_i$  and  $\beta_i$  represent arithmetic calculations of  $|A_i \cap B_i|$  and  $|A_i| + |B_i|$  for class  $i$ , respectively.  $J_i$  is the soft Jaccard index for class  $i$  where  $\epsilon > 0$  is a suitable number to avoid 0/0 if  $\alpha_i = \beta_i = 0$ .  $l(\mathbf{X}, \mathbf{Y}; \underline{\theta})$  is the soft Jaccard loss differentiable w.r.t.  $\underline{\theta}$ . The **soft Dice loss** is defined in a similar way.

For 3D segmentation,  $\underline{\alpha}$  and  $\underline{\beta}$  are calculated by three-dimensional sums over all pixels.

## Signal chain in DNN

- $\min_{\underline{\theta}} L(\underline{\theta}) = \frac{1}{N} \sum_{n=1}^N l(\underline{x}(n), \underline{y}(n); \underline{\theta})$
- $l(\underline{x}, \underline{y}; \underline{\theta})$  depends on the DNN output  $\underline{x}_L = f(\underline{x}; \underline{\theta})$  with  $\underline{x} = \underline{x}_0$ .
- $\underline{\theta}$  contains all parameters of  $\mathbf{W}_l = [w_{l,ij}]$  and  $\underline{b}_l = [b_{l,i}]$ ,  $1 \leq l \leq L$ .  
 $L(\underline{\theta})$  is a cascade of functions of the elements of  $\underline{\theta}$ .

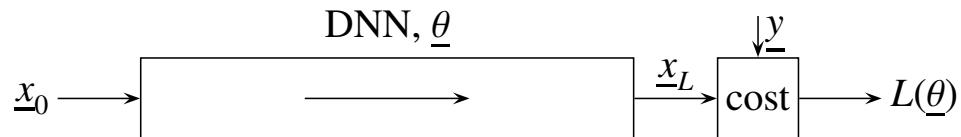
Layer  $L$ :  $\frac{\partial L(\underline{\theta})}{\partial w_{L,ij}}$ :

$$\underline{x}_0 \rightarrow \dots \rightarrow \underline{x}_{L-2} \xrightarrow[\underline{b}_{L-1}]{} \underline{a}_{L-1} \xrightarrow[\phi_{L-1}]{} \underline{x}_{L-1} \xrightarrow[\underline{b}_L]{} \underline{a}_L \xrightarrow[\phi_L]{} \underline{x}_L \rightarrow L(\underline{\theta})$$

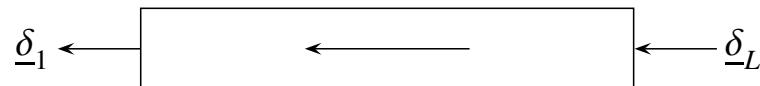
Layer  $L - 1$ :  $\frac{\partial L(\underline{\theta})}{\partial w_{L-1,ij}}$ :

## Forward pass vs. backward pass in DNN

**Forward pass** to calculate  $\underline{x}_L$  from  $\underline{x}_0$ :



**Backward pass** or **backpropagation** of so called **error vectors**  $\underline{\delta}_l^T := \mathbf{J}_L(\underline{a}_l) = \frac{\partial L(\underline{\theta})}{\partial \underline{a}_l} \in \mathbb{R}^{1 \times M_l}$ :



For  $l = L - 1, \dots, 1$

$$\underline{\delta}_l^T = \underline{\delta}_{l+1}^T \cdot \mathbf{J}_{\underline{a}_{l+1}}(\underline{x}_l) \mathbf{J}_{\underline{x}_l}(\underline{a}_l) = \boxed{\quad} \cdot \boxed{\quad}$$

$$\frac{\partial L(\underline{\theta})}{\partial w_{l,ij}} = \underline{\delta}_l^T \cdot \mathbf{J}_{\underline{a}_l}(w_{l,ij}) = \boxed{\quad} \cdot \boxed{\quad}$$

$$\frac{\partial L(\underline{\theta})}{\partial b_{l,i}} = \underline{\delta}_l^T \cdot \mathbf{J}_{\underline{a}_l}(b_{l,i}) = \boxed{\quad} \cdot \boxed{\quad}$$

## Jacobi vectors/matrices during backpropagation (1)

a) **Loss:**  $\underline{x}_L \rightarrow$  cost function  $L(\underline{\theta})$

$$\mathbf{J}_L(\underline{x}_L) = \frac{\partial L(\underline{\theta})}{\partial \underline{x}_L} = \left[ \frac{\partial L}{\partial x_{L,1}} \cdots \frac{\partial L}{\partial x_{L,M_L}} \right] \in \mathbb{R}^{1 \times M_L},$$

$$L(\underline{\theta}) = \frac{1}{N} \sum_{n=1}^N l(\underline{x}(n), \underline{y}(n); \underline{\theta}),$$

$$\frac{\partial L(\underline{\theta})}{\partial x_{L,i}} = \frac{1}{N} \sum_{n=1}^N \frac{\partial l(\underline{x}(n), \underline{y}(n); \underline{\theta})}{\partial x_{L,i}},$$

$$l(\underline{x}, \underline{y}; \underline{\theta}) = \begin{cases} \|\underline{y} - \underline{x}_L\|^2 & \text{for } l_2\text{-loss} \\ -\underline{y}^T \ln(\underline{x}_L) & \text{for categorical loss} \end{cases},$$

$$\frac{\partial l(\underline{x}, \underline{y}; \underline{\theta})}{\partial x_{L,i}} = \begin{cases} -2(y_i - x_{L,i}) & \text{for } l_2\text{-loss} \\ -\frac{y_i}{x_{L,i}} & \text{for categorical loss} \end{cases}.$$

## Jacobi vectors/matrices during backpropagation (2)

b) **Activation function:**  $\underline{a}_l \rightarrow \underline{x}_l = \phi_l(\underline{a}_l)$ ,  $1 \leq l \leq L$

$$\mathbf{J}_{\underline{x}_l}(\underline{a}_l) = \frac{\partial \underline{x}_l}{\partial \underline{a}_l} = \begin{bmatrix} \frac{\partial x_{l,1}}{\partial a_{l,1}} & \dots & \frac{\partial x_{l,1}}{\partial a_{l,M_l}} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_{l,M_l}}{\partial a_{l,1}} & \dots & \frac{\partial x_{l,M_l}}{\partial a_{l,M_l}} \end{bmatrix} \in \mathbb{R}^{M_l \times M_l}.$$

According to the derivative calculation of activation functions in ch. 4.4,

- Hidden layer  $l$ , ReLU  $\phi_l(a) = \phi(a) = \max(0, a)$  with  $\phi'(a) = u(a)$ :

$$\mathbf{J}_{\underline{x}_l}(\underline{a}_l) = \text{diag}(\phi'(a_{l,1}), \dots, \phi'(a_{l,M_l})) = \text{diag}(u(\underline{a}_l)).$$

- Hidden layer  $l$ , sigmoid  $\phi_l(a) = \sigma(a)$  with  $\sigma'(a) = \sigma(a)(1 - \sigma(a))$ :

$$\mathbf{J}_{\underline{x}_l}(\underline{a}_l) = \text{diag}(\sigma'(a_{l,1}), \dots, \sigma'(a_{l,M_l})) = \text{diag}(\underline{x}_l - \underline{x}_l \odot \underline{x}_l).$$

- Output layer  $L$ , identity activation function with  $\underline{x}_L = \underline{a}_L$ :

$$\mathbf{J}_{\underline{x}_L}(\underline{a}_L) = \mathbf{I}.$$

## Jacobi vectors/matrices during backpropagation (3)

- Output layer  $L$ , softmax  $\phi_L(\underline{a}_L) = \text{softmax}(\underline{a}_L)$ :

$$\mathbf{J}_{\underline{x}_L}(\underline{a}_L) = \text{diag}(\underline{x}_L) - \underline{x}_L \underline{x}_L^T.$$

- c) **Layer transition**  $\underline{x}_{l-1} \rightarrow \underline{a}_l = \mathbf{W}_l \underline{x}_{l-1} + \underline{b}_l$ ,  $1 \leq l \leq L$ :

$$\mathbf{J}_{\underline{a}_l}(\underline{x}_{l-1}) = \frac{\partial \underline{a}_l}{\partial \underline{x}_{l-1}} = \mathbf{W}_l \in \mathbb{R}^{M_l \times M_{l-1}}.$$

- d) **Weight**  $w_{l,ij} \rightarrow \underline{a}_l = \mathbf{W}_l \underline{x}_{l-1} + \underline{b}_l$ ,  $1 \leq l \leq L$ :

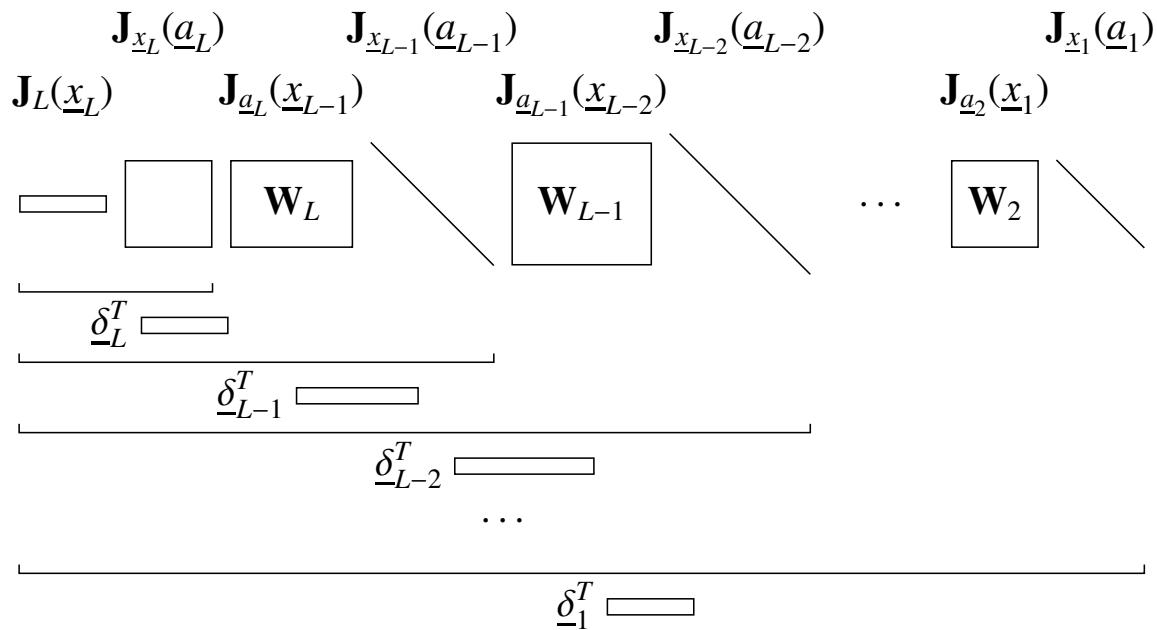
$$\mathbf{J}_{\underline{a}_l}(w_{l,ij}) = \frac{\partial \underline{a}_l}{\partial w_{l,ij}} = [0, \dots, 0, \underbrace{1}_{i-th \text{ position}}, 0, \dots, 0]^T = x_{l-1,j} \underline{u}_i \in \mathbb{R}^{M_l \times 1}.$$

- e) **Bias**  $b_{l,i} \rightarrow \underline{a}_l = \mathbf{W}_l \underline{x}_{l-1} + \underline{b}_l$ ,  $1 \leq l \leq L$ :

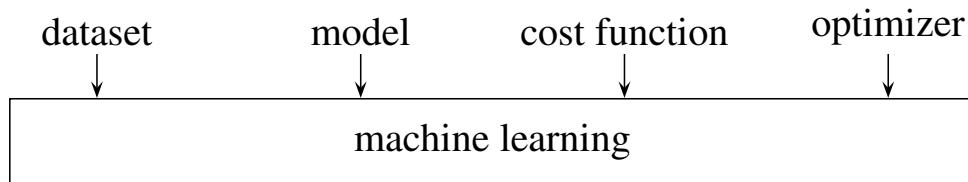
$$\mathbf{J}_{\underline{a}_l}(b_{l,i}) = \frac{\partial \underline{a}_l}{\partial b_{l,i}} = [0, \dots, 0, \underbrace{1}_{i-th \text{ position}}, 0, \dots, 0]^T = \underline{u}_i \in \mathbb{R}^{M_l \times 1}.$$

## Jacobi vectors/matrices during backpropagation (4)

$$\underline{\delta}_l^T = \underline{\delta}_{l+1}^T \cdot \mathbf{J}_{\underline{a}_{l+1}}(\underline{x}_l) \mathbf{J}_{\underline{x}_l}(\underline{a}_l) = \underline{\delta}_{l+1}^T \cdot \mathbf{W}_{l+1} \mathbf{J}_{\underline{x}_l}(\underline{a}_l), \quad l = L-1, \dots, 1$$



## Four components for machine learning

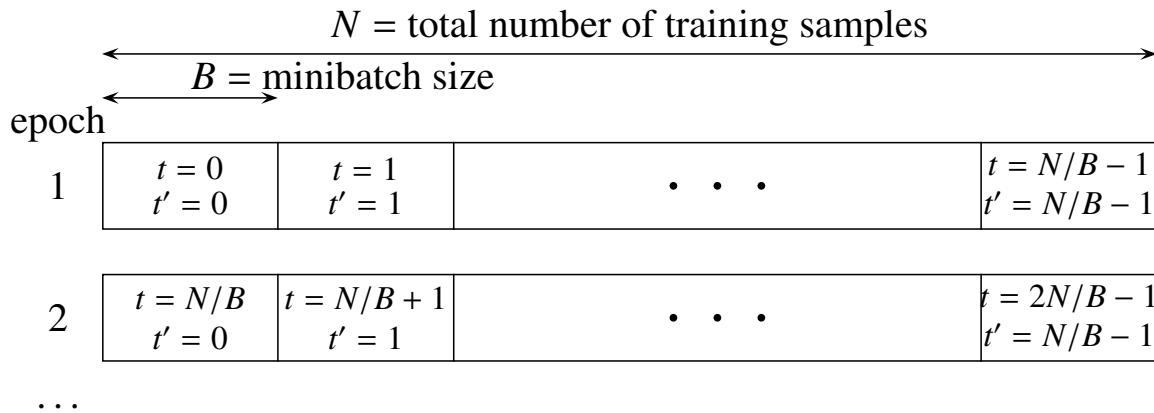


	Description	Deep learning	Other possibilities
Dataset	for training and test		
Model	A parametric model $q(\underline{y} \underline{x}; \underline{\theta})$ as an approximation for the posterior $p(\underline{y} \underline{x})$	DNN with the parameter vector $\underline{\theta}$	SVM, GMM
Cost function	$L(\underline{\theta})$ measures the deviation of the model output from ground truth	mostly $l_2$ or categorical loss	Jaccard/Dice loss, hinge loss, ...
Optimizer	optimization algorithm to minimize $L(\underline{\theta})$	gradient descent or variants	Newton, ...

You can combine these four components arbitrarily.

## Epoch and minibatch

For a better convergence, one needs to pass the training set  $D_{\text{train}}$  multiple times to the DNN. One **epoch** denotes one pass of  $D_{\text{train}}$  through the DNN. It consists of  $N/B$  minibatches of size  $B$ .<sup>4</sup>



- For each epoch,  $t' = \text{mod}(t, \frac{N}{B}) = 0, 1, \dots, \frac{N}{B} - 1$  is the minibatch index.
- For each minibatch  $t'$ ,  $t'B + 1 \leq n \leq (t' + 1)B$  is the sample index for  $\underline{x}(n), \underline{y}(n)$ .

+

<sup>4</sup>If  $N$  is not an integer multiple of  $B$ , there are  $\lceil N/B \rceil$  minibatches. The last one contains less than  $B$  samples.

## E4.5: MNISTnet1 – Baseline network (1)

The first neural network in this course for digit recognition on MNIST. It serves as a baseline network for further studies.

### Dataset: MNIST

- gray images for 10 digits  $0, 1, \dots, 9$
- 60,000 training images and 10,000 test images of size 28x28

### Network: MNISTnet1

- 1D fully connected neural network
- reshape each 28x28 gray image into a column vector  $\underline{x}_0(n) \in \mathbb{R}^{784}$
- input layer with  $M_0 = 784$  neurons
- 1 hidden layer with  $M_1 = 1024$  neurons and tanh activation function
- $M_2 = 10$  neurons in the output layer with softmax activation function
- $N_p = M_1(M_0 + 1) + M_2(M_1 + 1) = 814,090$  parameters
- $N_x = M_1M_0 + M_2M_1 = 813,056$  multiplications

## E4.5: MNISTnet1 – Baseline network (2)

### Import Python package and modules

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
```

### Load training and test images and vectorize them

$$\mathbf{X}(n) \in \mathbb{R}^{28 \times 28} \rightarrow \underline{x}_0(n) = \text{vec}(\mathbf{X}(n)) \in \mathbb{R}^{784}, \quad n = 1, \dots, 70,000$$

```
(x_train,y_train),(x_test,y_test) = mnist.load_data()
x_train = x_train.reshape(60000,784).astype('float32')
x_test = x_test.reshape(10000,784).astype('float32')
```

### E4.5: MNISTnet1 – Baseline network (3)

#### Generate class labels in one-hot coding

$$y(n) \in \{1, 2, \dots, 10\} \rightarrow \underline{y}(n) \in \{\underline{e}_1, \underline{e}_2, \dots, \underline{e}_{10}\}$$

```
y_train = keras.utils.to_categorical(y_train, num_classes=10)
y_test = keras.utils.to_categorical(y_test, num_classes=10)
```

#### Define the network

$$\begin{aligned}\underline{x}_0 &= \underline{x} \\ \underline{a}_l &= \mathbf{W}_l \underline{x}_{l-1} + \underline{b}_l, \quad l = 1, 2 \\ \underline{x}_1 &= \tanh(\underline{a}_1) \\ \underline{x}_2 &= \text{softmax}(\underline{a}_2) = f(\underline{x}; \underline{\theta})\end{aligned}$$

```
model = Sequential()
model.add(Dense(1024, input_shape=(784), activation='tanh'))
model.add(Dense(10, activation='softmax'))
```

## E4.5: MNISTnet1 – Baseline network (4)

**Define the loss**, the categorical cross entropy, and **choose the optimizer**, the stochastic gradient descent (SGD). You will learn step size lr, momentum, decay, nesterov etc. in ch. 5.

$$\underline{\theta}^{t+1} = \underline{\theta}^t - \frac{\gamma^t}{B} \sum_{n=t'B+1}^{(t'+1)B} \nabla l(\underline{x}(n), \underline{y}(n); \underline{\theta}) \Big|_{\underline{\theta}=\underline{\theta}^t}, \quad t' = \text{mod}(t, N/B)$$

```
sgd = optimizers.SGD(lr=0.1, momentum=0.0, decay=0.0,
                      nesterov=False)
model.compile(loss='categorical_crossentropy',
               optimizer=sgd, metrics=['accuracy'])
```

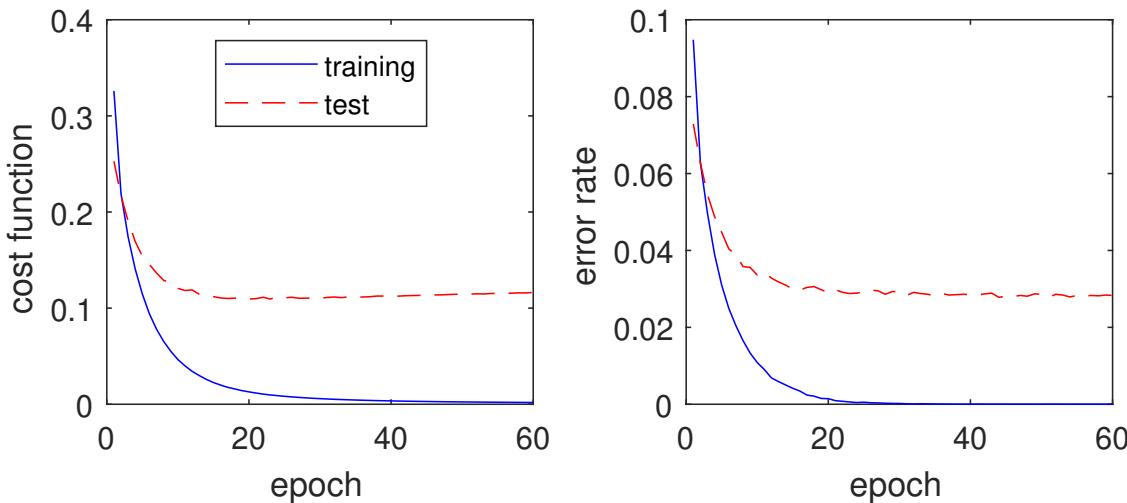
**Training and test** using minibatch size  $B = 128$  and 60 epochs

```
history = model.fit(x_train, y_train, batch_size=128,
                     epochs=60, verbose=1, validation_data=(x_test, y_test))
```

## E4.5: MNISTnet1 – Baseline network (5)

### Results

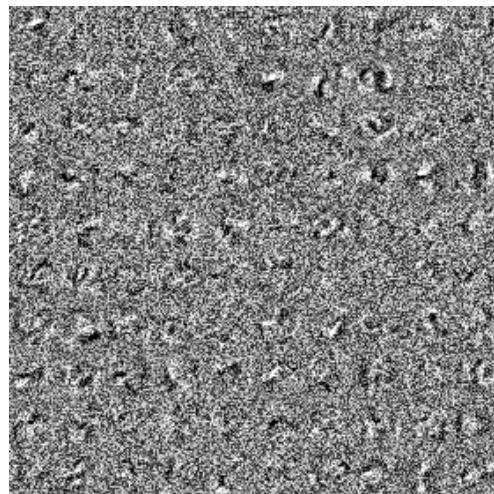
- technical metric: training/test cost function  $L(\theta)$  after each epoch
- objective metric: training/test classification error rate after each epoch
- almost zero training error rate
- but large gap between training and test error rate → overfitting



### E4.5: MNISTnet1 – Baseline network (6)

**Visualization of  $\mathbf{W}_1 \in \mathbb{R}^{1024 \times 784}$  after training**

- take the first 100 rows of  $\mathbf{W}_1$ : weight vectors of the first 100 hidden neurons
- normalize each vector of length 784 by its  $l_2$ -norm and reshape it to a 28x28 weight matrix
- arrange 100 weight matrices in a 10x10 grid, resulting in a 280x280 image
- the learned weight vectors are quite noisy and visually difficult to interpret



## Difficulties in optimization

Training a NN, in particular a DNN, is a challenging task and suffers from a number of optimization difficulties. Some of these difficulties are common to all optimization (OPT) problems, while some others are special to the training of a DNN.

Difficulty	OPT	DNN	Solutions
D1) stochastic gradient	×	×	larger minibatch size $B$ , 5.2
D2) ill conditioning	×	×	5.2, 5.4
D3) saddle point / plateau	×	×	noisy gradient
D4) sensitive to step size	×	×	5.3, 5.4
D5) local minimum	×	×	5.5, 5.6
D6) vanishing gradient		×	5.6

## Basics of optimization

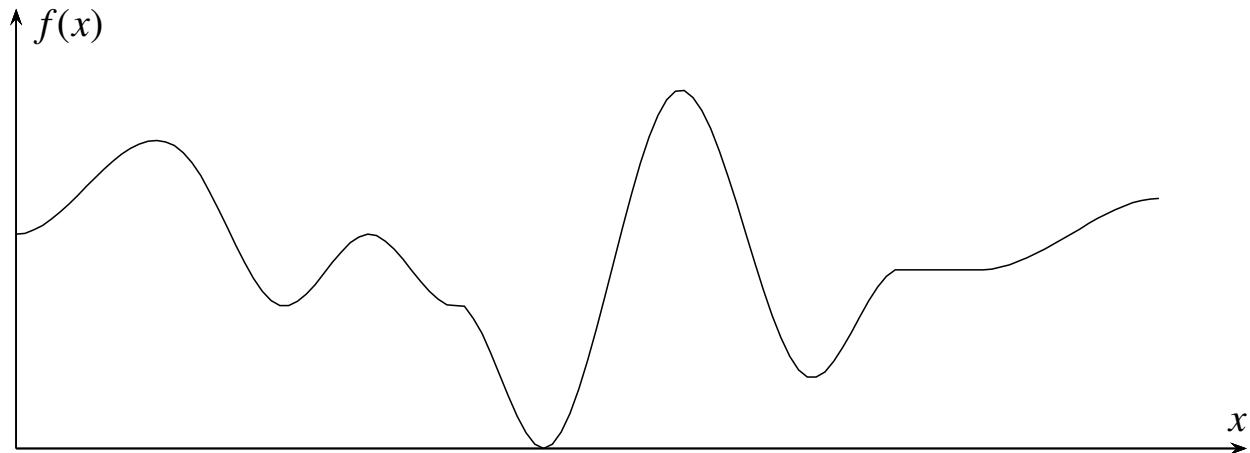
Given a differentiable function  $f(\underline{x}) : \underline{x} = [x_i] \in \mathbb{R}^N \rightarrow f \in \mathbb{R}$ .

- **gradient vector**  $\nabla f(x)$  and **Hessian matrix**  $\mathbf{H}(x)$ :

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_N} \end{bmatrix} \in \mathbb{R}^N, \quad \mathbf{H}(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_N \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_N^2} \end{bmatrix} \in \mathbb{R}^{N \times N}$$

- $\underline{x}^*$  is a **local minimum** of  $f(\underline{x})$  if there is no smaller value than  $f(\underline{x}^*)$  in a local neighborhood of  $\underline{x}^*$ , i.e.  $f(\underline{x}) \geq f(\underline{x}^*) \forall \|\underline{x} - \underline{x}^*\| \leq \delta$ .
- $\underline{x}^*$  is a **global minimum** of  $f(\underline{x})$  if there is no smaller value than  $f(\underline{x}^*)$  in  $\mathbb{R}^N$ , i.e.  $f(\underline{x}) \geq f(\underline{x}^*) \forall \underline{x} \in \mathbb{R}^N$ .
- $\underline{x}^*$  is a **local maximum** of  $f(\underline{x})$  if  $f(\underline{x}) \leq f(\underline{x}^*) \forall \|\underline{x} - \underline{x}^*\| \leq \delta$ .
- $\underline{x}^*$  is a **stationary point** of  $f(\underline{x})$  if  $\nabla f(\underline{x}^*) = \underline{0}$ .
- $\underline{x}^*$  is a **saddle point** of  $f(\underline{x})$  if it is a stationary point, but neither a local minimum nor a local maximum.

## 1D visualization



- + desired global minimum
  - 1 local minima
  - 2 local maxima
  - 3 saddle points
  - 4 **plateau**, region of constant  $f(\underline{x})$
- stationary points

## Conditions on a local minimum

- Necessary condition for  $\underline{x}^*$  being a local minimum:  
 $\underline{x}^*$  is a stationary point, i.e.  $\nabla f(\underline{x}^*) = \underline{0}$
- If  $f(\underline{x})$  is convex: The above condition is also sufficient for  $\underline{x}^*$  being a global minimum.
- If  $f(\underline{x})$  is non-convex: The above condition is only necessary. The Hessian matrix  $\mathbf{H}(\underline{x}^*)$  is required to distinguish between different types of stationary point:
  - ◊  $\mathbf{H}(\underline{x}^*)$  non-negative definite:  $\underline{x}^*$  local minimum
  - ◊  $\mathbf{H}(\underline{x}^*)$  non-positive definite:  $\underline{x}^*$  local maximum
  - ◊  $\mathbf{H}(\underline{x}^*)$  indefinite:  $\underline{x}^*$  saddle point
- Even if  $\underline{x}^*$  is a local minimum, there is no guarantee that it is a global minimum.

See course AM for more details.

The cost function  $L(\underline{\theta})$  of a DNN with nonlinear activation functions is non-convex.

## Gradient descent minimization

Task:  $\min_{\underline{x}} f(\underline{x})$

Gradient descent minimization

Parameter: step size  $\gamma^t > 0$

Initial value:  $\underline{x}^0$

FOR  $t = 0, 1, \dots$  DO

$$\underline{x}^{t+1} = \underline{x}^t - \gamma^t \nabla f(\underline{x})|_{\underline{x}=\underline{x}^t}$$



1

Properties:

- It is a local search.
- $-\nabla f(\underline{x})$  points to the direction of steepest descent at the current position  $\underline{x}$ . This is, however, often not the direction of the local minimum.
- $-\nabla f(\underline{x})$  orthogonal to the tangent of the contour line.
- No need to calculate Hessian matrix  $\mathbf{H}(\underline{x})$  and  $\mathbf{H}^{-1}(\underline{x})$  and hence much simpler than second-order methods like Newton method.
- It is like hiking downhill in fog without GPS and map.

<sup>1</sup><http://www.timetoclimb.com/hiking>

## D1) Stochastic gradient

The gradient vector of a batch cost function for a NN

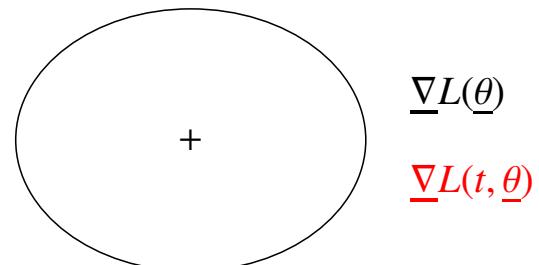
$$\underline{\nabla}L(\underline{\theta}) = \frac{1}{N} \sum_{n=1}^N \underline{\nabla}l(\underline{x}(n), \underline{y}(n); \underline{\theta})$$

is calculated over the complete set of  $N$  training samples. If  $N$  is large,  $\underline{\nabla}L(\underline{\theta})$  is almost deterministic.

When using minibatches, the gradient vector of the minibatch cost function

$$\underline{\nabla}L(t; \underline{\theta}) = \frac{1}{B} \sum_{n=t' B+1}^{(t'+1)B} \underline{\nabla}l(\underline{x}(n), \underline{y}(n); \underline{\theta}), \quad t' = \text{mod}(t, N/B)$$

is averaged over a smaller number of  $B \ll N$  samples. It is a stochastic gradient with a noisy update direction. This leads to a slow convergence.



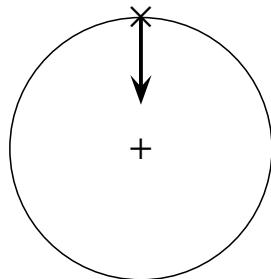
## D2) Ill conditioning

**well conditioned** (circular) contour lines

equal curvatures

$-\nabla L(t; \theta)$  points to local minimum

fast convergence

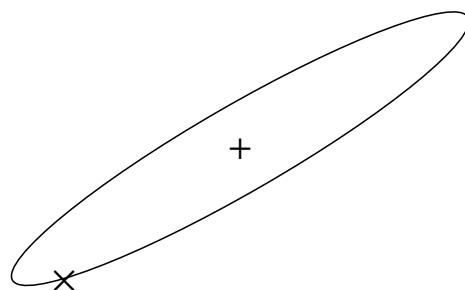


**ill conditioned** (narrow) contour lines

strongly different curvatures

$-\nabla L(t; \theta)$  mostly points to wrong directions

slow convergence (oscillation)



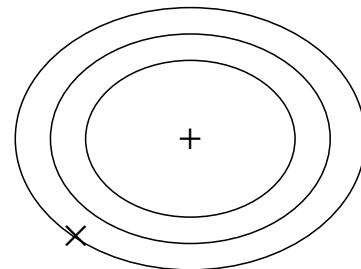
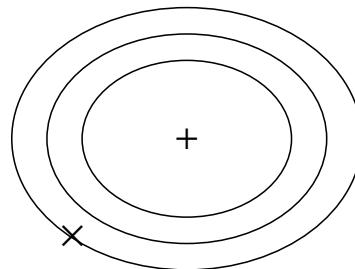
## D4) Sensitive to the choice of the step size

too small step size  $\gamma^t$

slow convergence

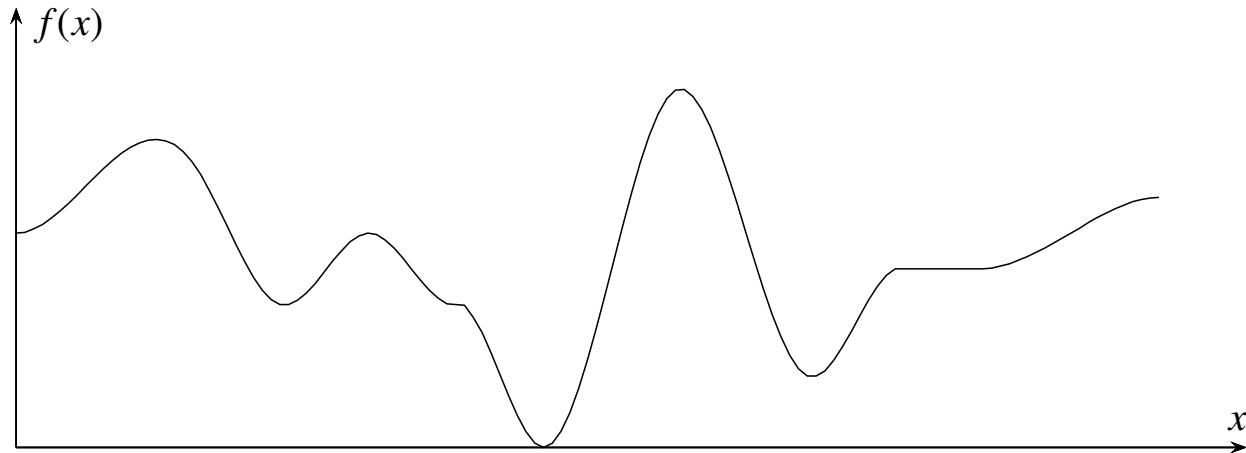
too large step size  $\gamma^t$

oscillation and no convergence



- The optimum step size  $\gamma^t$  depends on the cost function  $L(t; \underline{\theta})$  and the current position  $\underline{\theta}^t$  and is unknown in advance.

## D5) Local minimum



The gradient descent method often converges to a local minimum. There is no guarantee to find the global minimum. But a local minimum is not always bad.

- Local minimum with  $L(t; \underline{\theta})$  comparable to that of the global minimum: Acceptable
- Local minimum with  $L(t; \underline{\theta})$  much larger than that of the global minimum: Bad
- You do not know whether it is a good or bad local minimum.

## DNNs make D1-D5 even more serious

The difficulties D1–D5 are not new. They appear in all gradient descent minimization problems. In deep learning, however, they are even more serious because a DNN

- contains a very large number of parameters in  $\underline{\theta}$ ,
- consists of a deep cascade of nonlinear functions,
- is often over-designed (more parameters than necessary) in order to guarantee a satisfied performance for a given task.

As a result, the cost function  $L(\underline{\theta})$  of a DNN has

- a huge number of local minima and saddle points,
- plateaus (some parameters have a very small influence on  $L(\underline{\theta})$ ),
- many equivalent minima due to non-unique solutions (e.g. weight symmetry).

This makes the gradient descent training of a DNN much more difficult.

## Digital filters



There are two types of digital filter with the input signal  $x(n)$  and output signal  $y(n)$ :

- a) **Nonrecursive filter** or **FIR filter**  $\text{FIR}(M)$ :

$$y(n) = b_0x(n) + b_1x(n - 1) + \dots + b_Mx(n - M).$$

It has a finite (duration) impulse response (FIR) and the frequency response

$$H(\omega) = b_0 + b_1e^{-j\omega} + \dots + b_Me^{-jM\omega}.$$

- b) (Purely) **recursive filter** or **IIR filter**  $\text{IIR}(N, 0)$ :

$$y(n) + a_1y(n - 1) + \dots + a_Ny(n - N) = x(n).$$

It has an infinite (duration) impulse response (IIR) and the frequency response

$$H(\omega) = \frac{1}{a_0 + a_1e^{-j\omega} + \dots + a_Ne^{-jN\omega}},$$

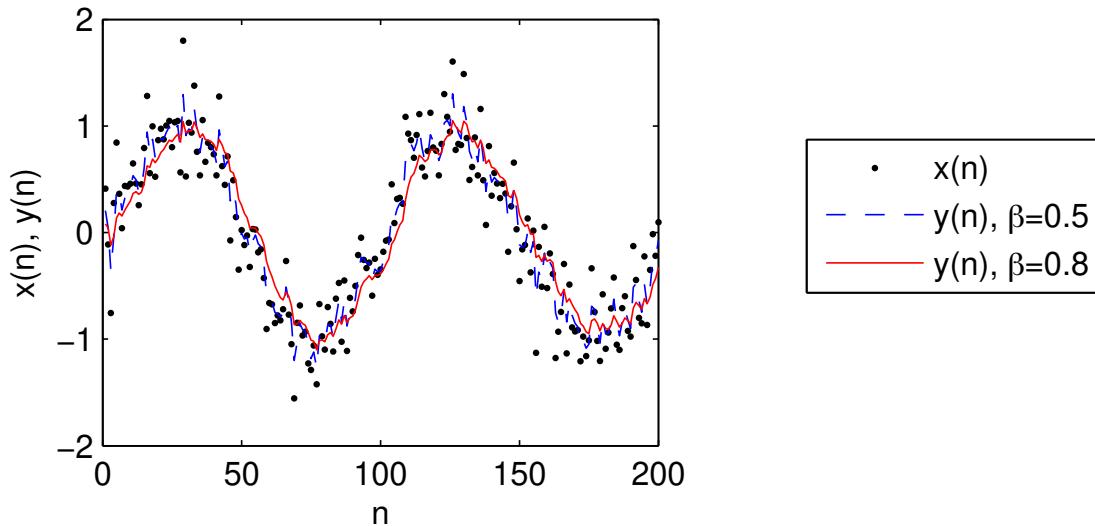
see Bachelor course "Digital signal processing".

## IIR(1,0) filter for recursive smoothing

With the coefficient  $0 < \beta = -a_1 < 1$ , the IIR(1,0) filter

$$y(n) = \beta y(n-1) + x(n) = x(n) + \beta x(n-1) + \beta^2 x(n-2) + \beta^3 x(n-3) + \dots$$

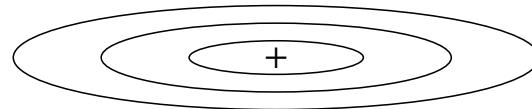
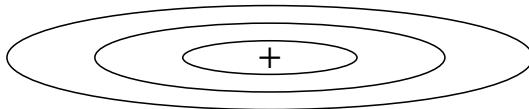
performs an **exponentially weighted average** of the input signal  $x(n)$  and returns a smoothed output  $y(n)$ . The larger  $\beta$ , the stronger the smoothing effect.<sup>2</sup>



+

<sup>2</sup>In the plot,  $y(n)$  is scaled with  $1 - \beta$  in order to compensate a growing amplitude of  $y(n)$ . In the momentum method, this scaling can be incorporated into the step size  $\gamma'$ .

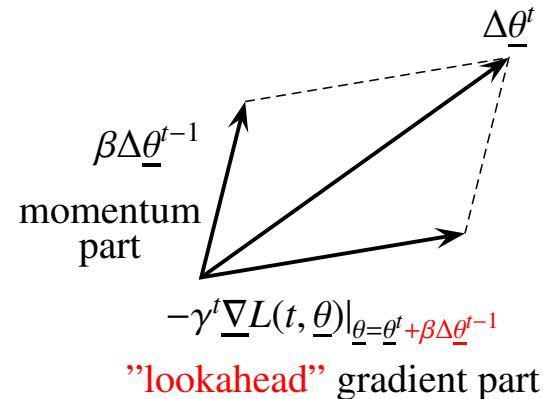
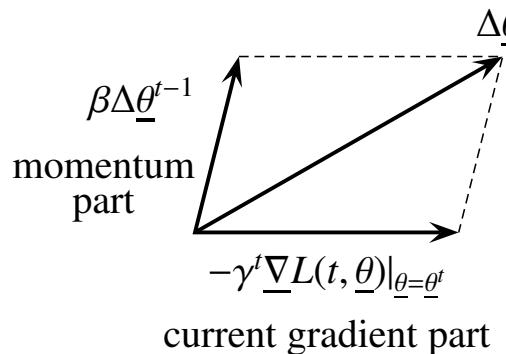
## Stochastic gradient descent vs. momentum



SGD	SGD with momentum
strong oscillation in undesired directions	reduced oscillation
slow convergence in desired direction	accelerated convergence in desired direction

Momentum performs a recursive smoothing of the stochastic gradients and accumulates the common gradient component along the desired (horizontal) direction while reducing oscillations along the undesired (vertical) direction. This accelerates the convergence.

## Momentum vs. Nesterov momentum

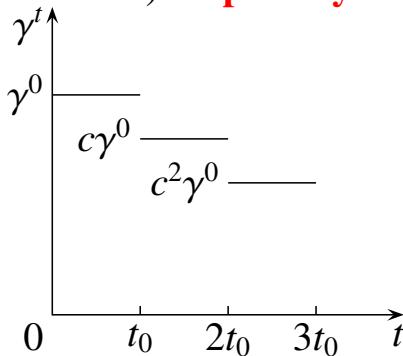


Nesterov momentum is like the momentum method except for the difference that the gradient vector  $\nabla L(t; \underline{\theta})$  is not calculated at the current position  $\underline{\theta} = \underline{\theta}^t$ , but rather at the "lookahead" position  $\underline{\theta} = \underline{\theta}^t + \beta \Delta \underline{\theta}^{t-1}$  after the application of the momentum  $\beta \Delta \underline{\theta}^{t-1}$  to  $\underline{\theta}^t$ .

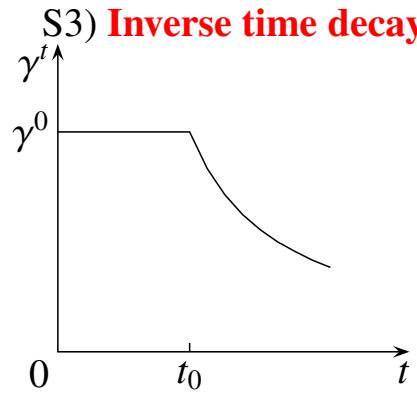
## Static schedules

**Static schedules:**  $\gamma^t$  fixed and independent of parameters

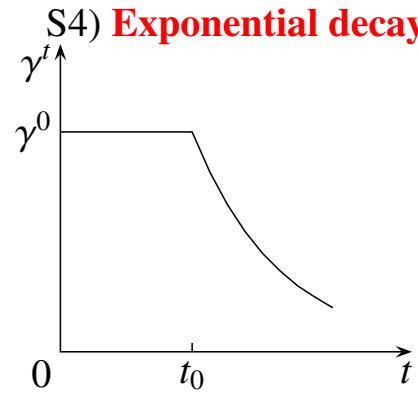
S2) Step decay



S3) Inverse time decay



S4) Exponential decay



$$\gamma^t = \begin{cases} \gamma^0 & t \leq t_0 \\ \frac{\gamma^0}{1+c(t-t_0)} & t > t_0 \end{cases}$$

$$\gamma^t = \begin{cases} \gamma^0 & t \leq t_0 \\ \gamma^0 e^{-c(t-t_0)} & t > t_0 \end{cases}$$

$t_0$  and the decay factor  $c > 0$  are two parameters to be suitably chosen.

## Adaptive schedules

- **RMSprop:** Root mean square propagation

The learning rate is adapted for each parameter of  $\theta$ . The idea is to divide the learning rate for a parameter by a running average of the magnitudes of recent gradients for that parameter.

- **Adam:** Adaptive moment estimation

It is an update to the RMSprop optimizer and uses running averages of both the gradients and the second moments of the gradients.

- **AdaGrad:** Adaptive gradient algorithm

The learning rate is adapted for each parameter of  $\theta$ . It increases the learning rate for more sparse parameters and decreases the learning rate for less sparse ones.

- **AdaDelta:**

It is a more robust extension of AdaGrad that adapts learning rates based on a moving window of gradient updates instead of accumulating all past gradients. This way, AdaDelta continues learning even when many updates have been done.

- ...

## Covariate shift

Different elements (channels) of the input  $\underline{x} = [x_i] = \underline{x}_0 \in \mathbb{R}^{M_0}$  of a DNN may have different means and variances, see a). Moreover, the distribution of the input samples may change from time to time, e.g. from training set to test set, see b). This phenomenon is called **covariate shift** in deep learning.

### E5.2: Covariate shift

#### a) Heterogenous input channels

$x_1$	acoustic measurement of a microphone
$x_2$	force measurement of a force sensor
$x_3, x_4, x_5$	3D components of an acceleration sensor
$x_6, x_7, x_8$	3D components of a gyroscope sensor (angular velocity)

Different sensors may have different offsets and gains. This may lead to ill conditioning (D2).

#### b) Cat recognition in images

- training set: black cats
- test set: white cats

### E5.3: A perceptron

Input  $\underline{x} \in \mathbb{R}^d$

Output  $f(\underline{w}) = \underline{w}^T \underline{x} \in \mathbb{R}$ , i.e. no hidden layers and a linear neuron.  
This is called Wiener filter in SASP.

Ground truth  $y \in \mathbb{R}$

Training set  $\underline{x}(n), y(n), 1 \leq n \leq N$

Cost function  $L(\underline{w}) = \frac{1}{N} \sum_{n=1}^N (y(n) - \underline{w}^T \underline{x}(n))^2 = \dots = \underline{w}^T \mathbf{R} \underline{w} - 2\underline{c}^T \underline{w} + \sigma_y^2$ ,

$\mathbf{R} = \frac{1}{N} \sum_{n=1}^N \underline{x}(n) \underline{x}^T(n)$  correlation matrix of  $\underline{x}(n)$

$\underline{c} = \frac{1}{N} \sum_{n=1}^N \underline{x}(n) y(n)$  cross-correlation vector between  $\underline{x}(n)$  and  $y(n)$

$\sigma_y^2 = \frac{1}{N} \sum_{n=1}^N y^2(n)$  power of  $y(n)$

The contour lines  $\{\underline{w}|L(\underline{w}) = \text{const}\}$  are ellipses in this case. Their shape and orientation are determined by the Hessian matrix  $\mathbf{H} = \nabla \nabla^T L = 2\mathbf{R}$ .

## Batch normalization (1)

The **batch normalization (BN)** of the activation  $\underline{a}_l(n)$  at layer  $l$  for one minibatch  $1 \leq n \leq B$  is defined as

$$\left[ \underline{a}_l(n) \right]_i \leftarrow \gamma_{l,i} \frac{\left[ \underline{a}_l(n) \right]_i - \mu_{l,i}}{\sqrt{\sigma_{l,i}^2 + \epsilon}} + \beta_{l,i}, \quad 1 \leq n \leq B, \quad 1 \leq i \leq M_l.$$

It consists of two steps for each element  $\left[ \underline{a}_l(n) \right]_i$  of  $\underline{a}_l(n)$ :

- A zero-mean unit-variance normalization like input normalization.

$$\mu_{l,i} = \frac{1}{B} \sum_{n=1}^B \left[ \underline{a}_l(n) \right]_i \quad \text{and} \quad \sigma_{l,i}^2 = \frac{1}{B-1} \sum_{n=1}^B \left( \left[ \underline{a}_l(n) \right]_i - \mu_{l,i} \right)^2$$

are the sample mean and variance of  $\left[ \underline{a}_l(n) \right]_i$  of this minibatch.  $\epsilon$  is a small positive number (e.g.  $10^{-5}$ ) to avoid division-by-zero. In contrast to the complete data set,  $\sigma_{l,i}^2 \approx 0$  is likely for a short minibatch.

- Scale-and-offset  $\gamma_{l,i} \square + \beta_{l,i}$  with two learnable parameters  $\gamma_{l,i}$  and  $\beta_{l,i}$  per neuron

## Batch normalization (2)

Why the second step  $\gamma_{l,i} \square + \beta_{l,i}$ ?

- Allow a flexible data dynamic range for each neuron and does not change the expressiveness of the network.
  - If  $\underline{\gamma}_l = \underline{\sigma}_l$  and  $\underline{\beta}_l = \underline{\mu}_l$ , batch normalization simplifies to the special case of no batch normalization. Normally,  $\underline{\gamma}_l \neq \underline{\sigma}_l$  and  $\underline{\beta}_l \neq \underline{\mu}_l$ .
  - $\underline{b}_l$  is redundant due to  $\underline{\beta}_l$  and can be omitted, i.e.  $\underline{a}_l(n) = \mathbf{W}_l \underline{x}_{l-1}(n)$ .
  - $\underline{\gamma}_l$  and  $\underline{\beta}_l$  are learned from data like  $\mathbf{W}_l$ . Hence each neuron adjusts the individually optimum dynamic range of  $[\underline{a}_l(n)]_i$  for the next nonlinear activation function  $\phi_l(\cdot)$ .
- Decouple the layers.
  - Batch normalization makes the dynamic range of one layer (partially) independent of the previous layers. This decouples the joint training of different layers to individual layers. It simplifies the learning, in particular for deep networks.
  - Batch normalization makes the surface of the cost function smoother.

## Alternatives to batch normalization★

Batch normalization normalizes the activation signals  $\underline{a}_l(n)$ . Some alternatives are:

- **Layer normalization:** Like batch normalization, but the mean and variance are not estimated across a minibatch  $1 \leq n \leq B$  for individual elements of  $\underline{a}_l$ , rather in an exchanged way across all elements of  $\underline{a}_l$  for individual samples  $n$  of a minibatch.
- **Weight normalization:** Normalize the weight tensors  $\mathbf{W}_l$ .
- For training GANs (ch. 9), the **spectral normalization** turns out to be effective. In this case, each weight tensor  $\mathbf{W}_l$  is normalized by its largest singular value  $\sigma(\mathbf{W}_l)$  in order to limit the so called Lipschitz constant of the discriminator. In practice, the largest singular value  $\sigma(\mathbf{W}_l)$  is approximated by the so called power iteration method.
- ...

## Parameter initialization

1) **Zero initialization**, e.g.  $\underline{\theta}^0 = \underline{0}$ .

Immediately after the zero initialization,

$$\begin{aligned}\underline{a}_l &= \mathbf{W}_l \underline{x}_{l-1} + \underline{b}_l = \underline{0}, \\ \underline{x}_l &= \phi_l(\underline{a}_l) = \phi_l(\underline{0})\end{aligned}$$

holds for all layers  $l$ . All neurons of a layer do the same calculation. The functionality of a layer is reduced to a single neuron. This is bad.

In general, a symmetry in  $\mathbf{W}_l$  and  $\underline{b}_l$  will lead to a symmetry of  $\underline{a}_l$  and  $\underline{x}_l$  and is not desired, because effectively the number of neurons is reduced. **Symmetry-breaking** is thus an important requirement for initialization.

In practice,

- all bias vectors  $\underline{b}_l$  are initialized with zero,
- all weight matrices  $\mathbf{W}_l$  are randomly initialized to break the symmetry.

## Improved model for fast learning

Improving the optimizer as in ch. 5.2–5.6 is not the only possibility to improve the learning of a DNN. Another more effective way is to change the model (DNN architecture) and thus the cost function  $L(\theta)$  to simplify the optimization. The universal approximation theorem guarantees the existence of a solution for each DNN model, but the solution may be easier to find in some models than in others.

One major difficulty in training a DNN is the vanishing gradient problem (D6), i.e. the shallow layers tend to have zero gradients. Hence a recent trend is to design a network with

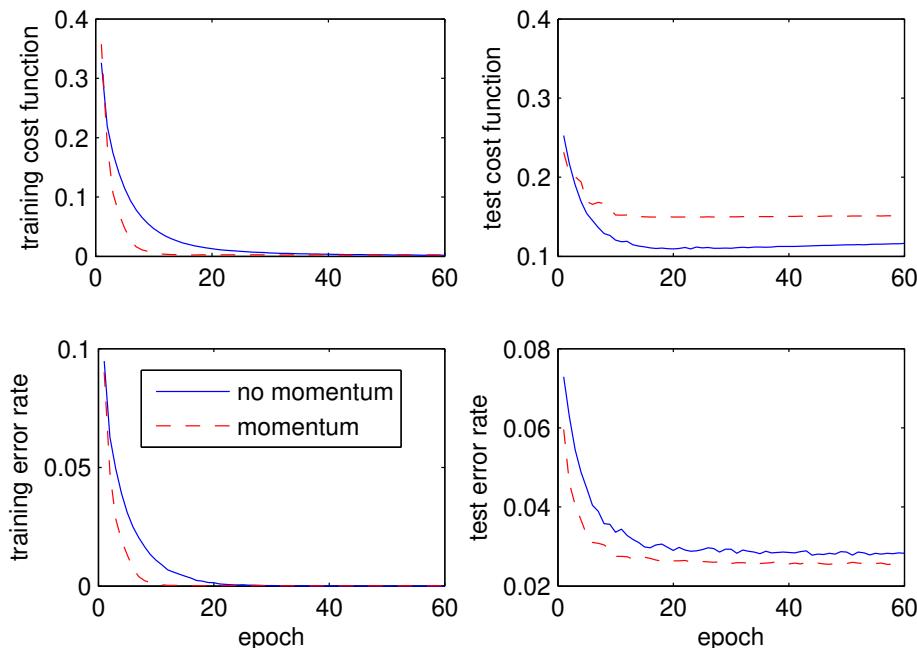
- a constant activation flow during the whole training period,
- a constant gradient flow during the whole training period,
- activation functions with significant slopes mostly.

*For easier hiking, you can buy good hiking shoes, but you can also select an easier hiking landscape.*

## E5.4: MNISTnet1 – Advanced optimizations (1)

We use the same baseline network, cost function and optimizer settings as in E4.5. Below we vary each time one optimizer setting and study its impact.

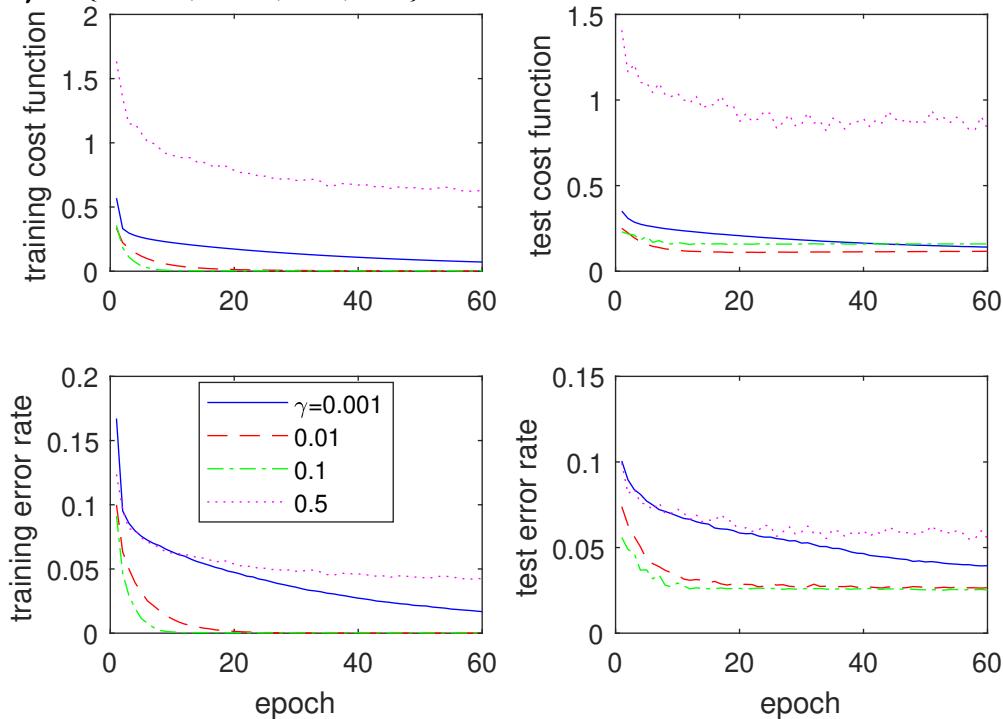
### a) Nesterov momentum with $\beta = 0.9$



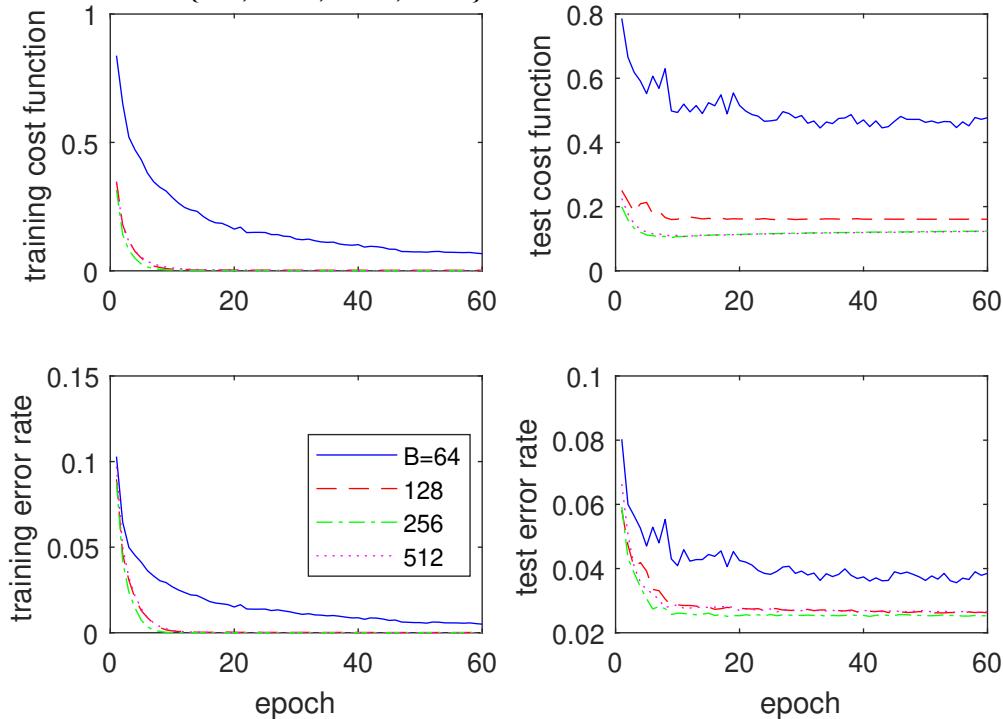
- Momentum accelerates the convergence.

**E5.4: MNISTnet1 – Advanced optimizations (2)**

b) Step size  $\gamma \in \{0.001, 0.01, 0.1, 0.5\}$



- A too small step size leads to a slow convergence.
- A too large step size leads to a divergence.

**E5.4: MNISTnet1 – Advanced optimizations (3)****c) Minibatch size  $B \in \{64, 128, 256, 512\}$** 

- A too large minibatch size could not fit into the RAM of CPU/GPU.
- A too small minibatch size leads to a noisy gradient vector and a slow convergence.

## E5.4: MNISTnet1 – Advanced optimizations (4)

### Observations

- A good choice of optimizer settings can accelerate the convergence of training.
- It does not, however, help to prevent overfitting. In all experiments, the best test error rate is roughly 2.5% while the corresponding training error rate is almost zero.
- The problem of overfitting is not solved yet.

## Model capacity, underfitting and overfitting

The goal of machine learning is to learn a **model**  $f(\underline{x}; \theta)$  with a low **test (generalization) error**, i.e. the model performs well for new unseen data  $\underline{x}$ , see ch. 1.1.

The **capacity** of a model is its ability to learn a mapping  $\underline{x} \rightarrow \underline{y}$  from training data. One can change the model capacity by changing the model function  $f(\underline{x}; \theta)$ , i.e. the DNN architecture. A simple function  $f$  means a low model capacity and a complicated function  $f$  with a large number of parameters implies a high model capacity.

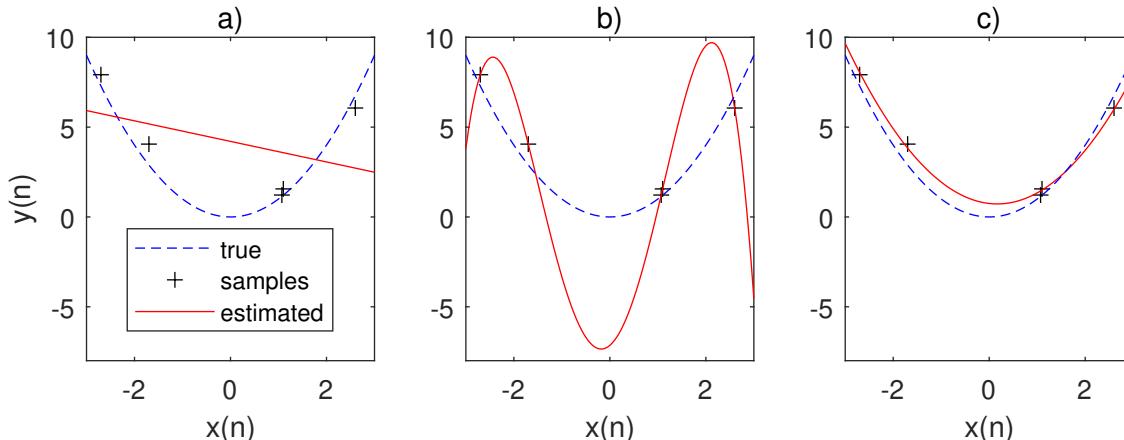
Models with low capacity fail to learn the mapping. This is called **underfitting**. The model is not able to obtain a sufficiently low error even on the training set.

Models with high capacity can learn the mapping with a low training error, but tend to *memorize* the training data without learning the underlying structure of the mapping  $\underline{x} \rightarrow \underline{y}$ . They perform poorly for new unseen data. The test error is much larger than the training error. This is called **overfitting** and happens frequently when the dataset is small and the model capacity is large.

### E6.1: Underfitting and overfitting in univariate regression by a polynomial

Training set:  $x(n), y(n) = x(n)^2 + N(0, 1)$ ,  $1 \leq n \leq N = 5$

- a) a too simple model:  $f(x) = a_1x + a_0 \rightarrow$  underfitting
- b) a too complicated model:  $f(x) = \sum_{i=0}^d a_i x^i$ ,  $d = 4 \rightarrow$  overfitting
- c) a good model:  $f(x) = a_2 x^2 + a_1 x_1 + a_0$



$$\{a_0, \dots, a_d\} = \{4.21, -0.57\}$$

$$\{-7.15, 2.29, 6.31, -0.41, -0.62\}$$

$$\{0.75, -0.30, 0.89\}$$

Overfitting is visible on large polynomial coefficients. They lead to an erratic curve.

## How to find the right model?

For multivariate regression and classification,

- perceptron  $\underline{w}^T \underline{x} + b$ : a (too) simple model
- shallow NN: a quite powerful model due to the universal approximation
- deep NN: even more powerful
  - + can solve more practical problems than shallow NN
  - tend more to overfitting

Challenge: Find the right model.

- complex enough to solve a given problem
- simple enough to avoid overfitting

But: No theory to predict the right model for a given problem!

Solution:

- use a powerful enough model (large DNN)
- and regularization

## Regularization

In machine learning, all techniques to prevent overfitting are known collectively as **regularization**.

A model with a higher capacity than necessary for a given task has a large solution space, i.e. many different solutions with a comparable training error. Some of the solutions are overfitted to the training data while others not. Regularization reduces the solution space and prefers these solutions with a good generalization.

There are different methods for regularization. Some of them are applicable to general optimization (OPT) problems, some others for ML, and some of them for DNN only.

Ch.	Methods	Change on	OPT	ML	DNN
6.2	weight norm penalty	cost function	×	×	×
6.3	early stopping	optimizer		×	×
6.4	data augmentation	dataset		×	×
6.5	ensemble learning	dataset/model/cost function/optimizer	×	×	×
6.6	dropout	model			×

## Weight norm penalty

Why does this work?

- A general observation in ML is that a model with large weights tends to overfitting. Large weights mean large changes in output for small changes in input. Thus, a model with large weights might fit better to observed samples (training data), but will behave erratically between these observed samples, see E6.1.
- Most practical problems have a smooth input-output relationship. Hence models with small weights are preferred because they behave smoothly between observed samples, not just right at them.

How to choose the regularization parameters?

- The regularization parameters  $\lambda_l$  have to be chosen carefully. Too small values will lead to no effective regularization. Too large values will seriously change the solution of the original problem  $\min L(\underline{\theta})$ , leading to underfitting.
- See hyperparameter optimization in ch. 6.7.

## Early stopping

If a model has a larger capacity than necessary, it tends to overfitting. In this case,

- the training error decreases continuously with the number of epochs. The longer the training, the smaller the training error.
- the test error, however, decreases first and then increases due to overfitting.

**Early stopping** stops the training before the test error increases.

- For this purpose, the learned model needs to be applied to the test set after each epoch in order to calculate the test error<sup>1</sup>.
- This is a simple regularization method without any changes on dataset, model, cost function and without tuning of any hyperparameters (like  $\lambda_l$  in ch. 6.2).

---

<sup>1</sup>To be more precise, the test set and test error for this purpose are called validation set and validation error, see ch. 6.7.

## Data augmentation

**Data augmentation** is a technique to generate synthetic but realistic training samples to extend the training set, to cover a larger part of the true data distribution and to prevent overfitting.

- Data augmentation is quite easy for classification by slightly modifying the true input samples without changing their class labels.
- e.g. image recognition:
  - translation/rotation/scaling/flip of true input images
  - add noise to images
  - modify colors, textures, . . .

However, it strongly depends on the application which modifications are allowed. In character recognition, for example, horizontal flip is not allowed as data augmentation due to confusion between "b" and "d".

- Data augmentation is more difficult for regression due to continuous-valued output.

## Ensemble learning

**Ensemble learning** is a model averaging method to reduce the test error by combining an ensemble of models:

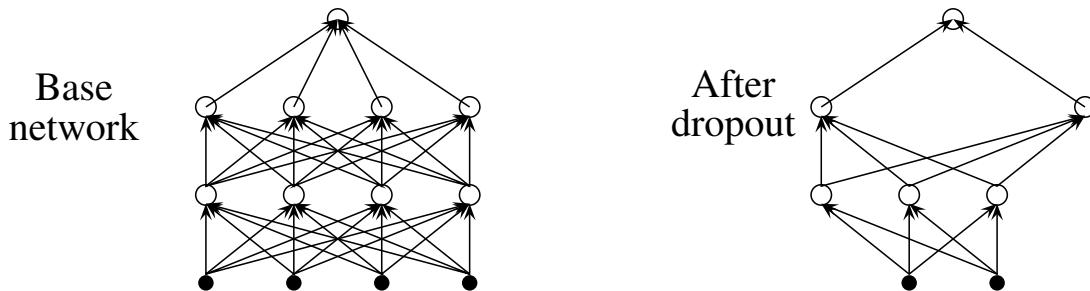
- train different (independent) models for the same task
- combine these models to reduce the test error
  - regression: average of the model outputs
  - classification: voting of the model outputs, e.g. 3× cat and 1× dog

It is unlikely that all models will make the same errors on the test set. Hence the averaged model is more robust.

The different models can be trained by using

- different subsets of the training set or
- different model architectures or
- different cost functions or
- different optimizers or
- combinations of them.

## Dropout (1)



### Training

- For each minibatch, **dropout** randomly removes some neurons in layer  $l$  of a base network with a probability  $d_l$ , the **dropout rate**.
- This results for each minibatch in a random subnetwork for solving the same task.<sup>2</sup>

### Inference

- No dropout.
- All outgoing weights of neurons in layer  $l$  are weighted by  $1 - d_l$  to correct the large fan-in to the neurons (number of inputs) of the next layer.

<sup>2</sup>N. Srivastava and G. Hinton et al, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research, 2014

## Dropout (2)

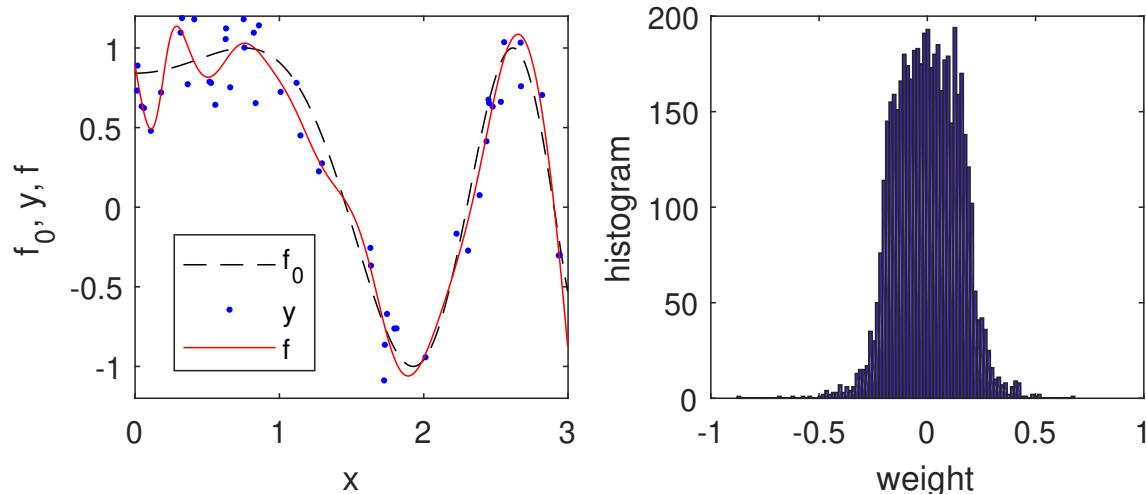
Why does dropout work?

- When we train a single network with a large capacity, it often happens that the weights of some neurons are sensitive to the weights of other neurons. They are co-adapted. By dropout, the weights of some neurons become more independent to other weights. This makes the model more robust. If a hidden neuron has to work well in different combinations with other hidden neurons, it's more likely that this hidden neuron does something individually useful.
- The final DNN for inference can be viewed as an average of a huge number of thinned networks. Each thinned network gets poorly trained (underfitting) due to a smaller number of inputs and a smaller number of hidden neurons. It will never be used alone. But the ensemble averaging of these weak models results in a strong model. This is the basic idea of ensemble learning.
- Dropout is simple to implement and does not boost the computational complexity. Yet it is effective to prevent overfitting.

## E6.2: Overfitting and regularization in regression by a DNN (1)

- true function  $f_0(x) = \sin(1 + x^2)$
- $N = 50$  noisy samples  $y(n) = f_0(x(n)) + N(0, 0.2^2)$  for  $0 \leq x(n) \leq 3$
- 2 hidden layers ( $M_1 = 100, M_2 = 50$ ) as in E4.4
- network output  $f(x)$  as approximation for  $f_0(x)$

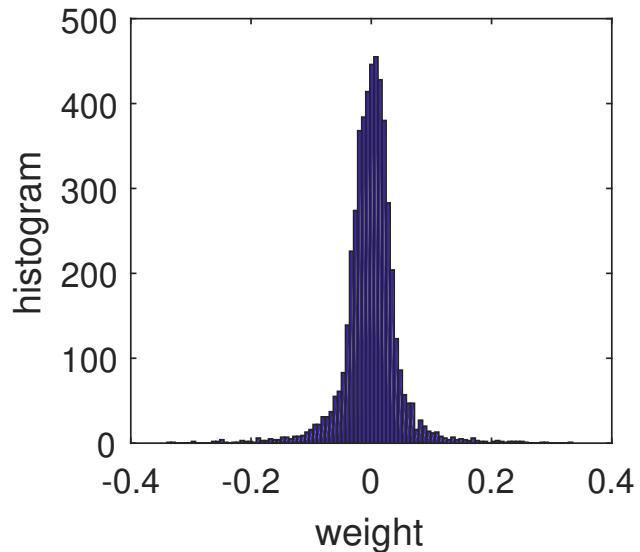
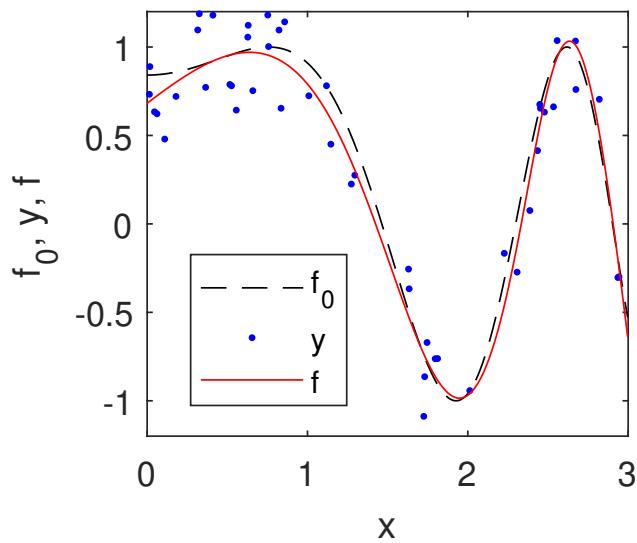
a) No regularization



Network learns a too complicated function  $f(x; \underline{\theta})$ . It is overfitted to training data.

**E6.2: Overfitting and regularization in regression by a DNN (2)**

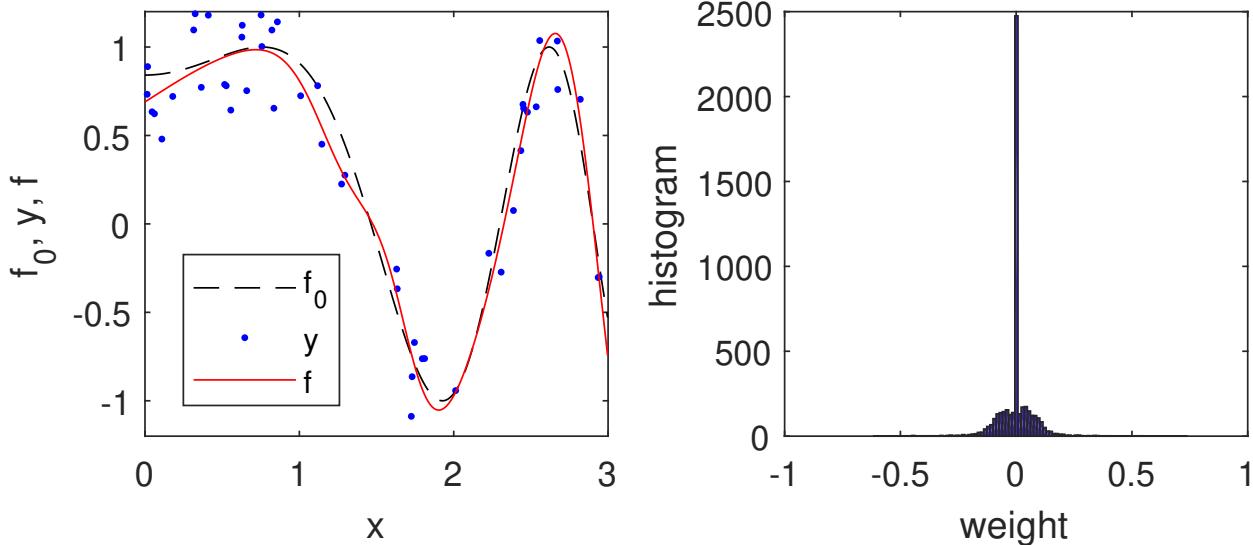
b)  $l_2$ -regularization with  $\lambda = 0.001$



Weights are smaller (weight decay) due to  $l_2$ -regularization and the function  $f(x; \theta)$  is more smooth.

## E6.2: Overfitting and regularization in regression by a DNN (3)

c)  $l_1$ -regularization with  $\lambda = 0.0001$



The weight matrix  $\mathbf{W}_2 \in \mathbb{R}^{50 \times 100}$  has 2316 elements (46%) with  $|w_{2,i,j}| \leq 10^{-5}$  due to the  $l_1$ -regularization. It is approximately a sparse matrix. In the previous  $l_2$ -regularization, only 3 elements of  $\mathbf{W}_2$  satisfy this condition.

### E6.3: MNISTnet2 (1)

The second fully connected neural network for MNIST.

#### Architecture

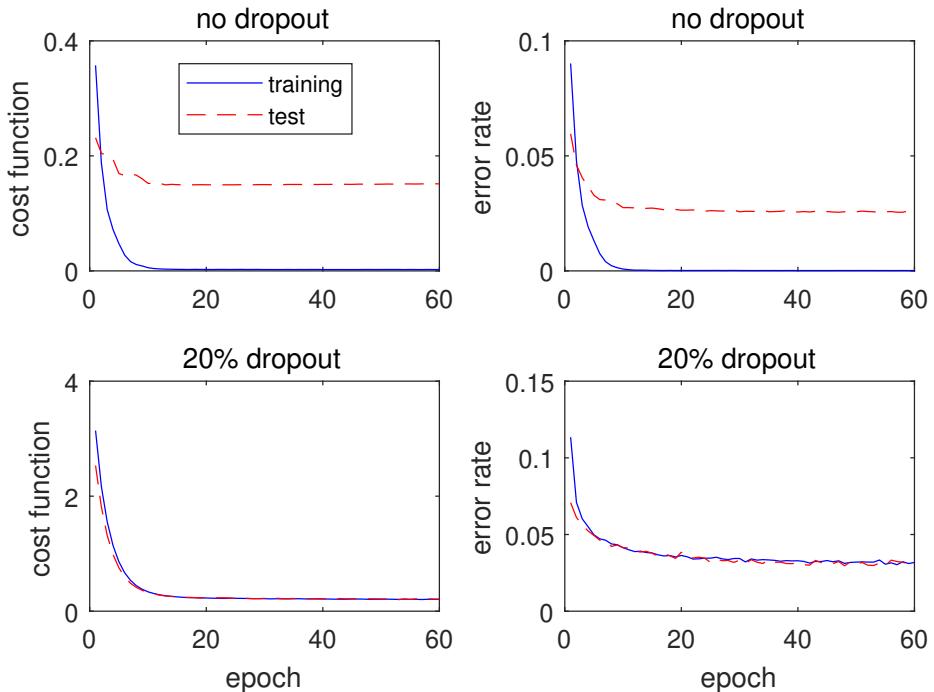
- $784 \times 512 \times 512 \times 512 \times 10$ , i.e. 3 hidden layers with each 512 neurons instead of one hidden layer with 1024 neurons
- $N_p = 932,362$  parameters and  $N_x = 930,816$  multiplications

#### Optimizer

- minibatch size  $B = 128$ , 60 epochs and fixed step size  $\gamma^t = 0.01$  as in MNISTnet1, but with
- momentum  $\beta = 0.9$  and Nesterov momentum
- $l_2$ -regularization with  $\lambda = 0.002$
- a) no dropout
- b) dropout rate 0.2 in all hidden layers, i.e. 20% of neurons are randomly set to zero

### E6.3: MNISTnet2 (2)

## Results



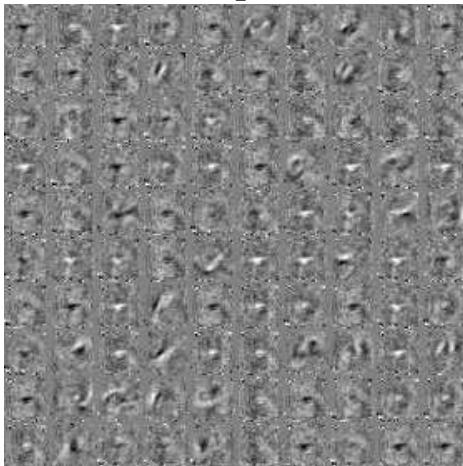
- dropout is very effective to reduce overfitting
- test error rate 3%, not really better than MNISTnet1

### E6.3: MNISTnet2 (2)

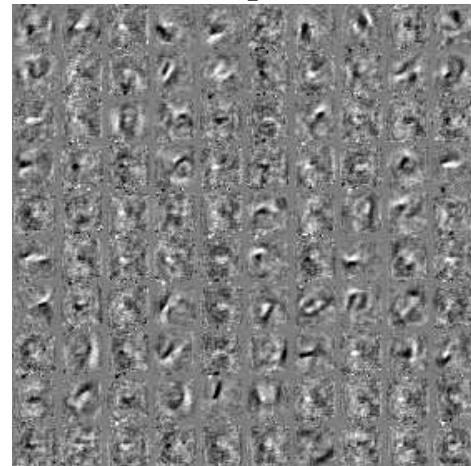
Visualization of  $\mathbf{W}_1 \in \mathbb{R}^{512 \times 784}$

- As in E4.5, the weight vectors of the first 100 neurons in the first hidden layer are normalized and reshaped to a 10x10 grid containing each a 28x28 weight matrix.

no dropout



dropout



- Each of the neurons seems to match to a digit-like input pattern.
- $\mathbf{W}_1$  with dropout shows a stronger pattern than that without dropout and from E4.5, i.e. dropout forces each neuron to learn something useful individually.

## Hyperparameters

What are they?

- In contrast to the model parameters  $\underline{\theta}$  (weights and biases) to be learned from training data, **hyperparameters**  $\underline{\eta}$  are configuration parameters of a machine learning algorithm which are not adapted during training.
- $\underline{\eta}$  is chosen before learning and remains fixed. It controls, together with  $\underline{\theta}$ , the behavior of the model  $f(\underline{x}; \underline{\theta}, \underline{\eta})$ , see next slide. Hence, they need an optimization.

Why are they not adapted?

- a) Hyperparameters are often discrete valued (e.g. number of layers/neurons, type of activation function). Gradient descent is not suitable for this kind of integer optimization.
- b) The training error is a monoton function of some hyperparameters, in particular those controlling the model capacity. An optimization of these hyperparameters would always maximize the model capacity (e.g. more layers, more neurons) resulting in overfitting. The training set alone is not suitable for hyperparameter optimization.

## Hyperparameters of a DNN

<b>Model</b>	
type of neural network	dense network/CNN/RNN/...
number of layers	$L$
number of neurons	$M_l, 1 \leq l \leq L$
type of activation function	$\phi_l(\cdot), 1 \leq l \leq L$
dropout rate	$d_l, 1 \leq l \leq L$
<b>Cost function</b>	
type of regularization	$l_2$ or $l_1$
regularization parameters	$\lambda_l, 1 \leq l \leq L$
<b>Optimizer</b>	
minibatch size	$B$
number of epochs	$N_{\text{epoch}}$
learning schedule and step size	$\gamma^t$
momentum factor	$\beta$
Nesterov momentum	yes/no
initial value of $\underline{\theta}$	$\underline{\theta}^0$

## Training set, validation set and test set

**Training set**  $D_{\text{train}}$ : It is used for training the model, i.e. learning the model parameters  $\underline{\theta}$  (weights and biases) for a fixed hyperparameter vector  $\underline{\eta}$ .

**Validation set**  $D_{\text{val}}$ : It is never used in training. It is reserved for optimizing the hyperparameter parameters  $\underline{\eta}$ .

**Test set**  $D_{\text{test}}$ : It is never used in training and hyperparameter optimization. It is used to calculate the test error of the trained ( $\underline{\theta}$ ) and tuned ( $\underline{\eta}$ ) model  $f(\underline{x}; \underline{\theta}, \underline{\eta})$  to exam its generalization capability. The motivations of using the test set are

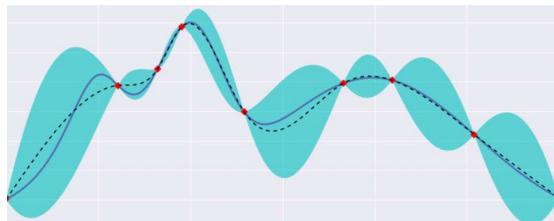
- to avoid overfitting of  $\underline{\theta}$  to the training set and
- to avoid overfitting of  $\underline{\eta}$  to the validation set.

### E6.4: Early stopping

Early stopping from ch. 6.3 is a hyperparameter optimization to determine the optimum value of the hyperparameter, the number of epochs. This is only possible by using a validation set because the training error often decreases continuously as the number of epochs increases.

## Hyperparameter optimization approaches

- **Grid search** A blind exhaustive search on a grid in the hyperparameter space, e.g.  $M_1 \in \{100, 150, 200\}$ ,  $\phi_l \in \{\text{sigmoid}, \text{ReLU}\}, \dots$ 
  - + simple,
  - time-consuming, especially for many hyperparameters and a fine grid
- **Bayesian optimization** Treat hyperparameter tuning as an optimization problem
  - probabilistic model for the posterior of the cost function based on Bayes rule (and Gaussian distribution)
  - samples of hyperparameters and cost function: iteratively refine the model
  - posterior: which regions of the hyperparameter space are worth exploring
  - choose next hyperparameter value based on posterior



— target function, - - - predicted function,  
● samples, ■ confidence interval

<https://github.com/fmfn/BayesianOptimization>:  
A Python package ready for use

## Drawbacks of dense networks

- Huge number of parameters in  $\theta$ . In particular, the number of parameters of a dense layer  $\mathbf{W}_l \in \mathbb{R}^{M_l \times M_{l-1}}$  increases quadratically with the number of neurons for  $M_l = M_{l-1}$ . The result is
  - a tremendous computational and memory complexity and
  - a higher overfitting risk due to a large model capacity.
- Not suitable to learn local patterns/features of input signal due to the full connectivity of neurons. Denser layers are good for classification/regression, but not for hierarchical feature learning.

### E7.1: Huge number of parameters of a dense network

a) tiny MNISTnet1 in E4.5:

$$M_0 = 28^2 = 784, M_1 = 1024, M_2 = 10 \rightarrow N_p = 814.090 \text{ parameters}$$

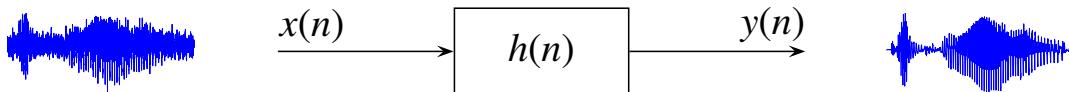
b) ImageNet, reduced image size 224x224:

$$M_0 = 224^2 = 50176, M_1 = 1024 \rightarrow \text{the first layer has } M_0(M_1 + 1) = 51.430.400 \text{ parameters!}$$

c) Full HD image 1920x1080:  $M_0 = 1920 \cdot 1080 = 2.073.600, \dots$

## 1D convolution

Bachelor course "Signals and systems" and "Digital signal processing":



- For a 1D **digital filter** with the input signal  $x(n)$  and **impulse response**  $h(n), n \in \mathbb{Z}$ , the output signal  $y(n)$  is given by the **convolution**

$$y(n) = \sum_m h(m)x(n-m).$$

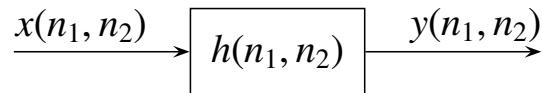
- A similar operation is the **correlation** between  $x(n)$  and  $h(n)$

$$\sum_m h(m)x(n+m) \stackrel{k=-m}{=} \sum_k h(-k)x(n-k).$$

Clearly, convolution is equivalent to correlation of  $x(n)$  with  $h(-n)$ .

- Such a filter is **time-invariant**, namely a shifted input signal results in the same shifted output signal:  $x(n - n_0) \rightarrow y(n - n_0)$ .
- The impulse response  $h(n)$  can have a finite or infinite length. In the former case, the filter is called a **FIR filter**, see page 5-11.

## 2D convolution



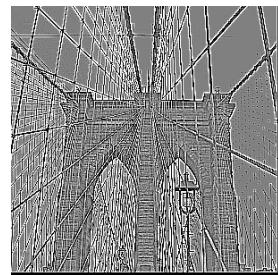
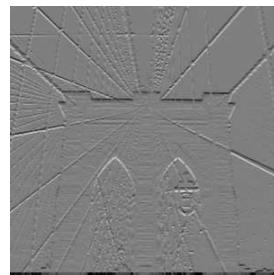
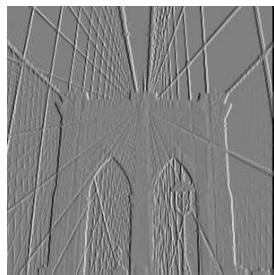
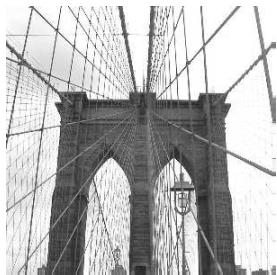
- Similarly, for a 2D digital filter with the input image  $x(n_1, n_2)$  and impulse response (**point spread function** in image processing)  $h(n_1, n_2)$ , the output image  $y(n_1, n_2)$  is given by the 2D convolution

$$y(n_1, n_2) = \sum_{m_1} \sum_{m_2} h(m_1, m_2) x(n_1 - m_1, n_2 - m_2).$$

- It is again equivalent to the correlation of  $x(n_1, n_2)$  with the flipped impulse response  $h(-n_1, -n_2)$ .
- The property  $x(n_1 - n_{10}, n_2 - n_{20}) \rightarrow y(n_1 - n_{10}, n_2 - n_{20})$  is now called **shift-invariance**.
- Again, the impulse response  $h(n_1, n_2)$  can have a finite or infinite large size. In image processing,  $h(n_1, n_2)$  has typically a small size, e.g.  $3 \times 3$  or  $5 \times 5$ .

## E7.2: 2D convolution for feature extraction

Different 2D filters can be used to extract different features (vertical edges, horizontal edges, edges) from an image.



$$h(n_1, n_2) =$$

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Sobel

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

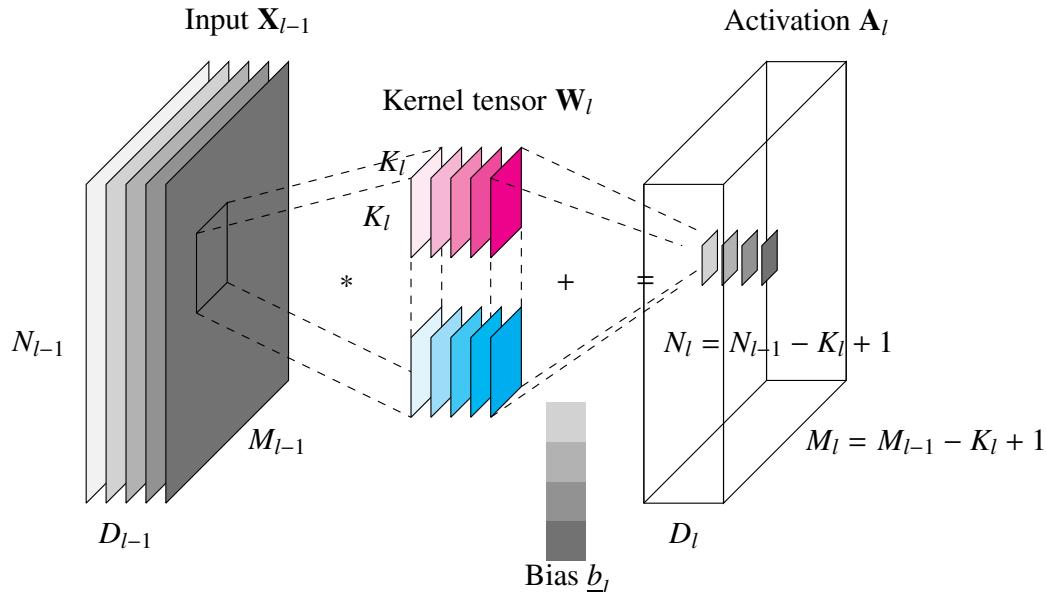
Sobel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Laplacian

## 2D convolutional layer $l$

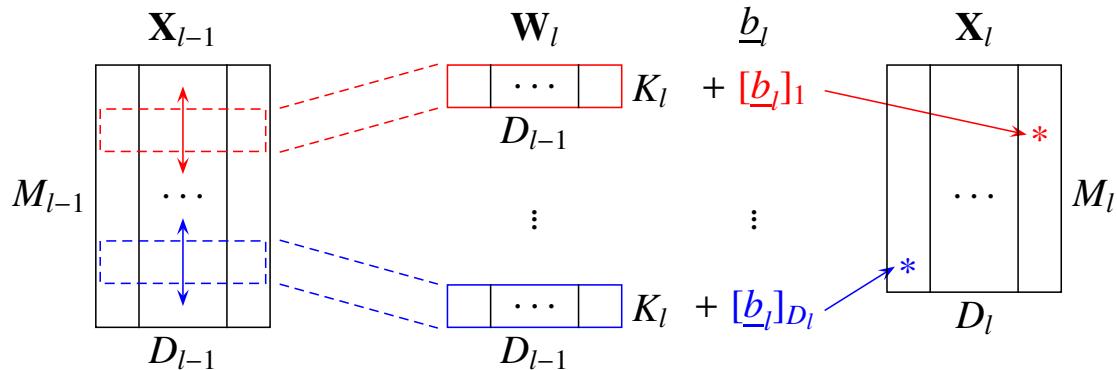
- The input  $\mathbf{X}_{l-1}$  contains  $D_{l-1}$  2D feature maps (or channels) of size  $M_{l-1} \times N_{l-1}$ .
- The layer contains  $D_l$  kernels to calculate  $D_l$  2D feature maps of size  $M_l \times N_l$ .
- Each kernel contains  $D_{l-1}$  2D filters of size  $K_l \times K_l$ , each performing a **sliding window** convolution or correlation.



# 1D convolutional layer

- input matrix  $\mathbf{X}_{l-1} \in \mathbb{R}^{M_{l-1} \times D_{l-1}}$ :  $D_{l-1}$  input vectors of length  $M_{l-1}$
- output matrix  $\mathbf{X}_l = \phi_l(\mathbf{A}_l) \in \mathbb{R}^{M_l \times D_l}$ :  $D_l$  output vectors of length  $M_l$
- kernel tensor  $\mathbf{W}_l \in \mathbb{R}^{K_l \times D_{l-1} \times D_l}$ :  $D_l$  kernels of size  $K_l \times D_{l-1}$ , kernel width  $K_l$
- bias vector  $\underline{b}_l \in \mathbb{R}^{D_l}$ : one bias value for one output vector
- activation matrix  $\mathbf{A}_l \in \mathbb{R}^{M_l \times D_l}$ :

$$[\mathbf{A}_l]_{mo} = \sum_{i=1}^{K_l} \sum_{d=1}^{D_{l-1}} [\mathbf{W}_l]_{ido} [\mathbf{X}_{l-1}]_{m+i-1,d} + [\underline{b}_l]_o, \quad 1 \leq m \leq M_l = M_{l-1} - K_l + 1, 1 \leq o \leq D_l.$$



## 3D convolutional layer

In some applications like computer tomography (CT) and magnetic resonance tomography (MRT), the input data is a 3D volume image. In this case, often a 3D CNN is required with 5D kernel tensors  $\mathbf{W}_l \in \mathbb{R}^{K_l \times K_l \times K_l \times M_{l-1} \times M_l}$ . The typical kernel size is  $3 \times 3 \times 3$  or  $5 \times 5 \times 5$ .

### E7.3: Convolution as a matrix multiplication

The convolution operation in a convolutional layer can be rewritten as a matrix multiplication as in a dense layer. For simplicity, we consider a 1D convolutional layer with the input vector  $[x_1, \dots, x_M]^T$ , the kernel  $[w_1, \dots, w_K]^T$ ,  $D_{l-1} = D_l = 1$  and zero bias. The activation is then

$$a_m = \sum_{i=1}^K w_i x_{m+i-1}, \quad 1 \leq m \leq M - K + 1$$

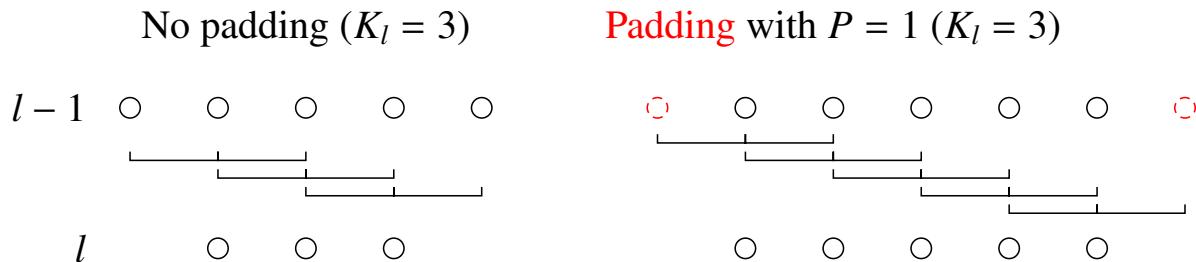
or in matrix notation

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{M-K+1} \end{bmatrix} = \begin{bmatrix} w_1 & \dots & w_K \\ w_1 & \dots & w_K \\ \ddots & & \ddots \\ w_1 & \dots & w_K \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}.$$

The corresponding  $(M - K + 1) \times M$  weight matrix is a Toeplitz band matrix with obvious **parameter sharing**. It contains only  $K$  parameters instead of  $(M - K + 1)M$  for a dense layer.

## Modifications to convolution: **Padding**

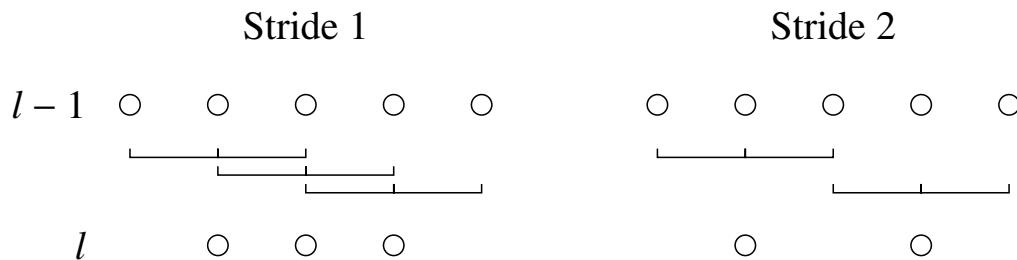
Zero-padding the input with  $P \in \mathbb{N}$  zeros at each side before convolution.



Effect: Larger output size. In particular, if  $P = \frac{K_l-1}{2}$ , the output has the same size  $M_l = M_{l-1} + 2P - K_l + 1 = M_{l-1}$  as the input. This property may be desired in some applications.

## Modifications to convolution: **Stride**

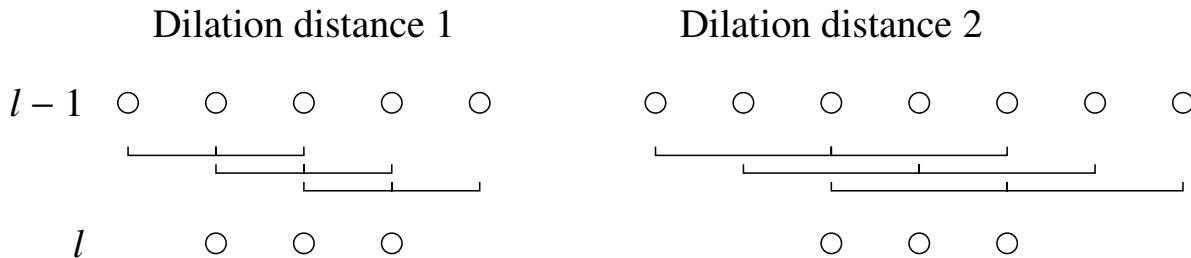
Move the kernel each time by  $S \in \mathbb{N}$  positions instead of one position.



Effect: **Downsampling** or **subsampling** or **decimation** of the output of a normal convolution with stride 1 by factor  $S$ , i.e. keep every  $S$ -th value of the output. This reduces further the output size.

## Modifications to convolution: Dilated convolution

Apply convolution to input samples with a **dilation distance**  $D \in \mathbb{N}$  instead of  $D = 1$ .



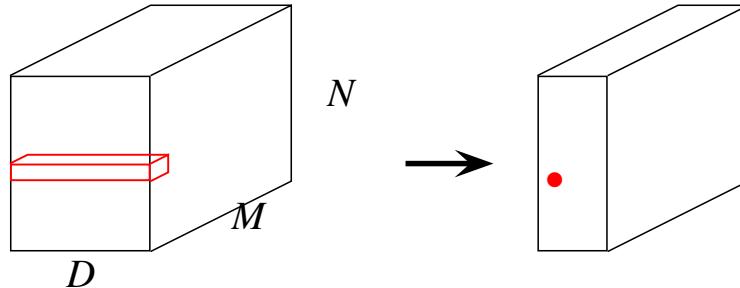
Effect: **Polyphase downsampling** of the input by factor  $D$ . This increases the receptive field without increasing the kernel width. The downsampling of the input may, however, lead to aliasing artifacts, see course "Digital signal processing".

All modifications padding, stride and dilated convolution can be used in any combination.

## **$1 \times 1$ convolution (1)**

**$1 \times 1$  convolution:** kernel size  $K = 1$

- no spatial processing
- combine input feature maps at every pixel (pointwise)

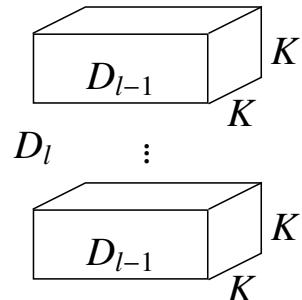


Why?

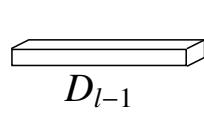
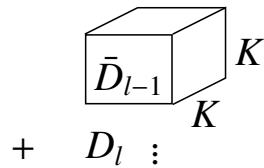
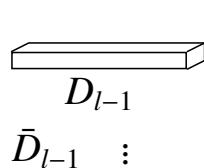
- reduce the channel depth  $D$  of the input to reduce the computational complexity at the next layer
- additional higher nonlinearity
- used e.g. in GoogLeNet (see ch. 10)

## Comparison of convolutions

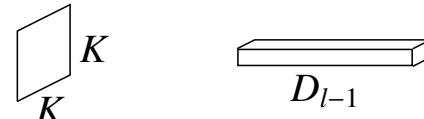
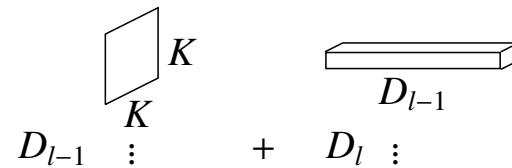
1) kernel of standard convolution



2) kernel of  $1 \times 1$  convolution



3) kernel of depthwise separable convolution



## Max pooling

Before max pooling

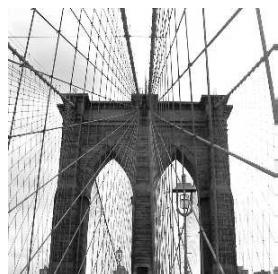
5	3	5	7
2	8	2	4
3	2	6	1
1	5	0	5

After  $2 \times 2$  max pooling

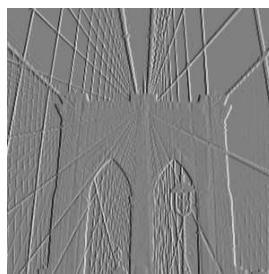
8	7
5	6

### E7.4: Max pooling

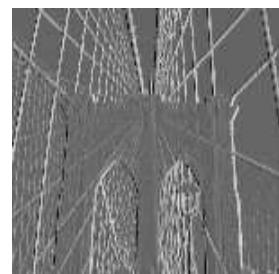
Image



Vertical edges



2x2 max pooling



4x4 max pooling



## Effects of pooling

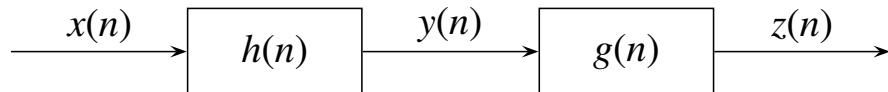
- Reduced spatial size of the feature maps  
→ reduced computational and memory complexity
- Abstraction of the relevant characteristics of the input
  - not the precise position, rather the structure of the input is important
  - useful for hierarchical representation in a deep CNN, see ch. 7.4
  - learn the underlying structure of the input, help to avoid overfitting
- One pooling layer is often **invariant** to a small (< stride) translation of the input.



- A stack of convolutional and pooling layers is **invariant** to a large translation of the input. This means, the classification output of a CNN is independent of the position of the object in the input image as desired. Note, translation-invariant is different from translation-equivariant.

## Deconvolution in signal processing

In signal processing, **deconvolution** is the process to reverse the effect of a previous convolution.



Given an input signal  $x(n)$ . A linear time-invariant (LTI) system with the impulse response  $h(n)$  and frequency response  $H(\omega)$  changes  $x(n)$  by a convolution

$$y(n) = (h * x)(n) = \sum_k h(k)x(n - k).$$

In the frequency domain,  $Y(\omega) = H(\omega)X(\omega)$ . A second LTI system in cascade with  $h(n)$  has the impulse response  $g(n)$  and frequency response  $G(\omega)$ . It computes  $z(n) = (g * y)(n)$  and  $Z(\omega) = G(\omega)Y(\omega) = G(\omega)H(\omega)X(\omega)$ .

If  $g(n)$  is designed to achieve  $z(n) = x(n)$ ,  $g(n)$  is called the deconvolution of  $h(n)$  and  $G(\omega) = 1/H(\omega)$  is the inverse of  $H(\omega)$ .

## Upsampling in signal processing

**Upsampling** in signal processing is an interpolation applied to a signal to enhance its sampling rate (resolution) without changing its waveform/shape. For an 1D signal  $x(n)$  and an integer **upsampling factor**  $S$ ,

- insert  $S - 1$  zeros between each pair of adjacent samples of  $x(n)$ :

$$\dots, x(n-1), \underbrace{0, \dots, 0}_{S-1}, x(n), \dots$$

- smooth out the discontinuities with a lowpass filter (convolution)

### E7.5: Upsampling in signal processing

original

$28 \times 28$

unpooling

$56 \times 56$

zero insertion

$56 \times 56$

+ lowpass

$56 \times 56$



## Different names in DNN and signal processing

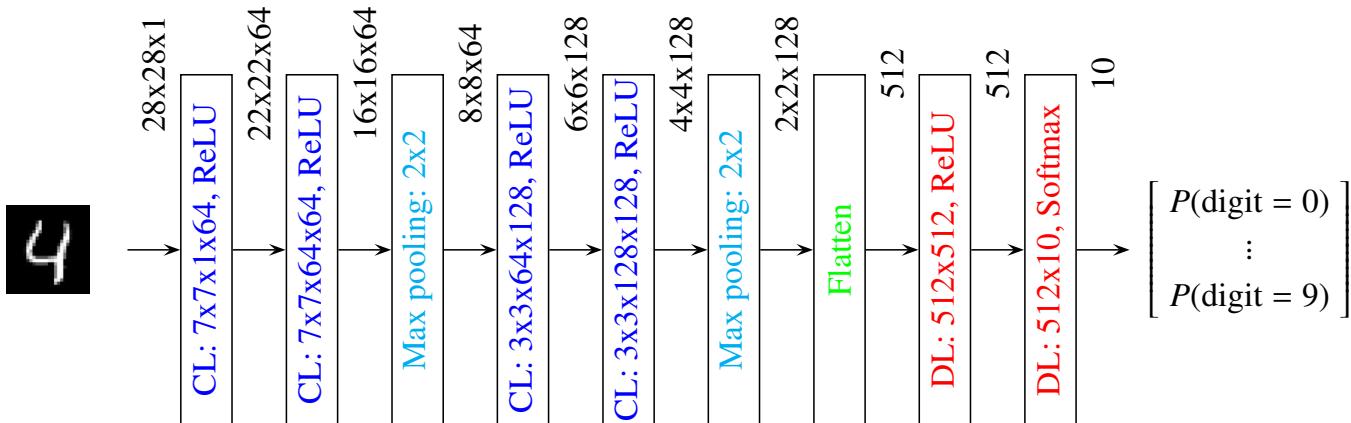
The DL community reinvented many new names for known concepts from signal processing, in particular for CNN. Unfortunately, some new names are *not* consistent to the old ones. This often causes confusion.

	in deep learning	in signal processing
$\mathbf{X}_{l-1}, \mathbf{X}_l$	feature maps	input/output signals
$\mathbf{W}_l$	kernel	impulse response
$\sum_i \sum_j w_{ij} x_{m+i,n+j}$	convolution	correlation
$x(n - n_0) \rightarrow y(n - n_0)$	translation-equivariant	shift-invariant
$x(n - n_0) \rightarrow y(n)$	translation-invariant	—
	padding	zero initialization
	stride	downsampling of output
	dilated convolution	polyphase downsampling of input
reverse convolution	—	deconvolution
	deconvolutional layer	learnable upsampling

## E7.6: MNISTnet3 (1)

The third neural network, a CNN, for digit recognition on MNIST.

### Architecture



- 4 convolutional layers (CL)
- 1 flatten layer
- 2 max pooling layers
- 2 dense layers (DL)

From shallow to deep layers, the number of feature maps  $D_l$  is often increased in parallel to the reduction of their spatial size  $M_l \times N_l$  in order to avoid too much information loss.

## E7.6: MNISTnet3 (2)

### Complexity

- Convolutional layer with kernel tensor  $\mathbf{W}_l \in \mathbb{R}^{K_l \times K_l \times D_{l-1} \times D_l}$  and output tensor  $\mathbf{X}_l \in \mathbb{R}^{M_l \times N_l \times D_l}$ :  $N_p = K_l^2 D_{l-1} D_l + D_l$ ,  $N_x = M_l N_l K_l^2 D_{l-1} D_l$
- Dense layer with weight matrix  $\mathbf{W}_l \in \mathbb{R}^{M_l \times M_{l-1}}$  and output vector  $\underline{x}_l \in \mathbb{R}^{M_l}$ :  $N_p = M_l(M_{l-1} + 1)$ ,  $N_x = M_l M_{l-1}$

	Weight tensor	#parameters $N_p$	Output tensor	#multiplications $N_x$
CL 1	7x7x 1x 64	3,200	22x22x64	1,517,824
CL 2	7x7x 64x 64	200,768	16x16x64	51,380,224
CL 3	3x3x 64x128	73,856	6x6x128	2,654,208
CL 4	3x3x128x128	147,584	4x4x128	2,359,296
DL 1	512x512	262,656	512	262,144
DL 2	512x 10	5,130	10	5,120
MNISTnet3		693,194		58,178,816
MNISTnet1	E4.5	814,090		813,056
MNISTnet2	E6.3	932,362		930,816

## E7.6: MNISTnet3 (3)

### Model

- dropout rate 0.25 for 2 convolutional layers before max pooling
- dropout rate 0.5 for the first dense layer

### Cost function

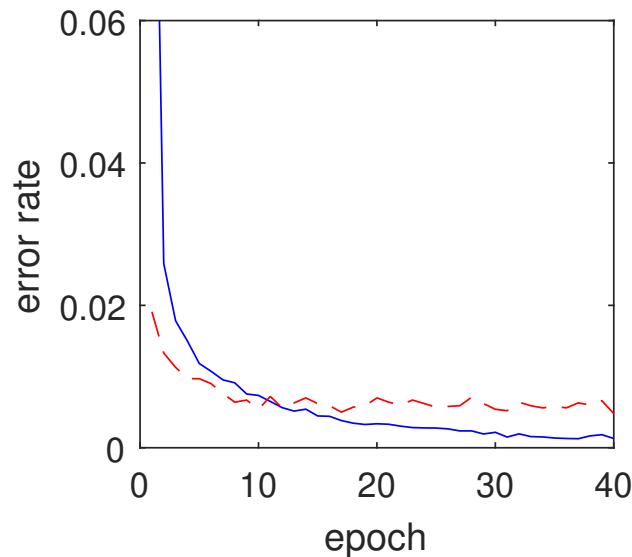
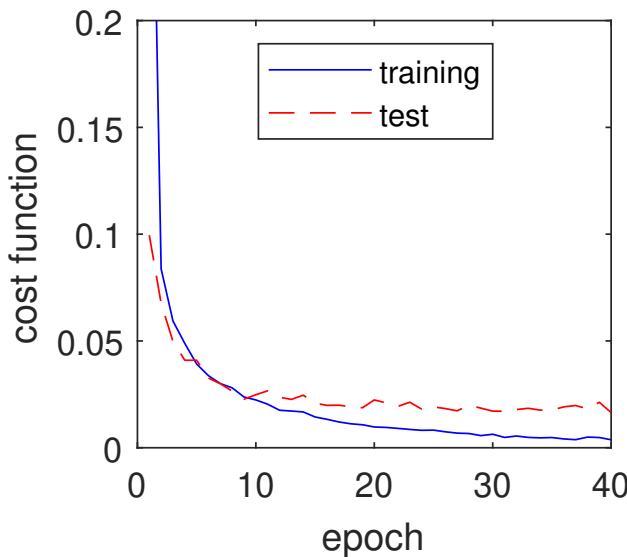
- categorical cross entropy

### Optimizer

- minibatch size  $B = 128$
- 40 epochs
- fixed step size  $\gamma^t = 0.01$
- momentum  $\beta = 0.9$  with Nesterov momentum

### E7.6: MNISTnet3 (4)

#### Results



- Early stopping after 15 epochs is recommended. After that, the network becomes again a little bit overfitted.
- Test error rate 0.48%, i.e. only 48 from 10,000 test images are wrong classified.

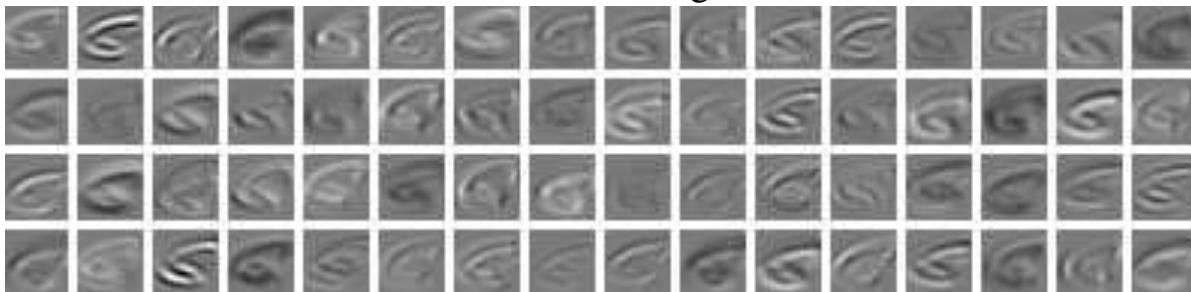
### E7.6: MNISTnet3 (5)

#### Wrong classified MNIST images



- They are really badly written! Even humans get trouble with some images.

Visualization of  $\mathbf{X}_1 \in \mathbb{R}^{22 \times 22 \times 64}$  for the first image "6" above



- The first convolutional layer mainly generates edges.
- Maybe that we do not need  $D_1 = 64$  feature maps. They look quite similar.

## MNIST experiments

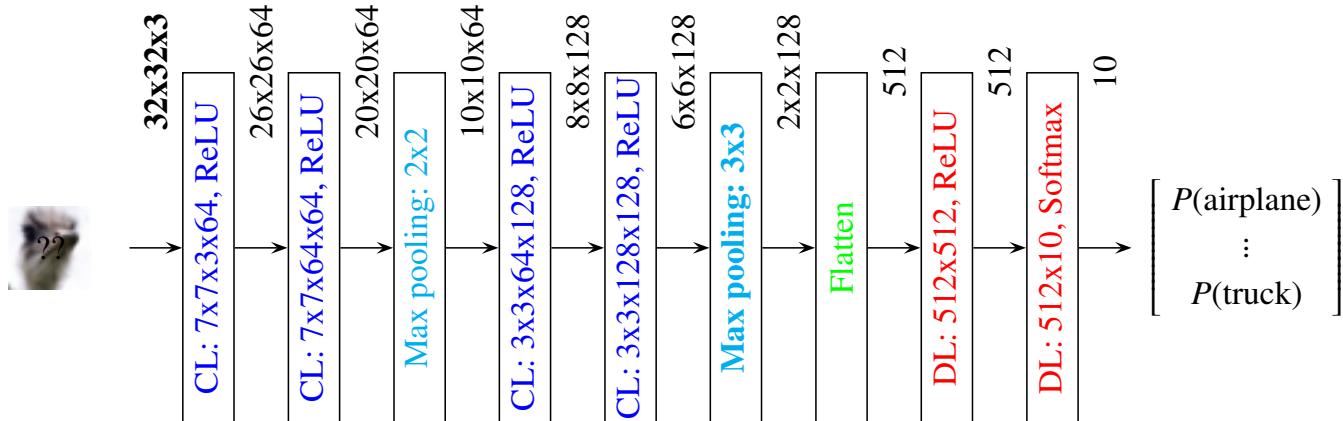
Example	E4.5	E5.4	E6.3	E7.6
Network $M_l$ $\phi_l$	1D MNISTnet1 784/1024/10 tanh/softmax	1D MNISTnet1 784/1024/10 tanh/softmax	1D MNISTnet2 784/512/512/512/10 tanh/softmax	2D MNISTnet3 4 CL, 2 DL ReLU/softmax
Purpose	baseline	optimizations	regularizations	solution
minibatch $B$	128	varying	128	128
epochs $N_{\text{epoch}}$	60	60	60	15
learning schedule	fixed	fixed	fixed	fixed
step size $\gamma$	0.1	varying	0.01	0.01
momentum $\beta$	0	varying	0.9	0.9
Nesterov	—	varying	yes	yes
$l_2$ -penalty $\lambda$	—	—	0.002	—
dropout rate	—	—	0.2	0.25/0.5
overfitting	yes	yes	no	no
test error rate	2.8%	2.5%	3%	0.48%

There are even better results than 0.48%.

## E7.7: CIFARnet (1)

A CNN for image recognition on CIFAR-10: 32x32 color images → 10 classes

### Architecture



- 4 convolutional layers (CL)
- 2 max pooling layers
- 1 flatten layer
- 2 dense layers (DL)

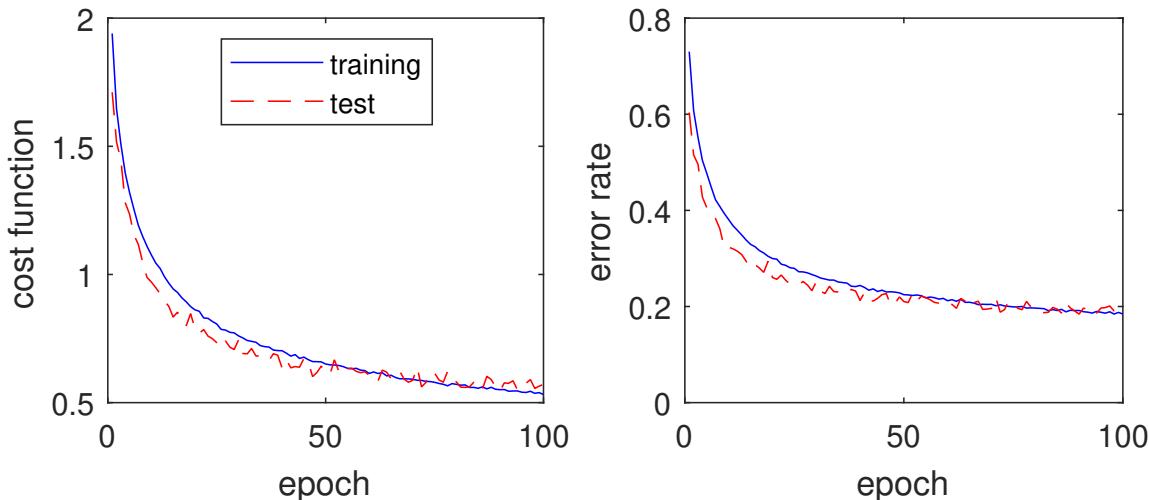
The same architecture as MNISTnet3 except for the second 3x3 max pooling layer.

## E7.7: CIFARnet (2)

### Data augmentation

- random rotation of images up to  $\pm 5^\circ$
- random translation of images up to  $\pm 20\%$  of the image size
- random horizontal flip of images

### Results



## E7.8: Image segmentation of street scenes (1)

**Image segmentation:** Pixelwise classification, see E1.6

### Dataset

- Cityscapes (Daimler), see page 2.24
- 5000 images: 2975/500/1525 for training/validation/test
- 8 classes: road/construction/object/nature/sky/human/vehicle/global
- image size downscaled to 224x224 to reduce complexity
- 48h training time on a GPU<sup>1</sup>

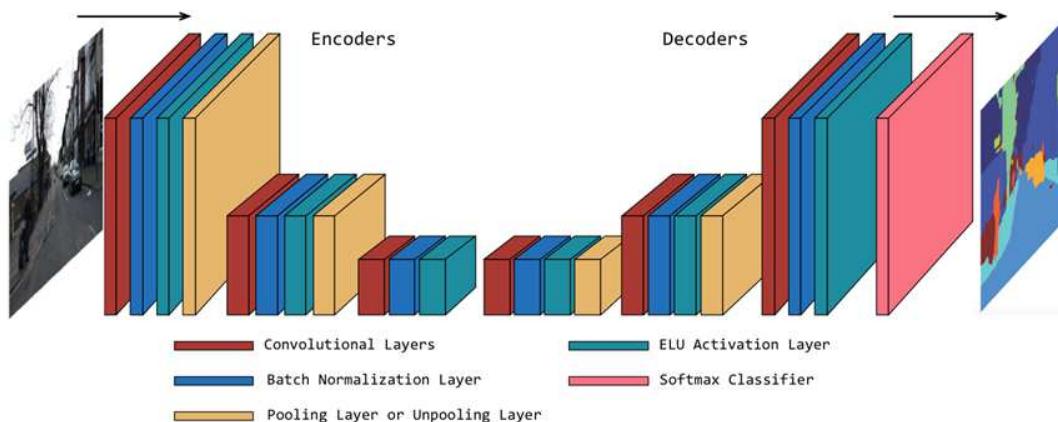
---

<sup>1</sup>Master thesis "Training convolutional neural network for semantic segmentation in consideration of easy extension for new object class", ISS, 2015

## E7.8: Image segmentation of street scenes (2)

### Network architecture

- a **fully convolutional network** consisting of 30 convolutional layers only
- each layer with padding, BN and ELU activation function
- first 15 (encoding) CLs: smaller image size by max pooling, more feature maps
- last 15 (decoding) CLs: larger image size by unpooling, less feature maps
- no flatten and dense layers
- shortcuts to avoid vanishing gradient



## E7.8: Image segmentation of street scenes (3)

### Network parameters

Layer	Kernel $K_l \times K_l \times D_{l-1} \times D_l$	Output $M_l \times N_l \times D_l$
Input		224x224x 3
3× CL Max Pooling	3x3x...	224x224x 64
	2x2	112x112x 64
3× CL Max Pooling	3x3	112x112x 128
	2x2	56x 56x 128
3× CL Max Pooling	3x3	56x 56x 256
	2x2	28x 28x 256
3× CL Max Pooling	3x3	28x 28x 512
	2x2	14x 14x 512
3× CL	3x3	14x 14x 1024
	...	...

Layer	Kernel	Output
	$K_l \times K_l \times D_{l-1} \times D_l$	$M_l \times N_l \times D_l$
...	...	...
3× CL	3x3	14x 14x 512
Unpooling	2x2	28x 28x 512
3× CL	3x3	28x 28x 256
Unpooling	2x2	56x 56x 256
3× CL	3x3	56x 56x 128
Unpooling	2x2	112x112x 128
3× CL	3x3	112x112x 64
Unpooling	2x2	224x224x 64
2× CL	3x3	224x224x 64
CL	3x3	224x224x 8
Softmax		224x224x 8

- $\approx 44$  million parameters
- $\approx 34$  billion of multiplications for one image

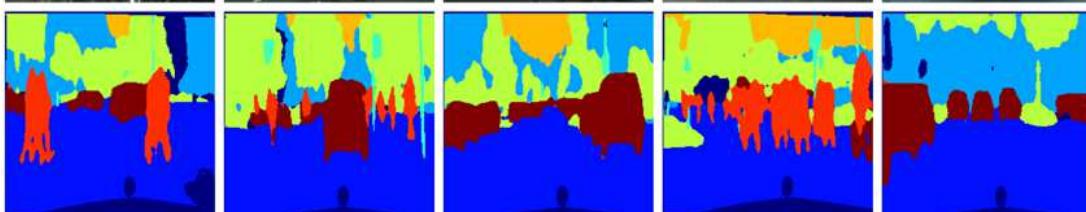
## E7.8: Image segmentation of street scenes (4)

### Results

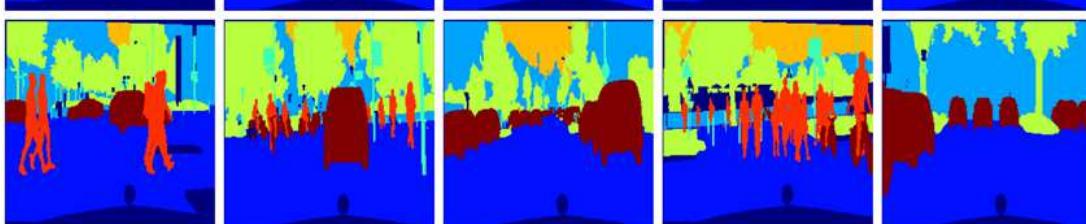
Test images



CNN output



Ground truth



Semantic image segmentation also important in medical imaging: Organ/fat/muscle/... segmentation

## FIR filter vs. CNN

	FIR filter	CNN
<b>Architecture</b>		
basic operation	convolution	convolution
activation function	no	yes
system behavior	linear	nonlinear
cascaded stages	no	yes
input feature maps	1	$\geq 1$
complexity	small	large
<b>Design</b>		
coefficients	calculated from filter specification	learned from data
<b>Purpose</b>		
application	linear filtering	nonlinear filtering

Hence CNN can be roughly interpreted as a nonlinear generalization of FIR filter with data-driven coefficients.

## Memory of nonrecursive and recursive filter (1)

A general **recursive filter** or **IIR filter**  $\text{IIR}(N, M)$  with the input  $x(n)$  and output  $y(n)$  is described by the recursive difference equation

$$y(n) + \sum_{i=1}^N a_i y(n-i) = \sum_{i=0}^M b_i x(n-i).$$

$y(n)$  is calculated recursively from  $y(n-1), \dots, y(n-N)$  and  $x(n), \dots, x(n-M)$ , see page 5-11 and Bachelor course "Digital signal processing". The simplest recursive filter is a first-order IIR(1,0) with  $y(n) + a_1 y(n-1) = b_0 x(n)$ . By using  $\beta = -a_1$ ,

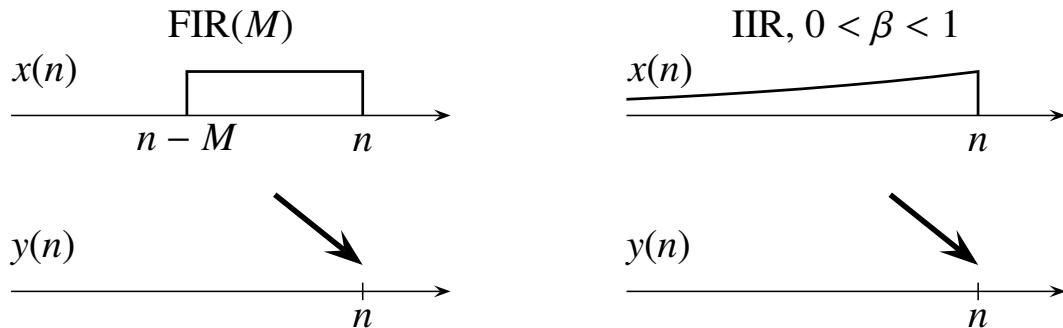
$$y(n) = \beta y(n-1) + b_0 x(n) = \dots = b_0 [x(n) + \beta x(n-1) + \dots + \beta^{n-1} x(1)] + \beta^n y(0).$$

This means,  $y(n)$  depends on *all* input samples  $x(n), \dots, x(1)$  from the beginning of the measurement till the current time instance  $n$ . An IIR filter has a long memory. In comparison, a **nonrecursive filter** or **FIR filter**  $\text{FIR}(M)$  of order  $M$

$$y(n) = \sum_{i=0}^M b_i x(n-i)$$

depends only on the last  $M + 1$  input samples and has a short memory.

## Memory of nonrecursive and recursive filter (2)



- Due to the feedback, IIR filter is the better choice than FIR filter to model systems with a long memory.

Roughly spoken, CNN is a nonlinear extension of FIR filter while RNN is a nonlinear extension of IIR filter.

## Feedforward vs. feedback neural networks

	Feedforward neural networks	Feedback neural networks
architecture	dense (ch. 4), CNN (ch. 7)	RNN (ch. 8)
feedback	no	yes
memory of neuron	no	yes
memory of network	short, depend on kernel size	long
input of network	a vector/matrix/tensor	a sequence of vectors/matrices/tensors
output of network	probabilities or a vector/matrix/tensor	probabilities or a sequence of vectors/matrices/tensors
temporal correlation	not considered	exploited

RNN processes typically sequential data (e.g. time series) with temporal correlation:

- speech or audio or acoustical signal
- written text
- video (sequence of images)
- ...

## E8.1: Applications of RNN

- a) Speech recognition: Translate a spoken speech (a sequence of phonemes) to a text (a sequence of words)
- b) Handwriting recognition: Translate an image of handwriting to a text
- c) Translation: Translate a speech/text from one language to another
- d) Image caption: Return a short text description for a given image. It typically uses a CNN to analyze the image and a RNN to generate natural language description.
- e) Music composition: Generate a well sounding music
- f) Ghost writer: Computer writes a grammatically correct text
- g) ...

In all applications above, the input and/or output are sequential data (speech, text, music) with a strong temporal correlation described by phonetics, grammar, melody etc.

## Unfolding of recurrent layer $l$

Network graph

$$\underline{s}_l(n) \xrightarrow{\quad}$$

$$\uparrow$$

$$\underline{x}_{l-1}(n)$$

Unfolded computational graph

$$\underline{s}_l(0) \xrightarrow{\mathbf{W}_{l,s}} \underline{s}_l(1) \xrightarrow{\mathbf{W}_{l,s}} \underline{s}_l(2) \xrightarrow{\mathbf{W}_{l,s}} \dots \xrightarrow{\mathbf{W}_{l,s}} \underline{s}_l(B)$$

$$\mathbf{W}_{l,x} \uparrow$$

$$\underline{x}_{l-1}(1)$$

$$\mathbf{W}_{l,x} \uparrow$$

$$\underline{x}_{l-1}(2)$$

$$\dots$$

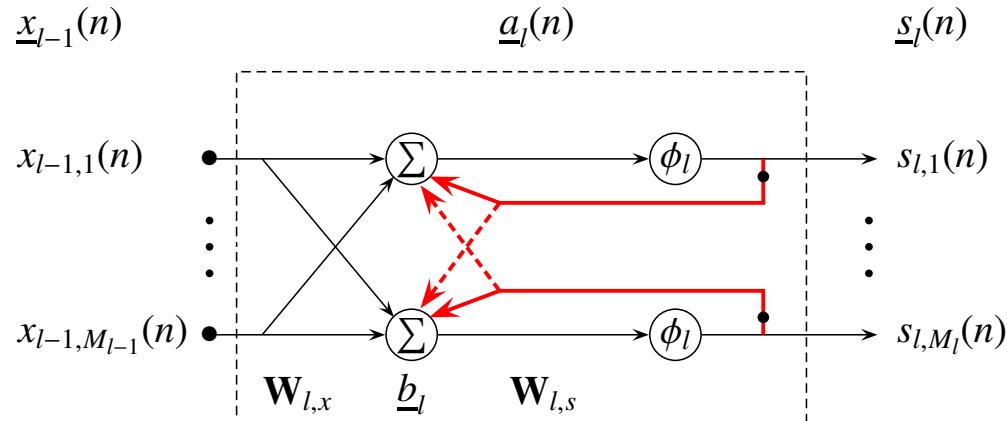
$$\mathbf{W}_{l,x} \uparrow$$

$$\underline{x}_{l-1}(B)$$

- **Unfolding** a time recursion  $\underline{s}_l(n) = f(\underline{x}_{l-1}(n), \underline{s}_l(n-1))$  along the time axis  $n = 1, 2, \dots, B$  yields a computational graph with repetitive structure. It is a convenient way to visualize the calculations in a RNN.
- The **unfolded graph** contains no feedback (cycles) and can be handled in the same way as feedforward networks.
- The same parameters  $\mathbf{W}_{l,x}, \mathbf{W}_{l,s}, \underline{b}_l$  are used in the unfolded graph for all time instances.

## Single-neuron vs. cross-neuron feedback

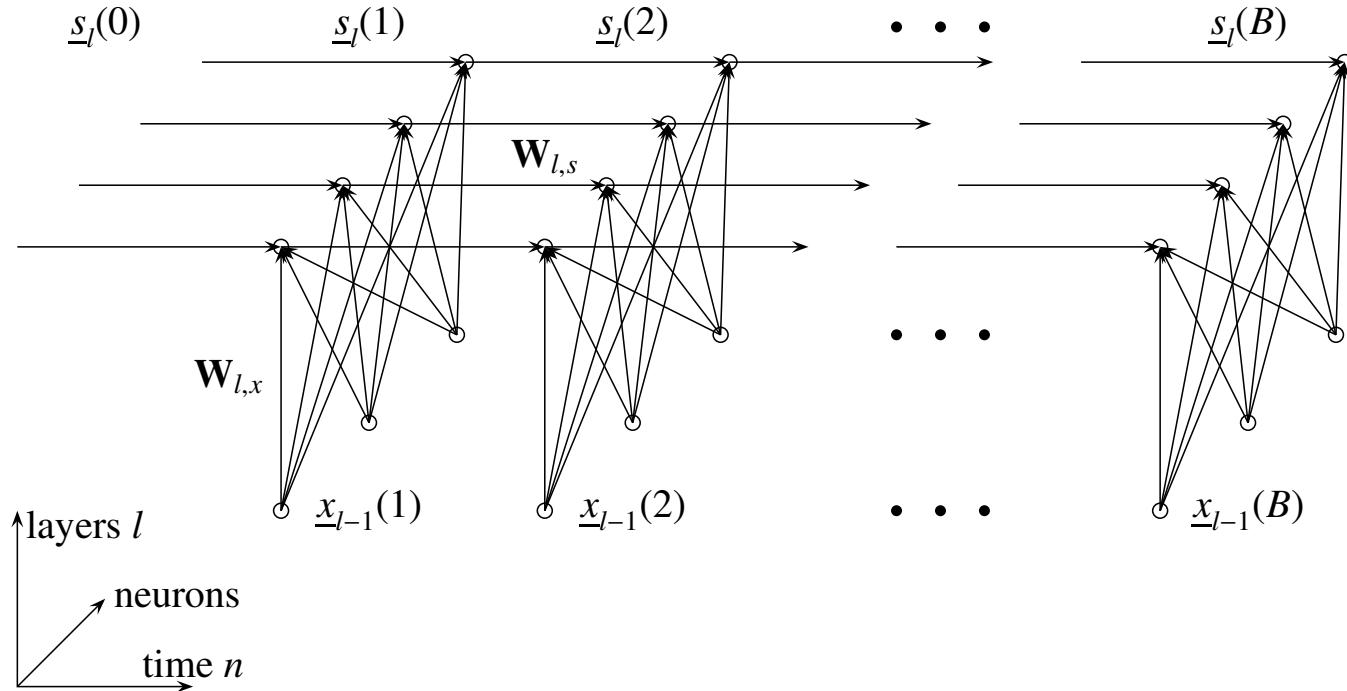
Recurrent layer  $l$ :



- feedforward:  $\mathbf{W}_{l,s} = \mathbf{0}$   $\longrightarrow$
- single-neuron feedback:  $\mathbf{W}_{l,s}$  diagonal,  $\longrightarrow$   $\longrightarrow$
- cross-neuron feedback:  $\mathbf{W}_{l,s}$  non-diagonal,  $\longrightarrow$   $\longrightarrow$   $\dashrightarrow$

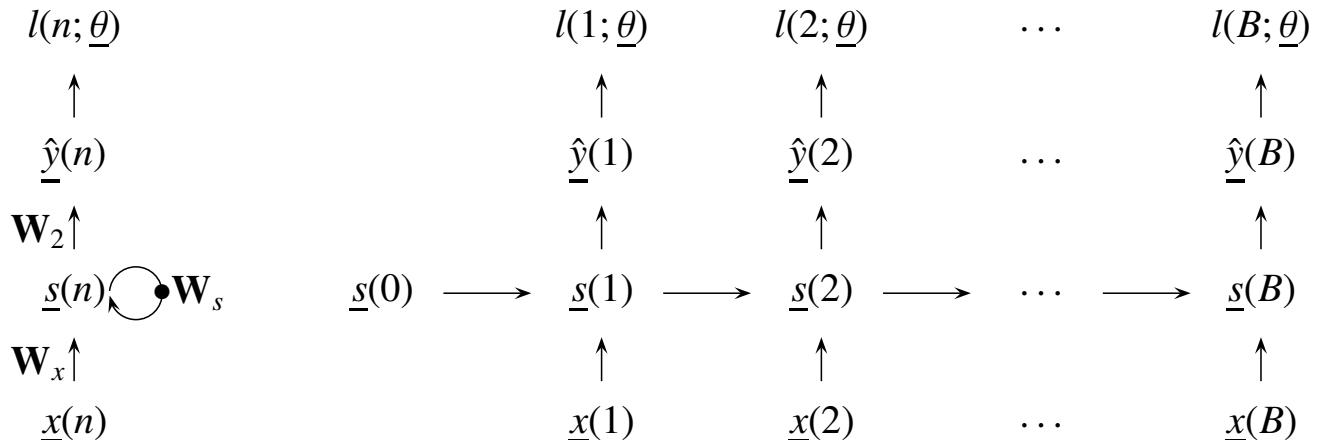
## Unfolded recurrent layer in details

with  $M_{l-1} = 3, M_l = 4$  and single-neuron feedback



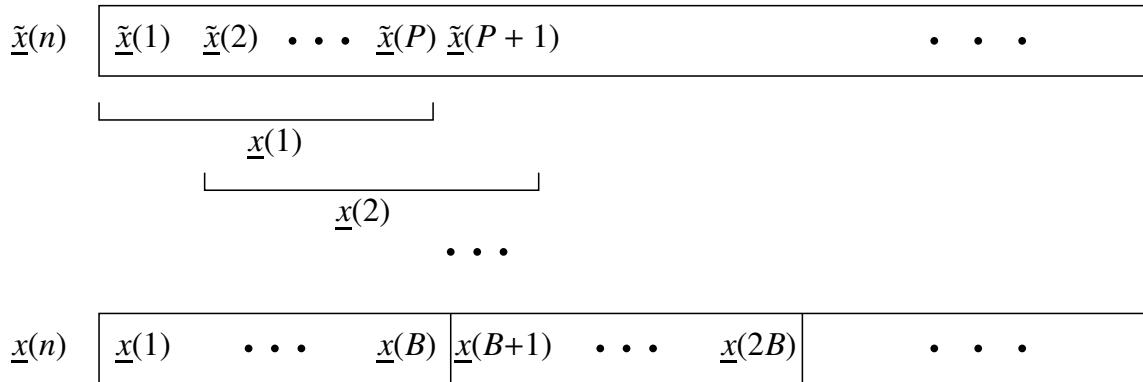
## E8.2: A simple RNN

- one input layer  $\underline{x}(n)$
- one recurrent layer  $\underline{s}(n) = \phi_1(\mathbf{W}_x \underline{x}(n) + \mathbf{W}_s \underline{s}(n-1) + \underline{b}_1)$
- one output layer  $\hat{\underline{y}}(n) = \phi_2(\mathbf{W}_2 \underline{s}(n) + \underline{b}_2)$ ,  $\phi_2()$  = softmax() for classification
- loss  $l(n; \underline{\theta})$  between RNN output  $\hat{\underline{y}}(n)$  and ground truth  $\underline{y}(n)$ , see slide 4-15
- cost function  $L(\underline{\theta}) = \frac{1}{B} \sum_{n=1}^B l(n; \underline{\theta})$  for one minibatch  $1 \leq n \leq B$
- $\underline{\theta}$  contains all parameters from  $\mathbf{W}_x, \mathbf{W}_s, \underline{b}_1, \mathbf{W}_2, \underline{b}_2$



In practice, an RNN may contain multiple recurrent layers.

## Organisation of input data

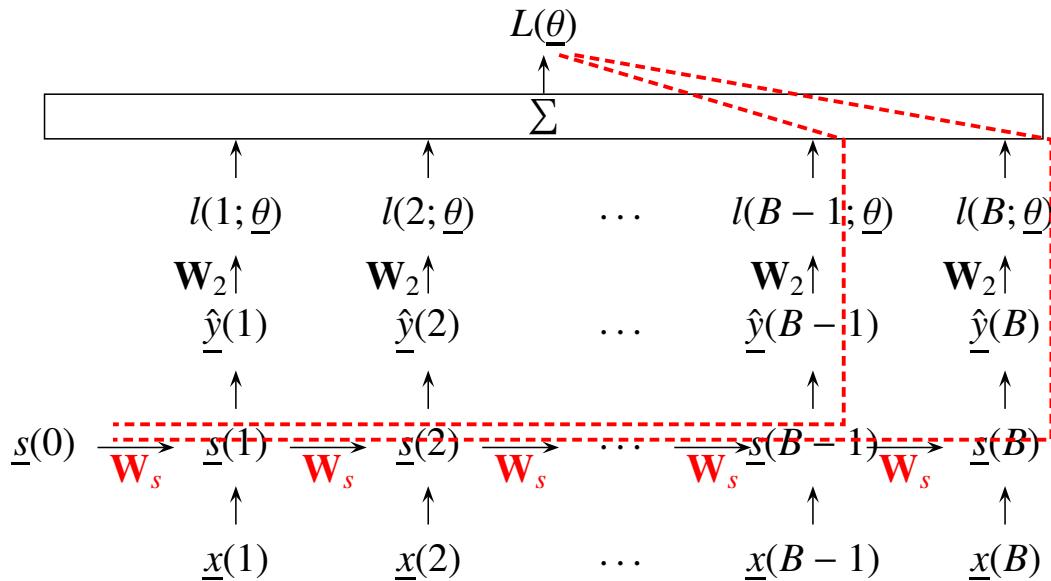


- the original input data  $\underline{\tilde{x}}(n) \in \mathbb{R}^d$  contains  $d$  time series<sup>1</sup>
- form a new sequence of input vectors  $\underline{x}(n) = [\underline{\tilde{x}}^T(n), \underline{\tilde{x}}^T(n+1), \dots, \underline{\tilde{x}}^T(n+P-1)]^T \in \mathbb{R}^{Pd}$  by using a sliding window of length  $P \geq 1$ . This corresponds to the non-recursive part of a recursive filter.
- the choice of the window length  $P$  depends on the application
- divide  $\underline{x}(n)$  into non-overlapping minibatches of length  $B$
- $\underline{x}(tB + n)$ ,  $1 \leq n \leq B; t = 0, 1, \dots$  is the  $t$ -th minibatch of input for the RNN

+

<sup>1</sup>If the input is a sequence of images or tensors, then a 2D or 3D RNN has to be used. Actually, the idea of recurrent layer can also be applied to CNN.

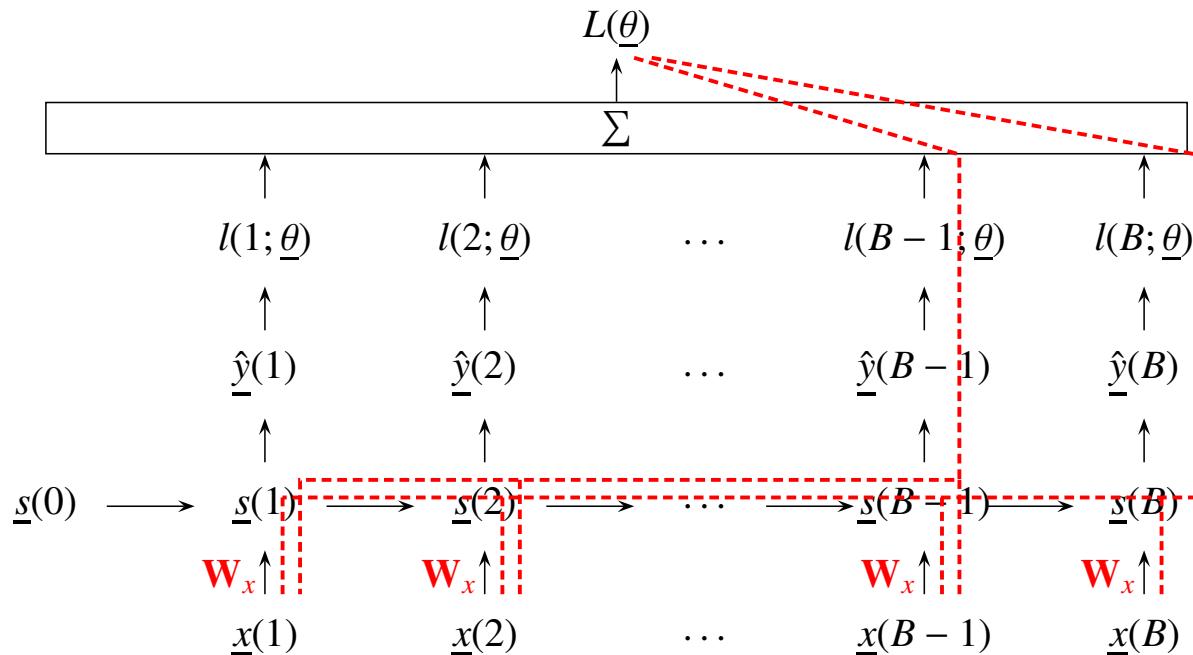
## How many paths from $\mathbf{W}_s$ to $L(\underline{\theta})$ in E8.2?



In  $\underline{s}(n) = \phi_1(\mathbf{W}_x \underline{x}(n) + \mathbf{W}_s \underline{s}(n-1) + \underline{b}_1)$ , both  $\mathbf{W}_s$  and  $\underline{s}(n-1)$  depend on  $\mathbf{W}_s$ . By applying the product rule of derivative, there are

- $B$  paths from  $\mathbf{W}_s$  to  $L(\underline{\theta})$  through  $l(B; \underline{\theta})$ , namely those over  $\underline{s}(1), \dots, \underline{s}(B)$ ,
- $B-1$  paths from  $\mathbf{W}_s$  to  $L(\underline{\theta})$  through  $l(B-1; \underline{\theta})$ , namely those over  $\underline{s}(1), \dots, \underline{s}(B-1)$ ,
- ...

## How many paths from $\mathbf{W}_x$ to $L(\underline{\theta})$ in E8.2?



$$\underline{s}(n) = \phi_1(\mathbf{W}_x \underline{x}(n) + \mathbf{W}_s \underline{s}(n-1) + \underline{b}_1)$$

## Motivation of bidirectional recurrent neural network

RNN: Recursive calculation  $\underline{s}(n - 1) \rightarrow \underline{s}(n)$  in the forward time direction. This corresponds to a causal behavior: Presence depends on past.

In some applications, however, a non-causal behavior is desired because the output  $\hat{y}(n)$  may depend on both past and future of input  $\underline{x}(n)$ , e.g.

- Text recognition: Linguistic dependency of current word on past and future words.
- Speech recognition: Co-articulation, i.e. the current phoneme also depends on the next few phonemes.

### Bidirectional recurrent neural network (BRNN)<sup>2</sup>

- presents one minibatch of training data in the forward- and backward-time direction to two separate recurrent layers and
- concatenation of both outputs.

This provides the next layer with complete past and future information of the input sequence at each time instance.

## Forward pass of a BRNN for one minibatch

Given:  $\underline{x}_{l-1}(n), 1 \leq n \leq B$

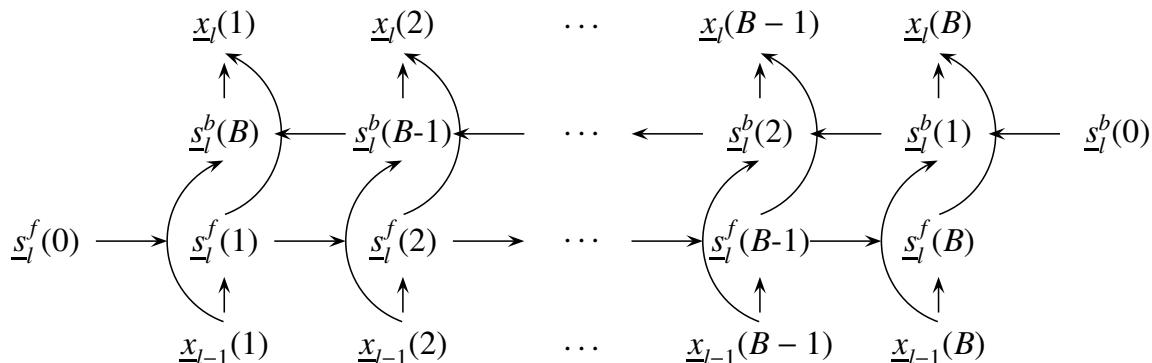
Initial values:  $\underline{s}_l^f(0), \underline{s}_l^b(0)$

FOR  $n = 1, 2, \dots, B$  DO

$$\underline{s}_l^f(n) = \phi_l(\mathbf{W}_{l,x}^f \underline{x}_{l-1}(n) + \mathbf{W}_{l,s}^f \underline{s}_l^f(n-1) + \underline{b}_l^f)$$

$$\underline{s}_l^b(n) = \phi_l(\mathbf{W}_{l,x}^b \underline{x}_{l-1}(B-n+1) + \mathbf{W}_{l,s}^b \underline{s}_l^b(n-1) + \underline{b}_l^b)$$

$$\underline{x}_l(n) = \begin{bmatrix} \underline{s}_l^f(n) \\ \underline{s}_l^b(B-n+1) \end{bmatrix}$$



## A LSTM cell

A **long short-term memory (LSTM)** cell/neuron/unit/block replaces a normal recurrent neuron in RNN. It contains<sup>3</sup>

- a memory storing the state  $s(n)$  at time  $n$  like a recurrent neuron and
- three multiplicative gates, the **input/forget/output gate**, which control the write/reset /read operation of the memory state.

## A LSTM layer $l$ (1)

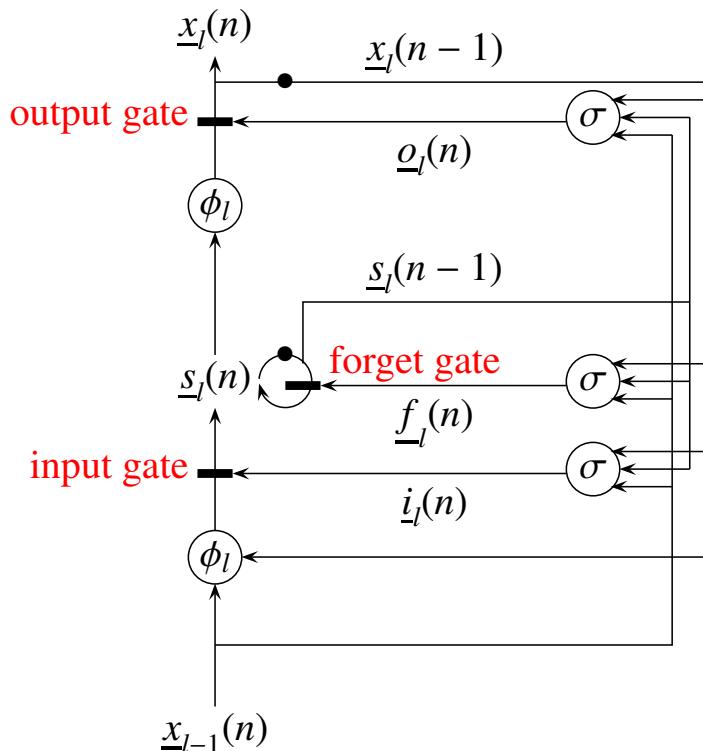
containing  $M_l$  LSTM cells:

$\underline{x}_{l-1}(n) \in \mathbb{R}^{M_{l-1}}$	layer input at time $n$
$\underline{x}_l(n) \in \mathbb{R}^{M_l}$	layer output at time $n$
$\underline{s}_l(n) \in \mathbb{R}^{M_l}$	memory state at time $n$ , $\underline{s}_l(n) \neq \underline{x}_l(n)$ in contrast to RNN
$\underline{i}_l(n) \in \mathbb{R}^{M_l}$	<b>input gate signal</b> at time $n$
$\underline{f}_l(n) \in \mathbb{R}^{M_l}$	<b>forget gate signal</b> at time $n$
$\underline{o}_l(n) \in \mathbb{R}^{M_l}$	<b>output gate signal</b> at time $n$

---

<sup>3</sup>S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.

## A LSTM layer $l$ (2)



- memory

— gate: elementwise multiplication  $\odot$

( $\sigma$ ) neuron with sigmoid activation function

( $\phi_l$ ) neuron with any activation function

## A LSTM layer $l$ (3)

### Gate signals

$$\begin{aligned}\underline{i}_l(n) &= \sigma(\mathbf{W}_{l,ix}\underline{x}_{l-1}(n) + \mathbf{W}_{l,is}\underline{s}_l(n-1) + \mathbf{W}_{l,io}\underline{x}_l(n-1) + \underline{b}_{l,i}), \\ \underline{f}_l(n) &= \sigma(\mathbf{W}_{l,fx}\underline{x}_{l-1}(n) + \mathbf{W}_{l,fs}\underline{s}_l(n-1) + \mathbf{W}_{l,fo}\underline{x}_l(n-1) + \underline{b}_{l,f}), \\ \underline{o}_l(n) &= \sigma(\mathbf{W}_{l,ox}\underline{x}_{l-1}(n) + \mathbf{W}_{l,os}\underline{s}_l(n-1) + \mathbf{W}_{l,oo}\underline{x}_l(n-1) + \underline{b}_{l,o}).\end{aligned}$$

They all have the range  $(0, 1)$ . Typically,  $\mathbf{W}_{l,*s}$  and  $\mathbf{W}_{l,*o}$  are diagonal, i.e. each gate signal is affected by the memory state and output of the same LSTM cell.

### Update memory state

$$\underline{s}_l(n) = \underline{f}_l(n) \odot \underline{s}_l(n-1) + \underline{i}_l(n) \odot \phi_l(\mathbf{W}_{l,sx}\underline{x}_{l-1}(n) + \mathbf{W}_{l,so}\underline{x}_l(n-1) + \underline{b}_{l,s})$$

### Layer output

$$\underline{x}_l(n) = \underline{o}_l(n) \odot \phi_l(\underline{s}_l(n))$$

Variants of feedback:

- only feedback from memory state:  $\mathbf{W}_{l,*o} = \mathbf{0}$
- only feedback from output:  $\mathbf{W}_{l,*s} = \mathbf{0}$

## Function of gates (1)

LSTM-RNN is the most successful type of RNN.

In the forward pass:

- The gating signals  $i_l(n)$ ,  $f_l(n)$ ,  $\underline{o}_l(n)$  are time-varying. This enables a dynamic and flexible short/long-term storage and access of information.

input gate	forget gate	output gate	memory state
open			write into memory
close	open		memory protected from overwriting and forgetting
		open	read from memory
	close		clear memory

- Instead of making manual decisions for opening/closing gates, a LSTM-RNN learns to open/close the gates automatically, conditioned on the input  $x_{l-1}(n)$ , state  $s_l(n-1)$  and output  $x_l(n-1)$ .

## Function of gates (2)

In the backward pass:

- Conventional RNN:

$$\underline{s}_l(n) = \phi_l(\mathbf{W}_{l,x}\underline{x}_{l-1}(n) + \mathbf{W}_{l,s}\underline{s}_l(n-1) + \underline{b}_l)$$

$\underline{s}_l(n-1)$  is inside the activation function  $\phi_l(\cdot)$ . Hence vanishing gradient happens during the backpropagation  $\frac{\partial L}{\partial \underline{s}_l(n)} \rightarrow \frac{\partial L}{\partial \underline{s}_l(n-1)}$  if  $\left| \frac{\partial \phi_l(a)}{\partial a} \right| < 1$ .

- LSTM-RNN:

$$\underline{s}_l(n) = \underline{f}_l(n) \odot \underline{s}_l(n-1) + \underline{i}_l(n) \odot \phi_l(\mathbf{W}_{l,sx}\underline{x}_{l-1}(n) + \mathbf{W}_{l,so}\underline{x}_l(n-1) + \underline{b}_{l,s})$$

$\underline{s}_l(n-1)$  is outside the activation function  $\phi_l(\cdot)$ . If the forget gate is open, i.e.  $\underline{f}_l(n)$  close to one, there is no gradient attenuation during the backpropagation  $\frac{\partial L}{\partial \underline{s}_l(n)} \rightarrow \frac{\partial L}{\partial \underline{s}_l(n-1)}$ . Hence a LSTM-RNN can have a long memory without vanishing gradients.

### E8.3: MIDI composition (1)

Musical instrument digital interface (**MIDI**): A digital standard for describing and exchanging music information between electronic musical instruments

#### Training set

- single-channel ( $d = 1$ ) MIDI data  $\tilde{x}(n)$
- integer coded MIDI data  $\tilde{x}(n) \in \{1, 2, \dots, 375\}$  for  $C = 375$  possible single tones (pressing one key) and accords (pressing multiple keys)
- 188 MIDI files of classical piano (Bach, Beethoven, Brahms, Chopin, ...)
- tone duration roughly 1/4s
- in total 222407 training tones (total duration 15.4h)
- sliding window length  $P = 100$ , i.e.  $\underline{x}(n) = [\tilde{x}(n), \tilde{x}(n+1), \dots, \tilde{x}(n+P-1)]^T \in \mathbb{N}^{100}$
- minibatch size  $B = 128$

**Task:** MIDI "composition". Given the first  $P = 100$  MIDI tones, the RNN should continue to "compose" in a sliding-window way:

$$\tilde{x}(n), \tilde{x}(n+1), \dots, \tilde{x}(n+P-1) \rightarrow \tilde{x}(n+P), \quad n = 1, 2, \dots$$

## E8.3: MIDI composition (2)

### RNN architecture

- input layer of  $P = 100$  neurons
- 1. recurrent layer of 512 LSTM units and 30% dropout
- 2. recurrent layer of 512 LSTM units and 30% dropout
- 3. recurrent layer of 512 LSTM units and 30% dropout
- 1. dense layer of 256 neurons and 30% dropout
- 2. dense layer of 375 neurons
- output layer with softmax for 375 possible tones/accords at the next time instant
- categorical cross-entropy loss (classification of 375 classes)

### E8.3: MIDI composition (3)

#### Hearing experiments

- a) an original MP3 file from Chopin (Prelude Op. 28 No. 15)
- b) MP3 converted to MIDI with the original time-varying tone duration
- c) MP3 converted to MIDI with a fixed ton duration
- d) MIDI "composed" by the LSTM-RNN.
  - The first  $P/4 = 25$ s contain the initialization  $\tilde{x}(1), \dots, \tilde{x}(P)$  from c)
  - The rest  $\tilde{x}(P + 1), \tilde{x}(P + 2), \dots$  is generated by the LSTM-RNN.
- e) MIDI containing randomly selected tones/acords
- f) another "composed" MIDI by the LSTM-RNN

See also E1.11 for Schubert's unfinished symphony finished by AI.

## Unsupervised learning (1)

### Unsupervised learning

- learn the structure of unlabeled input samples  $\underline{x}(n)$ ,  $1 \leq n \leq N$ . See below for more about "structure".
- no ground truth  $\underline{y}(n)$  available
- no classification and regression

### Why unsupervised learning?

- massive amount of unlabeled data available (e.g. music, images, videos, sensor data on internet)
- labeling is very expensive
- need better representation of raw data
- unsupervised learning as a preprocessing for other tasks like supervised learning, sorting, comparison etc.

## Unsupervised learning (2)

### Tasks of unsupervised learning

- **Clustering:** Group input samples to different clusters based on their similarity
- **Dimension reduction:** Reduce the number of elements of a sample  $\underline{x}(n)$  or the number of extracted features without considerable information loss
- **Feature learning, representation learning, manifold learning:** Discover low-dimensional features/representations of  $\underline{x}(n)$  required for a certain task like classification. This replaces the manual feature extraction in traditional machine learning.

### Traditional methods for unsupervised learning

- Clustering:  $k$ -means, Gaussian mixture model (GMM), mean-shift, DBSCAN, ...
- Dimension reduction: principal component analysis (PCA), ...

see course DPR.

## E9.1: Representations (1)

Any measured signal (speech, image, ...) may have different representations. Its native representation, the raw data of measurement, is not always optimum for a certain task. Finding a suitable representation can greatly simplify the task.

- a) Time- vs. frequency-domain representation in signal processing
- b) The arabic notation is much easier for numerical calculation than the Roman one.

0	1	2	3	4	5	6	7	8	9	10	50	100	500	1000
-	I	II	III	IV	V	VI	VII	VIII	IX	X	L	C	D	M

Roman multiplication	Arabic multiplication
$\begin{array}{r} \text{XVIII} \\ \times \quad \text{XXII} \\ \hline \end{array}$ (18) (22)	$\begin{array}{r} 18 \\ \times \quad 22 \\ \hline \end{array}$
$\begin{array}{r} \text{VI} \\ \text{XXX} \\ \hline \end{array}$ (2*3=6) (2*15=30)	$\begin{array}{r} 36 \\ + \quad 36 \\ \hline \end{array}$
$\begin{array}{r} \text{CLX} \\ + \quad \text{CC} \\ \hline \end{array}$ (20*8=160) (20*10=200)	$\begin{array}{r} 396 \\ = \quad \text{CCCXCVI} \end{array}$

- c) Binary representation on computers

## E9.1: Representations (2)

### d) MNIST

There are much more information contained in MNIST than just the digit labels:

- appearance
- orientation
- identity of writer
- ...



The native representation of MNIST, the pixels, is not optimum for digit recognition because a small distortion to one image can change the digit:

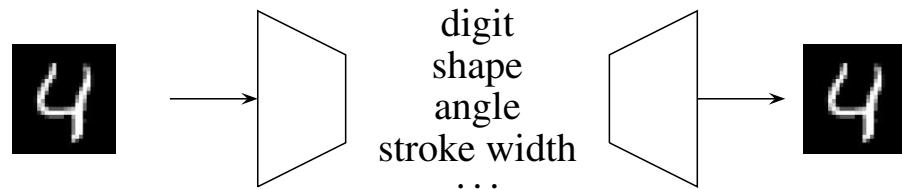
$$1 \leftrightarrow 7, \quad 4 \leftrightarrow 9, \quad 0 \leftrightarrow 6, \quad \dots$$

This is why MNISTnet3 in E7.6 needs multiple convolutional layers to first extract a more discriminative representation of the MNIST images before classification.

## E9.2: Latent variable

The latent variable  $\underline{z}$  is a hidden and compressed representation/code for the input  $\underline{x}$  because  $\underline{x}$  can be well reconstructed from  $\underline{z}$  with only small differences.  $\underline{z}$  may contain information about:

a) MNIST:



b) Face:



## Comments on autoencoder

- A perfect self-copy  $\hat{x} = \underline{x}$  is never the purpose of an autoencoder. Actually, it is designed to be *unable* to do that. An autoencoder is restricted to reconstruct the input  $\underline{x}$  approximately.
- There are different ways to achieve this:
  - Use an **undercomplete autoencoder** with  $c \ll d$ . This forces the autoencoder to capture the most important features of the input data.
  - Use a **regularized autoencoder** with a loss function containing e.g. the regularization term  $\|\underline{z}\|_1$  to force a sparse latent representation  $\underline{z}$ .
- In an autoencoder, we are not interested in the output  $\hat{x}$ . Instead, the latent variable  $\underline{z}$  as a compressed representation for input  $\underline{x}$  is of interest.
- An autoencoder can be trained in exactly the same way as supervised learning with the "ground truth"  $y = \underline{x}$ . The latent representation  $\underline{z}$  is learned automatically from the unlabeled data  $\underline{x}(n), 1 \leq n \leq N$ .
- PCA is a special case of autoencoder with a linear encoder  $\underline{z} = \mathbf{W}^T \underline{x}$ , a linear decoder  $\hat{x} = \mathbf{W}\underline{z}$  and the MSE loss  $E(\|\underline{x} - \hat{\underline{x}}\|^2)$ , see course DPR. In other words, autoencoder is a more powerful nonlinear generalization of PCA.

## E9.3: MNISTnet4 - Denoising autoencoder (1)

### Task

- denoise a noisy digit image

### Data set

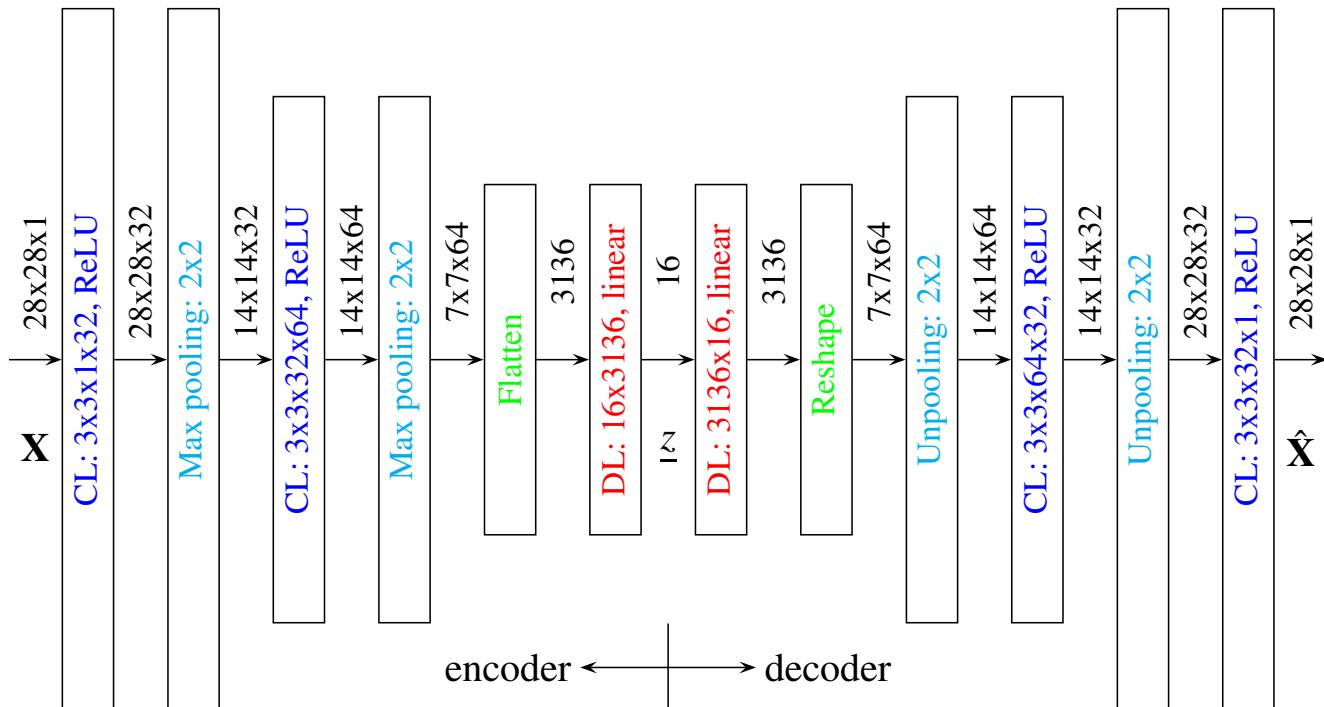
- MNIST image + Gaussian pixel noise  $N(0, 0.5^2)$  as input for the autoencoder
- corresponding clean MNIST image as desired output

### Training

- $l_2$ -loss
- minibatch size 128
- 30 epochs
- Adam optimizer
- learning rate  $\gamma = 0.001$

### E9.3: MNISTnet4 - Denoising autoencoder (2)

**Autoencoder:** Using a latent space of  $\mathbb{R}^{16}$  for a 28x28 image

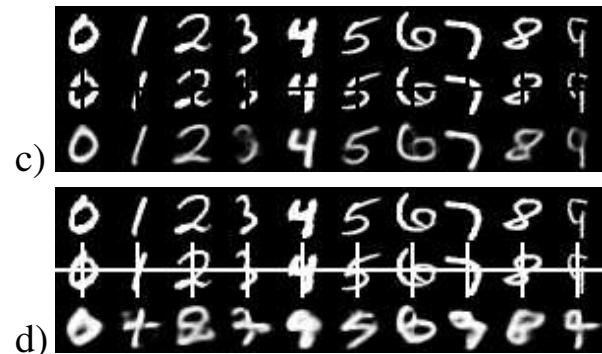
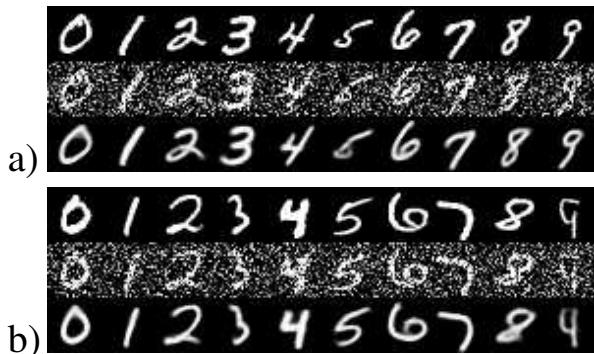


Padding is used here for each convolutional layer to maintain its input image size.

### E9.3: MNISTnet4 - Denoising autoencoder (3)

#### Results

- From top to bottom in each block: original MNIST, noisy/corrupted MNIST, denoised MNIST
- a) From training set of noisy MNIST
- b) From test set of noisy MNIST
- c) middle rows and columns in each MNIST image set to 0 (dark)
- d) middle rows and columns in each MNIST image set to 1 (bright)



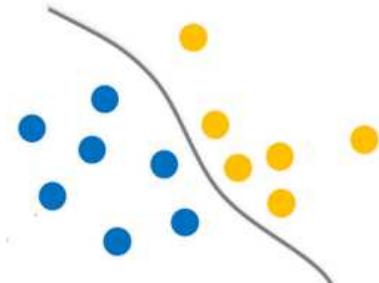
The denoising autoencoder performs well for learned noise in b), but bad for unseen corruption in d).

## Discriminative vs. generative models

In supervised learning,  $\underline{x}$  is the observation and  $\underline{y}$  the desired output.

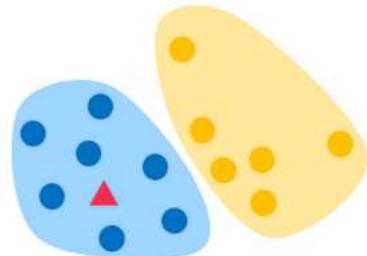
### Discriminative models (algorithms):

- learn the (posterior) conditional distribution  $p(\underline{y}|\underline{x})$
- useful for discrimination tasks  $\max_{\underline{y}} p(\underline{y}|\underline{x})$  like classification and regression
- focus on decision boundaries
- dense network, CNN, RNN in ch. 4, 7, 8



### Generative models (algorithms):

- learn the joint distribution  $p(\underline{x}, \underline{y})$
- useful for discrimination due to  $p(\underline{y}|\underline{x}) = p(\underline{x}, \underline{y})/p(\underline{x})$ , but more important for generation of  $(\underline{x}, \underline{y})$
- more challenging than discriminative models
- VAE in ch. 9.2 and GAN in ch. 9.3



In unsupervised learning with  $\underline{x}$  only, generative models learn the distribution  $p(\underline{x})$ .

## E9.4: Discriminative vs. generative models

a) Painting  $\underline{x}$  and painting style  $y \in \{\text{oil, ink}\}$



- Discriminative model  $p(y|\underline{x})$ : Can distinguish between both painting styles  $y$  for a given painting  $\underline{x}$  without being able to paint in these styles.
- Generative model  $p(\underline{x}, y)$ : Can paint in both styles and, naturally, can also distinguish between them.

b) Speech  $\underline{x}$  and language  $y \in \{\text{English, German, ...}\}$

I am generative in English, German and Chinese, but I am only discriminative between, say, Japanese and Arabic.

## Autoencoder vs. variational autoencoder



An autoencoder (AE) is not generative:

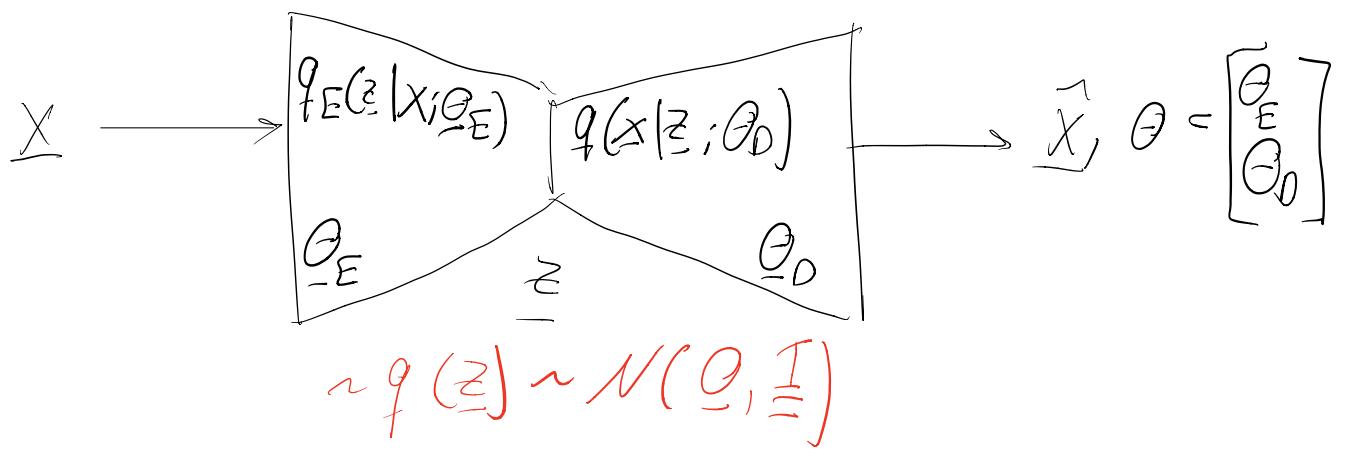
Though we find a suitable latent space  $\underline{z}$  for  $\underline{x}$ , we don't know the distribution of  $\underline{z}$ ; it can be arbitrary. Without the encoded and stored "code"  $\underline{z}$  for  $\underline{x}$ , there is no way to calculate  $\hat{\underline{x}}$ . An AE just memories the input  $\underline{x}$  in terms of its code  $\underline{z}$ ; it cannot generate new  $\hat{\underline{x}}$  because we don't know how to create  $\underline{z}$  without  $\underline{x}$ .

A **variational autoencoder (VAE)** is generative:

It is similar to an AE except for the additional constraint that the latent variable  $\underline{z}$  has a known distribution, say  $N(\underline{0}, \mathbf{I})$ <sup>1</sup>. After training, the encoder is no longer required. We can draw random samples of  $\underline{z}$  from  $N(\underline{0}, \mathbf{I})$  and use the **decoder**, a deterministic nonlinear mapping, to translate  $\underline{z}$  to new realistic-looking input samples  $\hat{\underline{x}}$  without  $\underline{x}$ .

+

<sup>1</sup>The actual distribution of  $\underline{Z}$  is not relevant. The first layers of the decoder are able to map  $N(\underline{0}, \mathbf{I})$  to any other desired distribution in the latent space.



i.e.  $q(\underline{x}; \theta) = \int q(\underline{x}, z; \theta) dz = \int q(\underline{x}|z; \theta) \cdot q(z) dz$

integral hard to calculate      ↑ well known  $N(\underline{\theta}, \underline{I})$   
 no analytic function

## 1. Trick

Replace  $-\ln(q(\underline{x}; \theta))$  by its variational upper bound

Upper bound

$$-\ln(q(\underline{x}; \theta)) \leq \underbrace{\dots}_{\text{can be easily calculated}}$$

## Variational upper bound for $-\ln p(\underline{x})$ (1) ★

$\underline{X}$  and  $\underline{Z}$  have the joint PDF  $p(\underline{x}, \underline{z})$  with the corresponding marginal PDFs  $p(\underline{x}), p(\underline{z})$  and conditional PDFs  $p(\underline{z}|\underline{x}), p(\underline{x}|\underline{z})$ .  $q(\underline{z}|\underline{x})$  is another PDF approximating  $p(\underline{z}|\underline{x})$ . Often the integral in  $-\ln p(\underline{x}) = -\ln \int p(\underline{x}|\underline{z})p(\underline{z})d\underline{z}$  is difficult to calculate. But it has been proven:

$$-\ln p(\underline{x}) \leq -E_{\underline{z} \sim q(\underline{z}|\underline{x})} \ln p(\underline{x}|\underline{Z}) + D_{KL}(q(\underline{z}|\underline{x}) \| p(\underline{z})) .$$

The right-hand side is called the **variational upper bound** for  $-\ln p(\underline{x})$  and is easier to calculate (see next slide).

Proof:

$$\begin{aligned} D_{KL}(q(\underline{z}|\underline{x}) \| p(\underline{z}|\underline{x})) - D_{KL}(q(\underline{z}|\underline{x}) \| p(\underline{z})) &= E_{\underline{z} \sim q(\underline{z}|\underline{x})} \ln \left( \frac{q(\underline{Z}|\underline{x})}{p(\underline{Z}|\underline{x})} \right) - E_{\underline{z} \sim q(\underline{z}|\underline{x})} \ln \left( \frac{q(\underline{Z}|\underline{x})}{p(\underline{Z})} \right) \\ &= E_{\underline{z} \sim q(\underline{z}|\underline{x})} \ln \left( \frac{p(\underline{Z})}{p(\underline{Z}|\underline{x})} \right) = E_{\underline{z} \sim q(\underline{z}|\underline{x})} \ln \left( \frac{p(\underline{x})}{p(\underline{x}|\underline{Z})} \right) = \ln p(\underline{x}) - E_{\underline{z} \sim q(\underline{z}|\underline{x})} \ln p(\underline{x}|\underline{Z}). \end{aligned}$$

The Bayes rule  $p(\underline{z}|\underline{x}) = p(\underline{x}|\underline{z})p(\underline{z})/p(\underline{x})$  is used in the last line. Since  $D_{KL}$  is always non-negative,

$$-\ln p(\underline{x}) \leq -\ln p(\underline{x}) + D_{KL}(q(\underline{z}|\underline{x}) \| p(\underline{z}|\underline{x})) = -E_{\underline{z} \sim q(\underline{z}|\underline{x})} \ln p(\underline{x}|\underline{Z}) + D_{KL}(q(\underline{z}|\underline{x}) \| p(\underline{z})).$$

The equality holds if  $q(\underline{z}|\underline{x})$  and  $p(\underline{z}|\underline{x})$  are identical.

## Variational upper bound for $-\ln p(\underline{x})$ (2) ★

Comments:

- The variational upper bound looks like an autoencoder:  $q(\underline{z}|\underline{x})$  encodes  $\underline{x}$  into  $\underline{z}$  while  $p(\underline{x}|\underline{z})$  decodes  $\underline{x}$  from  $\underline{z}$ .
- Since the posterior PDF  $q(\underline{z}|\underline{x})$  is much narrower than the prior PDF  $p(\underline{z})$ , the expectation  $E_{\underline{Z} \sim q(\underline{z}|\underline{x})}(\square)$  can be approximated by the sample mean of  $\square$  by drawing a few samples  $\underline{z}_i$  from  $q(\underline{z}|\underline{x})$ . If  $q(\underline{z}|\underline{x})$  is very narrow, e.g.  $q(\underline{z}|\underline{x}) \approx \delta(\underline{z} - \underline{z}_0)$ ,

$$-E_{\underline{Z} \sim q(\underline{z}|\underline{x})} \ln p(\underline{x}|\underline{Z}) = - \int \ln p(\underline{x}|\underline{z}) q(\underline{z}|\underline{x}) d\underline{z} \approx -\ln p(\underline{x}|\underline{z}_0).$$

This is often done in practice.

- $\underline{Z}$  has a desired known PDF  $p(\underline{z})$ , e.g.  $N(\underline{0}, \mathbf{I})$ .  $q(\underline{z}|\underline{x})$  is unknown, but close to Gaussian  $N(\underline{\mu}(\underline{x}), \mathbf{C}(\underline{x}))$  due to min.  $D_{KL}(q(\underline{z}|\underline{x})||p(\underline{z}))$ . This KL divergence between two Gaussian distributions can be calculated analytically.

The approximation of the difficult-to-calculate integral in  $-\ln p(\underline{x})$  by an easy-to-calculate sample mean and KL divergence is the key benefit of the variational upper bound.

## Application of the variational upper bound to VAE<sup>★</sup>

with some renamed notations  $p \rightarrow q$ ,  $q(\underline{z}|\underline{x}) \rightarrow q_E(\underline{z}|\underline{x})$ :

$$\textcolor{red}{(L(x; \theta)) = -\ln q(\underline{x}; \theta) \leq \underbrace{-E_{\underline{Z} \sim q_E(\underline{z}|\underline{x}; \theta_E)} \ln q(\underline{x}|\underline{Z}; \theta_D)}_{l_{\text{rec}}(\underline{x}; \theta_E, \theta_D)} + \underbrace{D_{\text{KL}}(q_E(\underline{z}|\underline{x}; \theta_E) \| q(\underline{z}))}_{l_{\text{KL}}(\underline{x}; \theta_E)}}$$

- $q(\underline{x}; \theta)$ : The hard-to-calculate parametric model for the unknown true distribution  $p(\underline{x})$  of  $\underline{x}$  of a generative model. Instead of minimizing  $-\ln q(\underline{x}; \theta)$ , the variational upper bound is minimized.
- $q_E(\underline{z}|\underline{x}; \theta_E)$ : The posterior distribution of the latent variable  $\underline{z}$  for a given  $\underline{x}$ . It describes the encoder of VAE with the parameter vector  $\theta_E$ .
- $q(\underline{x}|\underline{z}; \theta_D)$ : How can the original input  $\underline{x}$  be reconstructed from the latent variable  $\underline{z}$ ? It describes the decoder of VAE with the parameter vector  $\theta_D$ .
- $q(\underline{z})$ : The desired prior distribution of the latent variable  $\underline{z}$ .  $\sim \mathcal{N}(\underline{\theta}, \underline{\Sigma})$
- $\min_{\theta} l_{\text{rec}}$ : Train the decoder  $\theta_D$  and encoder  $\theta_E$  for an optimum reconstruction of  $\underline{x}$  from  $\underline{z}$  (as in normal autoencoders).
- $\min_{\theta} l_{\text{KL}}$ : Train the encoder  $\theta_E$  to force  $\underline{z}$  to the desired (Gaussian) distribution  $q(\underline{z})$ .

$$\Rightarrow \min_{\theta} L(\theta) = \frac{1}{N} \sum_{n=1}^N l(\underline{x}(n); \theta)$$

$$l(\underline{x}; \theta) = -\ln q(\underline{x}; \theta) \leq l_{\text{rec}}(\underline{x}; \theta_E; \theta_0) + l_{\text{KL}}(\underline{x}; \theta_E)$$

## 2. Trick

AE : deterministic encoding

encoder output = value of  $\underline{z}$  for a given  $\underline{x}$   
→ replicate the same  $\underline{x}$

VAE : stochastic encoding:

encoder output  $\neq \underline{z}$ , rather,

= mean  $\underline{\mu}(\underline{x})$  and covariance matrix

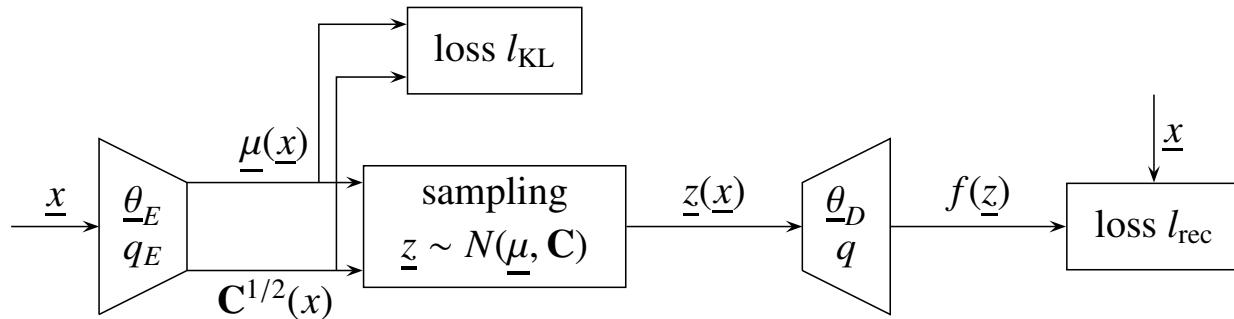
$\underline{\Sigma}(\underline{x})$  of  $\underline{z} \sim \mathcal{N}(\underline{\mu}(\underline{x}), \underline{\Sigma}(\underline{x}))$  to draw

random samples of  $\underline{z}$

→ generate new similar samples

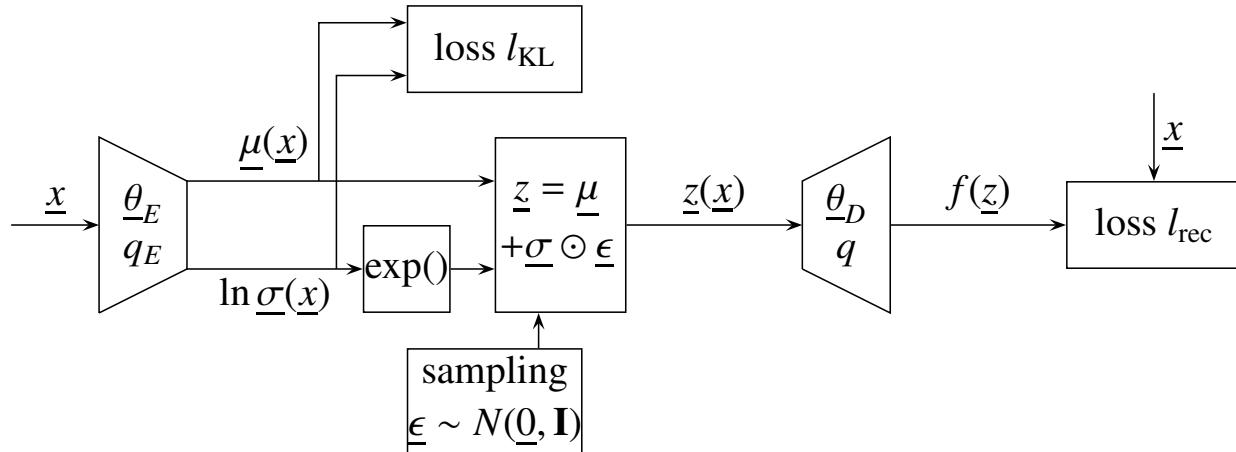
*stochastic encoding*

## VAE without reparametrization ★



- $\mathbf{C}^{1/2}$  is the square root of the covariance matrix  $\mathbf{C} = \mathbf{C}^{T/2} \cdot \mathbf{C}^{1/2}$ .
- Mostly,  $\mathbf{C}$  is assumed to be diagonal  $\text{diag}(\sigma_1^2, \dots, \sigma_d^2)$  with the variances  $\sigma_i^2$  to simplify the training. In this case,  $\mathbf{C}^{1/2} = \text{diag}(\underline{\sigma})$  is diagonal as well with  $\underline{\sigma} = [\sigma_1, \dots, \sigma_d]^T$  containing the standard deviations.
- The forward pass of the above VAE is as usual. Both loss terms  $l_{\text{KL}}$  and  $l_{\text{rec}}$  can be calculated and averaged over all samples  $\underline{x}(n)$ .
- The backpropagation of gradients to learn  $\underline{\theta}_E$  and  $\underline{\theta}_D$  can, however, not pass the sampling unit. The sampling of  $\underline{z} \sim q_E(\underline{z}|\underline{x}) = N(\underline{\mu}(\underline{x}), \mathbf{C}(\underline{x}))$  is a non-continuous operation and has no gradient.

## VAE with reparametrization★

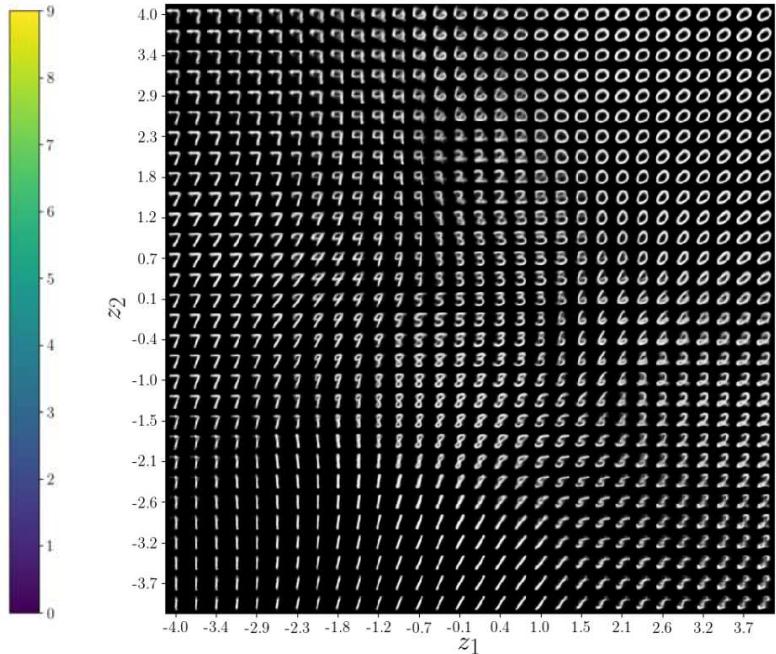
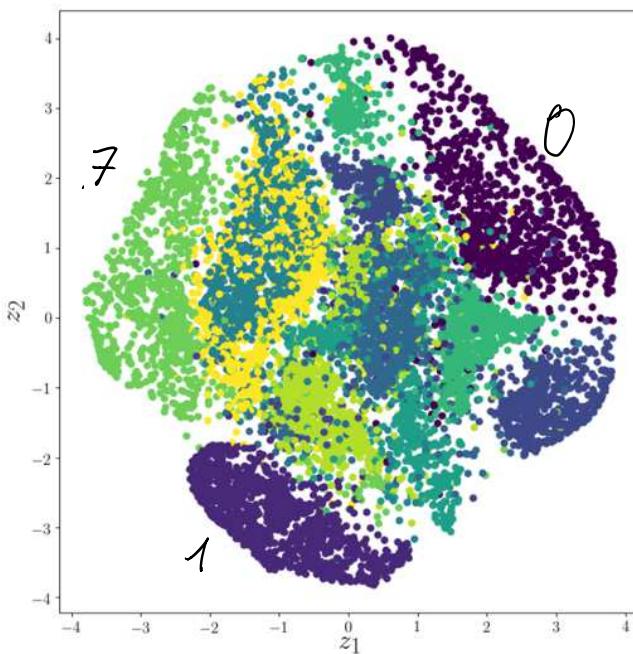


### 3. trick: Reparametrization:

- Instead of drawing a sample  $\underline{z}$  from  $N(\underline{\mu}, \mathbf{C})$ , we draw a sample  $\underline{\epsilon}$  from  $N(\underline{0}, \mathbf{I})$  and calculate  $\underline{z} = \underline{\mu} + \underline{\sigma} \odot \underline{\epsilon}$ . Here a diagonal  $\mathbf{C} = \text{diag}(\underline{\sigma} \odot \underline{\sigma})$  is assumed.
- The sampling  $\underline{\epsilon} \sim N(\underline{0}, \mathbf{I})$  is now outside the path of backpropagation.  $\underline{z}$  is continuous in  $\underline{\mu}$  and  $\underline{\sigma}$  which are continuous in  $\theta_E$ . Backpropagation is thus possible.
- $\underline{\sigma}$  is always non-negative while  $\ln \underline{\sigma}$  can be positive and negative as  $\underline{\mu}$ . It is easier for the encoder to output  $\ln \underline{\sigma}$  than  $\underline{\sigma}$ .

## E9.5: Generation of MNIST digits

- a VAE using a 2D latent space  $\underline{z} = [z_1, z_2]^T \sim N(\underline{0}, \mathbf{I})$
- trained on MNIST dataset
- Left (encoder): distribution of the 10 MNIST digits in the latent space
- Right (decoder): generated 28x28 MNIST digits in the latent space  $(z_1, z_2)$



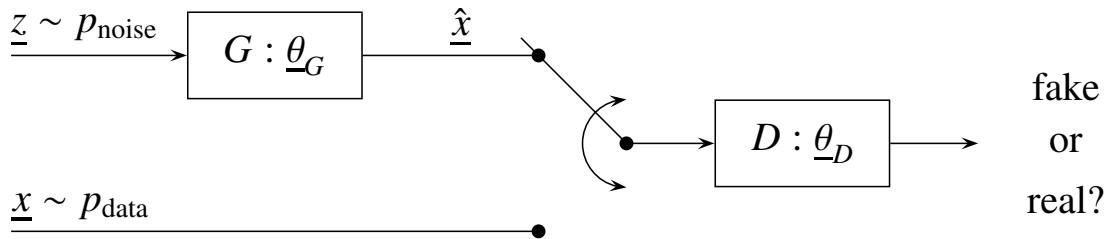
## Basic idea of generative adversarial network (GAN)

illustrated on an example of fake money: A game between two opponents.



- **Generator**  $G$  = forger: Try to make fake money as realistic as possible.
- **Discriminator**  $D$  = police: Try to distinguish between real and fake money as good as possible.
- $G$  has a spy in the police (backpropagation) and learns why can  $D$  discriminate.  $G$  improves his faking technique to further fool  $D$ .
- $D$  tries to find new small differences between real and fake money to avoid to be fooled.
- A continuous competition between  $G$  and  $D$  (called **game theory**).
- At the end: fake moneys are indistinguishable from real ones.

## Structure of a GAN



- $\underline{z}$ : noise sample drawn from a known PDF  $p_{\text{noise}}(\underline{z})$ , e.g.  $N(\underline{0}, \mathbf{I}) \hat{=} \text{latent variable}$
- $G$ : generator (a DNN) with the parameter vector  $\underline{\theta}_G \hat{=} \text{decoder in VAE}$
- $\hat{x} = G(\underline{z}) = G(\underline{z}; \underline{\theta}_G)$ : generated fake sample
- $\underline{x}$ : real sample with the unknown PDF  $p_{\text{data}}(\underline{x})$
- $D$ : binary discriminator/classifier (a DNN) with the parameter vector  $\underline{\theta}_D$ . Its output  $D(\underline{x}) = D(\underline{x}; \underline{\theta}_D) \in [0, 1]$  is the probability of  $\underline{x}$  being real, i.e. class label 1 for real and 0 for fake. Hence,  $D$  uses a sigmoid activation function in the output layer.

## 9.3 Generative adversarial network (GAN)

- generative model, very powerful
- differs from VAE
- „coolest thing in DL“

Intuitions for Training:

$$\begin{aligned} D: & \text{ a) } \underline{x} \sim p_{\text{data}} : \max D(\underline{x}) \\ & \text{ b) } \underline{x} = g(\underline{z}), \underline{z} \sim p_{\text{noise}} : \min D(\underline{x}) \quad \left. \begin{array}{l} \text{adversarial} \\ \vdots \end{array} \right. \\ G: & \text{ c) } \underline{x} = g(\underline{z}), \underline{z} \sim p_{\text{noise}} : \min D(\underline{x}) \quad \left. \begin{array}{l} \text{enemy} \\ \vdots \end{array} \right. \end{aligned}$$

objective function:

$$L(\underline{\theta}_G, \underline{\theta}_D) = E_{\underline{x} \sim p_{\text{data}}} \underbrace{\ln(D(\underline{x}; \underline{\theta}_D))}_{\text{↑ a)}} + E_{\underline{z} \sim p_{\text{noise}}} \ln \underbrace{(1 - D(g(\underline{z}; \underline{\theta}_G), \underline{\theta}_D))}_{\text{↓ b)}}_{\text{↑ c)}}$$

D :

a)

↑

b)

↓

G :

independent of  $\underline{\theta}_G$

c)

↑

Training :

$$\min_{\underline{\theta}_G} \max_{\underline{\theta}_D} L(\underline{\theta}_G, \underline{\theta}_D) \stackrel{!}{=} \text{minimax optimization}$$

$\underline{\theta}_G$      $\underline{\theta}_D$

Why the previous objective function with  $\ln(\cdot)$ ?

Ch. 4.6.2 : Categorical EE for binary classification

$$\min - \left[ \begin{array}{c} ? \\ \text{class label} \end{array} \right] \stackrel{!}{=} \max \left[ \begin{array}{ll} y \ln(f(\underline{x}; \underline{\theta})) & (1-y) \ln(1-f(\underline{x}; \underline{\theta})) \\ y=1 & y=0 \end{array} \right]$$

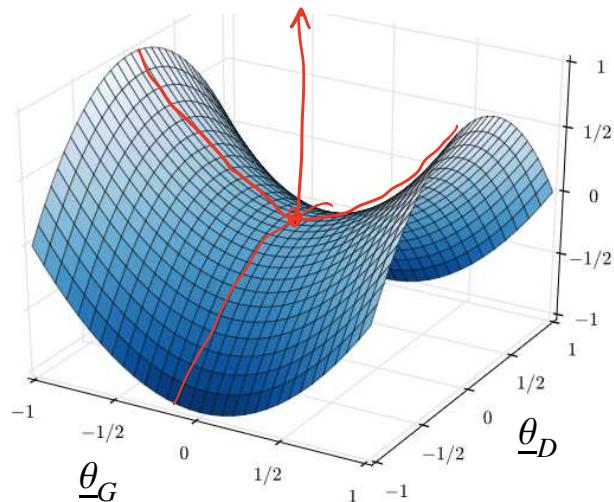
## Training of GAN (1)

One **training step** consists of

- one minibatch for updating  $D$ :  $\max_{\underline{\theta}_D} L(\underline{\theta}_G, \underline{\theta}_D)$ 
  - draw  $B$  samples  $\underline{z}(n)$  from  $p_{\text{noise}}(\underline{z})$
  - select  $B$  samples  $\underline{x}(n)$  from the training set
  - calculate the gradient  $\frac{\partial}{\partial \underline{\theta}_D} \frac{1}{B} \sum_{n=1}^B [\ln D(\underline{x}(n); \underline{\theta}_D) + \ln(1 - D(G(\underline{z}(n); \underline{\theta}_G); \underline{\theta}_D))]$
  - update  $\underline{\theta}_D$  by stochastic gradient **ascent** for a fixed  $\underline{\theta}_G$ :  $\underline{\theta}_D^{t+1} = \underline{\theta}_D^t + \dots$
  
- and one minibatch for updating  $G$ :  $\min_{\underline{\theta}_G} L(\underline{\theta}_G, \underline{\theta}_D)$ 
  - draw  $B$  samples  $\underline{z}(n)$  from  $p_{\text{noise}}(\underline{z})$
  - ~~– select  $B$  samples  $\underline{x}(n)$  from the training set~~
  - calculate the gradient  $\frac{\partial}{\partial \underline{\theta}_G} \frac{1}{B} \sum_{n=1}^B [\ln(1 - D(G(\underline{z}(n); \underline{\theta}_G); \underline{\theta}_D))]$
  - update  $\underline{\theta}_G$  by stochastic gradient **descent** for a fixed  $\underline{\theta}_D$ :  $\underline{\theta}_G^{t+1} = \underline{\theta}_G^t - \dots$

## Training of GAN (2)

- The training of a GAN consists of a sequence of alternating updates of  $\underline{\theta}_D$  and  $\underline{\theta}_G$ .
- Instead of one minibatch for each of  $D$  and  $G$ , an alternation between  $k_D \geq 1$  minibatches for  $\underline{\theta}_D$  and  $k_G \geq 1$  minibatches for  $\underline{\theta}_G$  is possible.
- The minimax optimization is a typical problem from the game theory, a branch of mathematics.<sup>2</sup>
- The solution of minimax is a **Nash equilibrium**, comparable to a saddle point.



<sup>2</sup>The movie "A beautiful mind" tells the story of the mathematician who developed the game theory.

## Training in practice:

◦)  $E(L)$  replaced by  $\frac{1}{B} \sum_{n=1}^B L()$  of a minibatch

◦) alternatively  $\max_{\underline{\theta}_D} L()$  for a fixed  $\underline{\theta}_G$

and

$\min_{\underline{\theta}_G} L()$  for a fixed  $\underline{\theta}_D$

## VAE vs. GAN $\star$

VAE:

- + allows inference  $p(\underline{z}|\underline{x})$  of latent variable  $\underline{z}$  by encoder, i.e. one gets for each  $\underline{x}$  its corresponding hidden code  $\underline{z}$
- low quality of generated samples  $\hat{\underline{x}}$
- approximation error due to variational bound

GAN:

- no calculation of latent variable  $\underline{z}$  because of missing encoder
- + high quality of generated samples  $\hat{\underline{x}}$
- difficult convergence to equilibrium
- **mode collapse**, i.e. the generator generates a limited diversity of samples or even the same sample regardless of the input

### E9.6: Image generation by GAN (1)

a) Generation of realistic photos: Which photo is real and which one is generated by a GAN?<sup>3</sup>

*all AI*

a)



b)



b) More photos

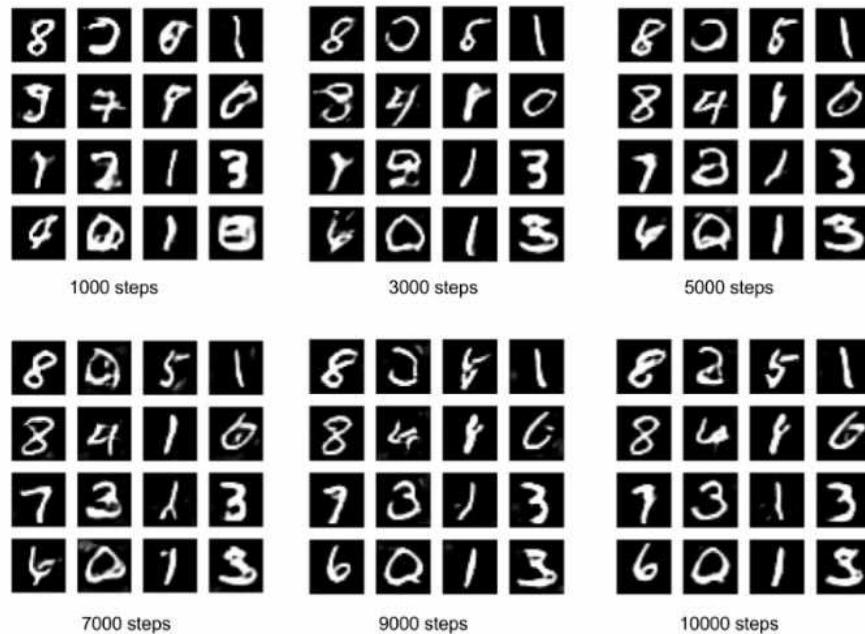
Comments:

- Generation is not copying. There are no real persons like shown. The photos are generated by passing random samples of  $\underline{z}$  to the generator.
- The GAN has learned all characteristics of human faces.

<sup>3</sup>NVIDIA, A style-based generator architecture for generative adversarial networks, arXiv:1812.04948, 2018

**E9.6: Image generation by GAN (2)**

c) Generation of MNIST digit images for a varying number of training steps. These digits are much more realistic (i.e. like hand-written) than those in E9.5 with a VAE.



Application of GAN: generate realistic samples

- data augmentation
- data generation without privacy concern

Limitation of GAN:

noise  $\xrightarrow{\text{GAN}}$   $\hat{x}$ , no control about  $\hat{x}$  except for, realistic e.g. which MNIST digit

Solution: conditional GAN (cGAN)

Application of cGAN:

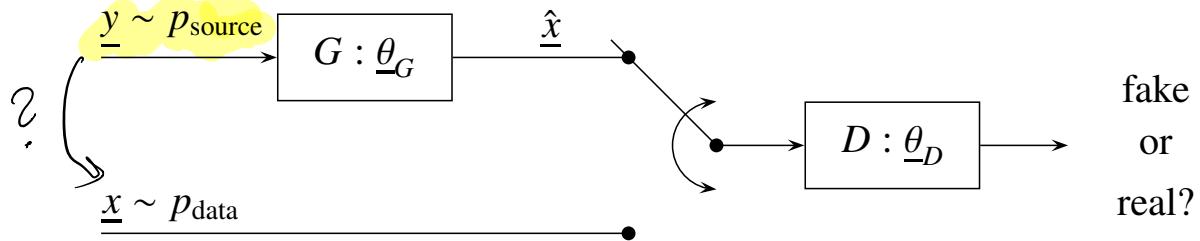
- Signal translation

$y \rightarrow \boxed{\text{cGAN}} \rightarrow \hat{x}$ , close to  $x$  much like

linear filters/systems.

- Any nonlinear mapping
  - mapping/translation is not manually designed, rather learned from examples.
- $\hat{x}$  general nonlinear system identification without any signal model

## Structure of a conditional GAN

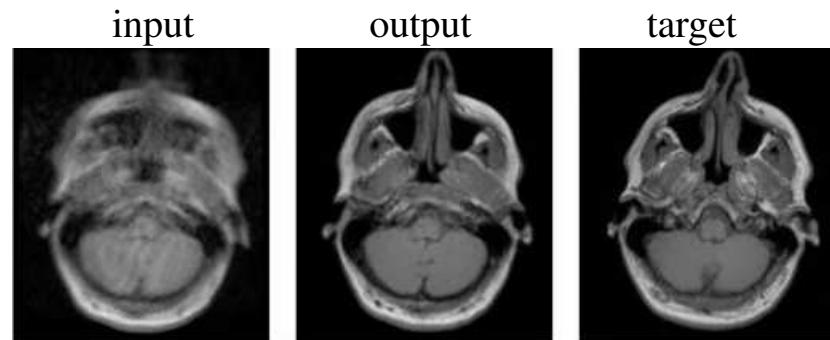


- $\underline{y}$ : source domain sample from the unknown distribution  $p_{\text{source}}(\underline{y})$  instead of a latent variable  $\underline{z}$
- $G$ : generator transforming the input  $\underline{y}$  into the desired target domain
- $\hat{x} = G(\underline{z}) = G(\underline{z}; \underline{\theta}_G)$ : synthetically generated target domain sample
- $\underline{x}$ : true target domain sample with the unknown distribution  $p_{\text{data}}(\underline{x})$
- $D$ : a binary discriminator/classifier as before

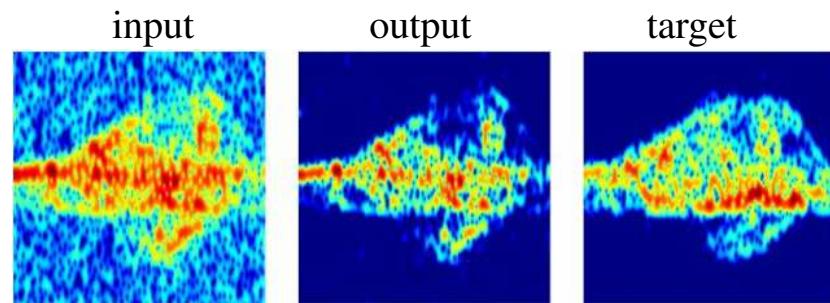
The process of mapping a source domain signal  $\underline{y}$  to a target domain signal  $\hat{x}$  is called **signal-to-signal translation**, or for images, **image-to-image translation**.

### E9.7: Image-to-image translation

a) Motion correction in magnetic resonance (MR) images<sup>4</sup>



b) Denoising in radar spectrograms<sup>5</sup>



+

<sup>4</sup>ISS, Retrospective correction of rigid and non-rigid MR motion artifacts using GANs, arXiv:1809.06276 , 2018

<sup>5</sup>ISS, Towards adversarial denoising of radar micro-doppler signatures, arXiv:1811.04678, 2018

# 10. Popular Networks and Models

Up to now: different types of DNN: dense NN,

CNN, RNN, AE, VAE, GAN

Now:

- popular DNNs
- popular models (building blocks) to design your NN

## 10.1 for Image classification

1) a stack of  $S$  CL's with  $3 \times 3 \times D \times D$  kernels

$$N_{p,1} \approx S \cdot 3^2 \cdot D \text{ parameters}$$

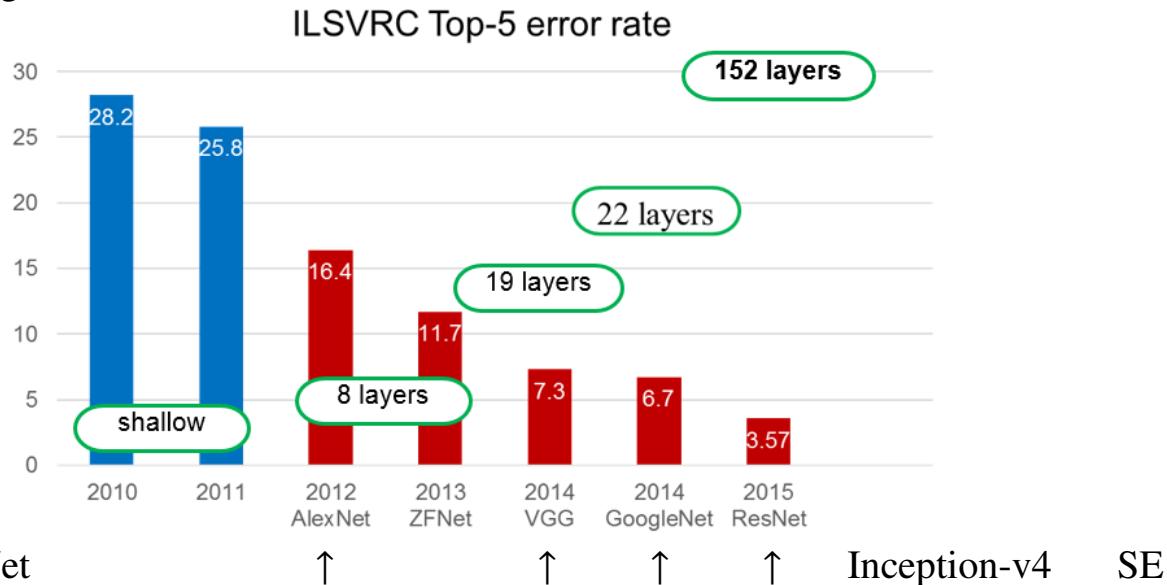
$$\text{effective kernel size } K_{\text{eff}} = 2S+1$$

2) a single CL with  $(2S+1) \times (2S+1) \times D \times D$  kernel

$$\begin{aligned} N_{p,2} &\approx (2S+1)^2 \cdot D^2 \\ \frac{N_{p,1}}{N_{p,2}} &\approx \frac{g_s}{(2S+1)^2} = \frac{S=1}{K_{\text{eff}}} \begin{array}{c|c|c|c} \hline & 2 & 3 & 4 \\ \hline 1 & 72\% & 55\% & 44\% \\ \hline 5 & & 7 & 9 \\ \hline \end{array} \end{aligned}$$

## Popular networks and modules

- For image classification

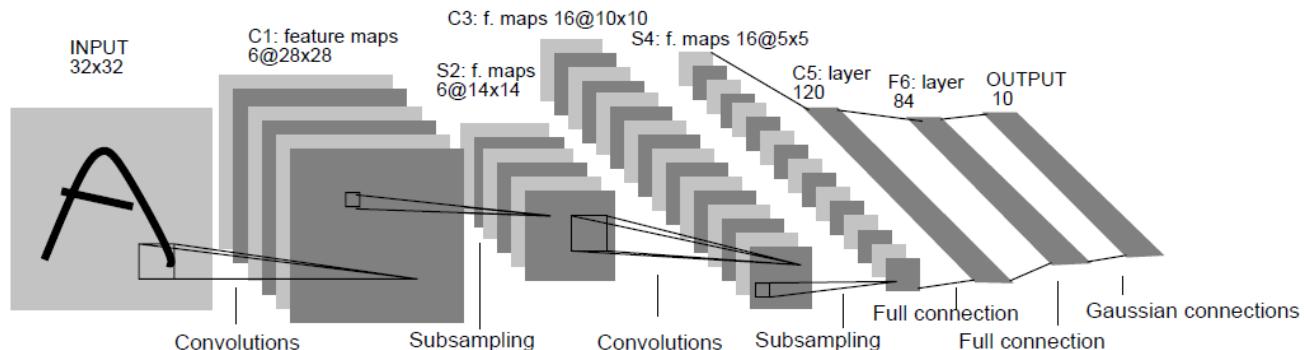


and CBAM, MobileNet, Xception

- For object detection: Faster R-CNN, R-FCN, YOLO, SSD
- For image segmentation: U-Net, V-Net, DeepLabv3+
- For generative tasks: WaveNet

## LeNet (1)

- **LeNet**: Yann LeCun et al (1998) for MNIST digit recognition<sup>1</sup>
- 7 layers, 5 weight layers (3 CL + 2 dense),  $\approx 60k$  parameters, 0.96% error rate
- great impact to today's CNN architecture:
  - convolutional layers, sub-sampling layers, dense layers, output layer
- some unusual design elements (see below)
- *first successful CNN, but for a small task*



<sup>1</sup>Y. LeCun et al, Gradient based learning applied to document recognition, Proc. of the IEEE, 1998

## LeNet (2)★

All convolutional layers (CL): no padding  $P = 0$ , stride  $S = 1$  and dilation  $D = 1$ .

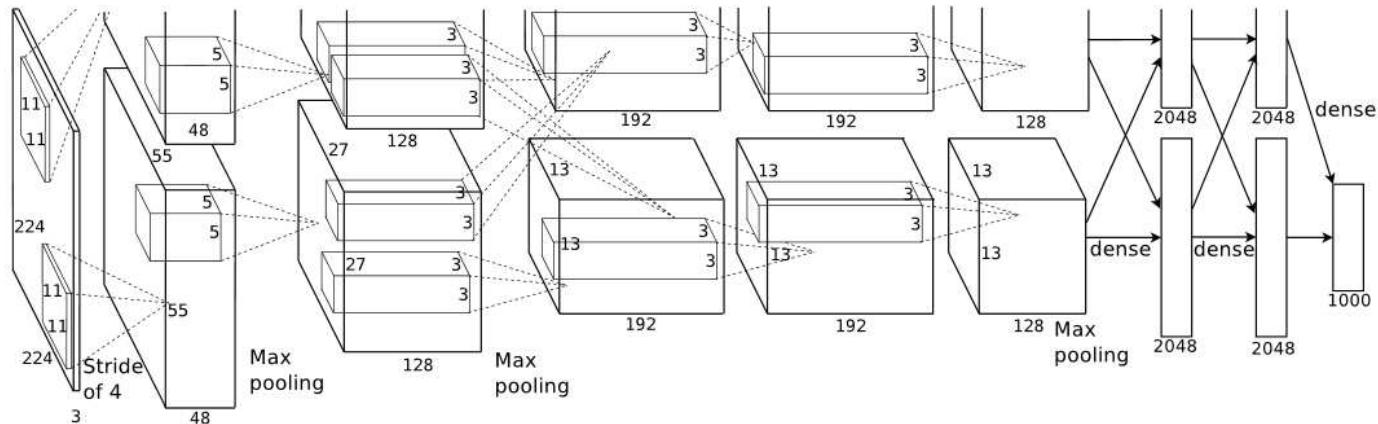
	Layer	Kernel	Output
0	input		32x32x1
1	CL	5x5x1x6	28x28x6
2	sub-sampling	2x2	14x14x6
3	CL	5x5x□x16	10x10x16
4	sub-sampling	2x2	5x5x16
5	CL	5x5x16x120	120
6	dense		84
7	output		10 radial base functions

Unusual design elements:

- 2x2 sub-sampling: not max-pooling, rather sum of 4 pixels and multiplied by a weight as a trainable offset
- □ in 3. layer: sum over 3-6 instead of all 6 input feature maps
- output layer: radial base functions instead of softmax

## AlexNet (1)

- **AlexNet**: Alex Krizhevsky et al from University of Toronto for ILSVRC<sup>2</sup>
- 8 weight layers (5 CL + 3 dense),  $\approx 60M$  parameters, 16.4% top-5 error rate, 9.4% improvement to 2011
- first time ReLU instead of sigmoid, heavy data augmentation, dropout in CL
- *first large-scale CNN, won ILSVRC 2012, beginn of the success story of DNN*



<sup>2</sup>A. Krizhevsky et al, ImageNet classification with deep convolutional neural networks, NIPS, 2012

## AlexNet (2)★

	Layer	Kernel	Output
0	input		224x224x3
1	CL	11x11x3x96, $S=4$	54x54x96
2	max-pooling	2x2	27x27x96
3	CL	5x5x48x256, $P=2$	27x27x256
4	max-pooling	2x2	13x13x256
5	CL	3x3x256x384, $P=1$	13x13x384
6	CL	3x3x192x384, $P=1$	13x13x384
7	CL	3x3x192x256, $P=1$	13x13x256
8	max-pooling	3x3	4x4x256
9	flatten		4096
10	dense		4096
11	dense		4096
12	output		1000-class softmax

16m parameters  
16m parameters  
4m parameters

## VGGNet (1)

- **VGGNet**: Karen Simonyan et al from the Visual Geometry Group at University of Oxford for ILSVRC<sup>3</sup>
- 6 different architectures evaluated. The most famous ones are
  - **VGG16**: 16 weight layers, 138M parameters
  - **VGG19**: 19 weight layers, 144M parameters, 7.3% top-5 error rate
- simple architecture: 3x3 kernels, 2x2 max-pooling + doubled channel depth
- 2. place in ILSVRC 2014
- *Deep narrow kernels better than shallow wide kernels.*

For example, a stack of four 3x3 kernels has the same receptive field as a single 9x9 kernel, but has

- a higher nonlinearity
- less parameters
- better performance than AlexNet

---

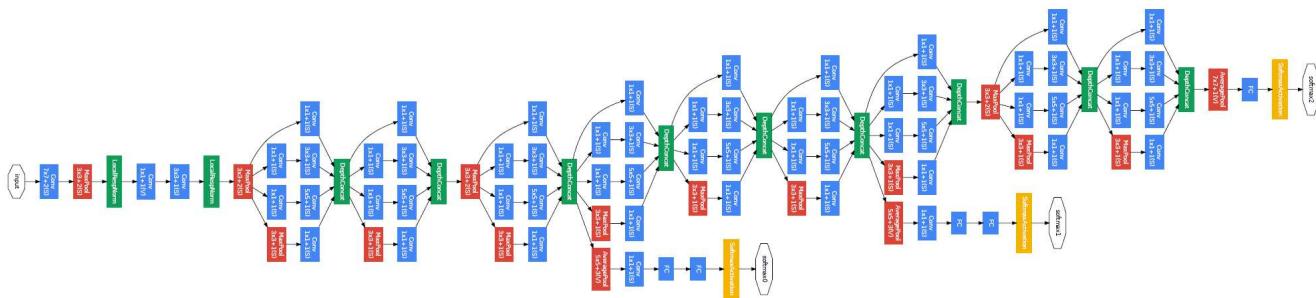
<sup>3</sup>K. Simonyan et al, Very deep convolutional networks for large-scale image recognition, arXiv:1409.1556, 2014

# VGGNet (2)★

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

## GoogLeNet (1)

- **GoogLeNet or Inception-v1**: Christian Szegedy et al from Google for ILSVRC<sup>4</sup>
- 22 weight layers, won ILSVRC 2014, 6.7% top-5 error rate
- only *5M parameters* because of
  - global average pooling (see ch. 7.4) instead of dense layers before softmax
  - stack of 9 Inception modules
- *Inception module*



<sup>4</sup>C. Szegedy et al, Going deeper with convolutions, arXiv:1409.4842, 2014

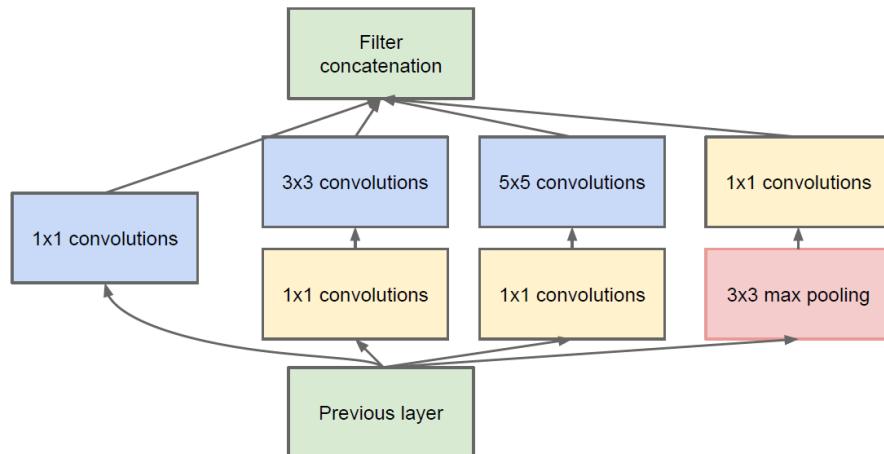
## GoogLeNet (2)

**Inception module** (movie Inception: dream in dream in dream, i.e. deep dream)

- a carefully designed building block
- the first time *parallel feature extraction with different kernel widths* ( $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$ ) from the same input and then concatenation as channels

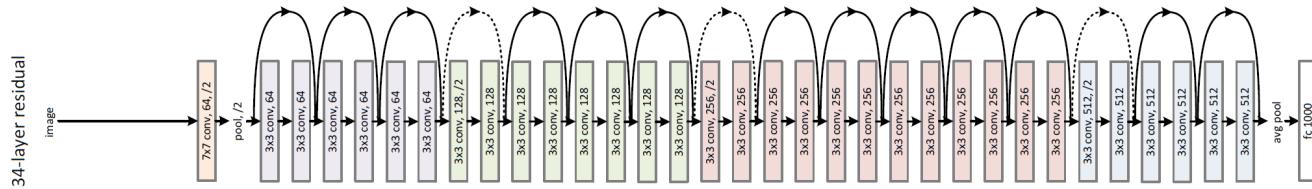
*1x1 convolution* (see ch. 7.2):

- no spatial processing, reduce channel depth and complexity for the next layer
- higher nonlinearity



## ResNet (1)

- **ResNet**: Kaiming He et al from Microsoft for ILSVRC<sup>5</sup>
- 5 different architectures RetNet-18/34/50/101/152 evaluated
- RetNet-152: 152 weight layers, 60M parameters, won ILSVRC 2015
- 3.57% top-5 error rate, *better than human (5%)*
- global average pooling instead of dense layers before softmax
- stack of *residual blocks*
- *To train very deep networks, residual block is a must-have.*



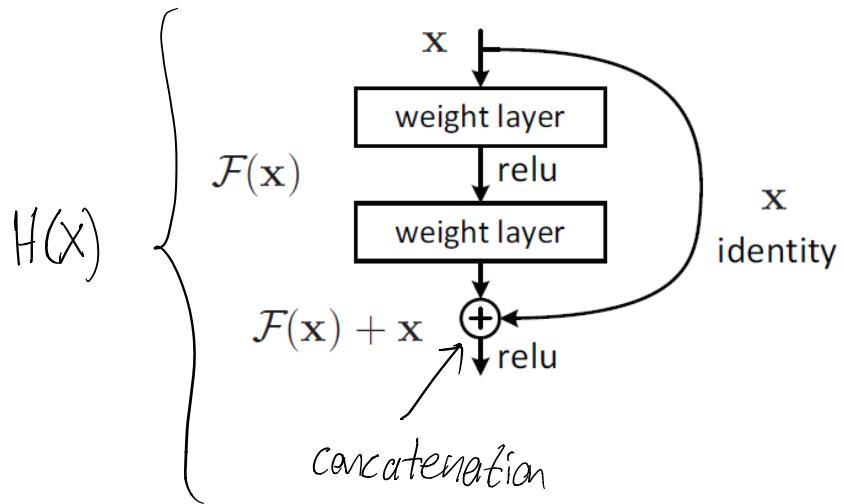
<sup>5</sup>K. He et al, Deep residual learning for image recognition, arXiv:1512.03385, 2015

## ResNet (2)

### Residual block:

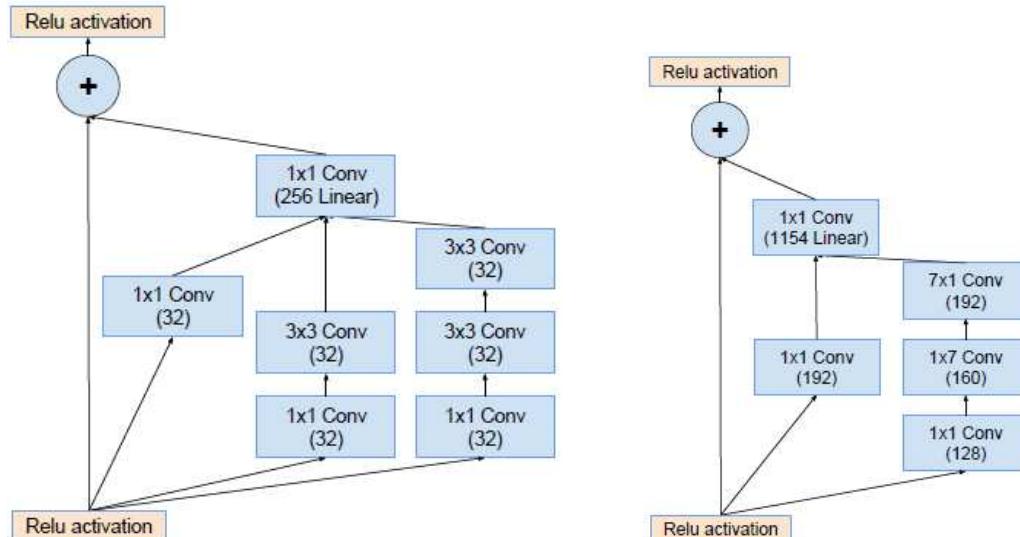
Let  $x \rightarrow H(x)$  be the desired mapping to be learned.  $F(x) = H(x) - x$  is the residual mapping. It is non-trivial to realize the identity mapping  $H(x) = x$  by a stack of nonlinear layers, but it is trivial to realize the corresponding residual mapping  $F(x) = H(x) - x = 0$ .

- Forward pass: Learn the residual mapping  $F(x)$  instead of  $H(x) = F(x) + x$ .
- Backward pass: Shortcut connections help to avoid vanishing gradients.



## Inception-v4

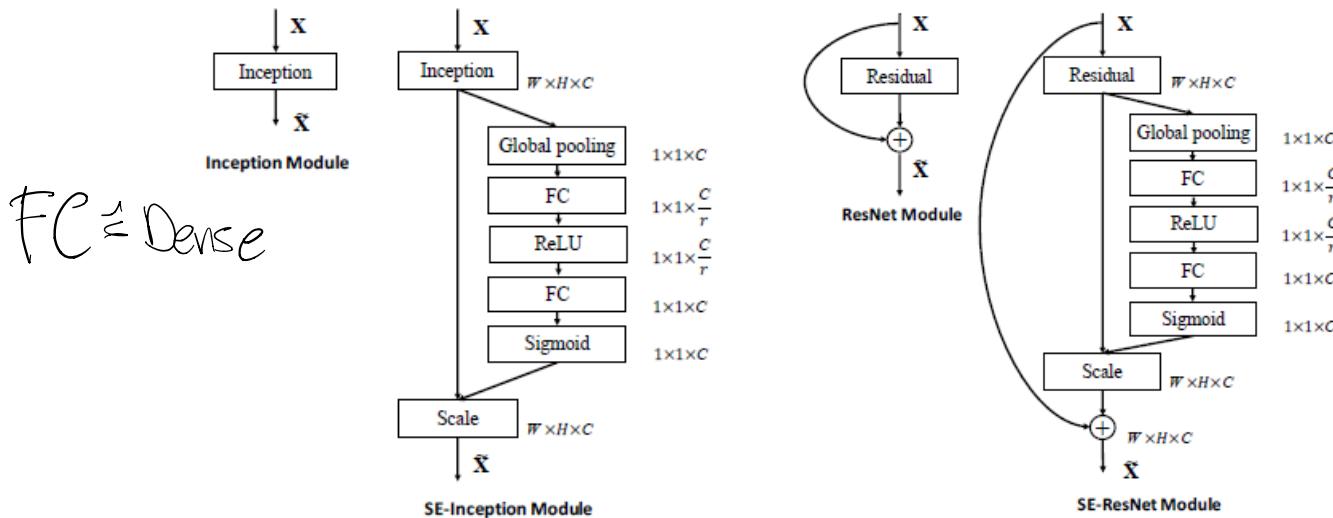
- **Inception-v4 or Inception-ResNet**: Christian Szegedy et al (2016) from Google for ILSVRC<sup>6</sup>
- combine Inception module and residual block
- 3.08% top-5 error rate
- Inception-ResNet-A and Inception-ResNet-B module



<sup>6</sup>C. Szegedy et al, Inception-v4, Inception-ResNet and the impact of residual connections on learning, arXiv:1602.07261, 2016

## Squeeze and Excitation

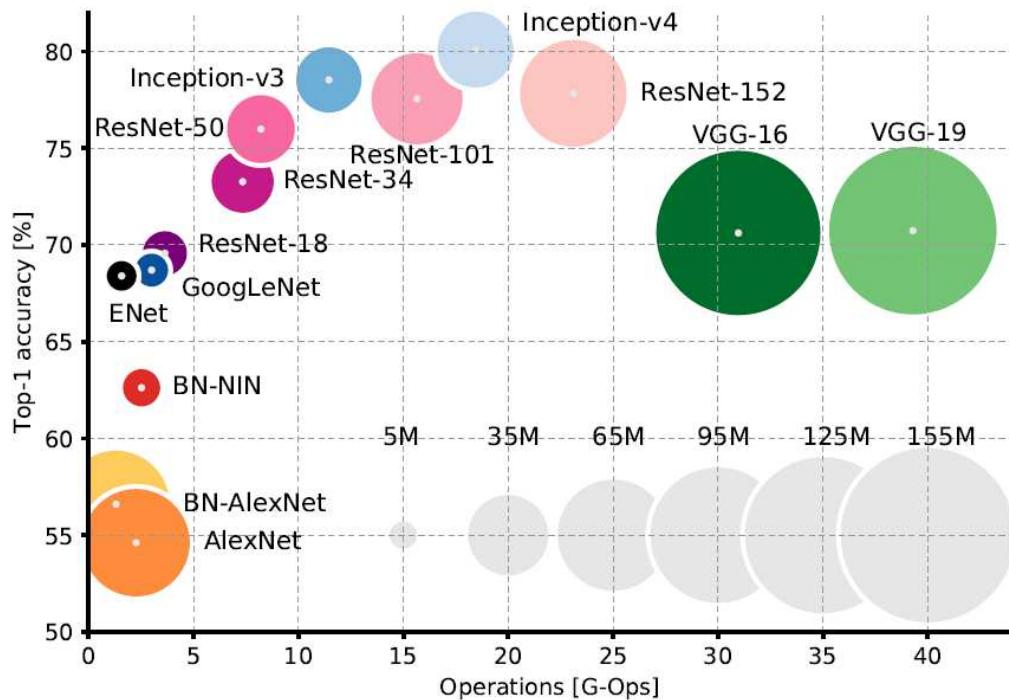
- **Squeeze and Excitation (SE)**: Jie Hu et al for ILSVRC<sup>7</sup>
- **SE block**: Squeeze feature maps to single numbers by global pooling, capture channel dependencies by dense layers and use the results to recalibrate the feature maps. This is also called *channel attention*.
- can be applied to any other modules (e.g. Inception, ResNet)
- 2.25% top-5 error rate, won ILSVRC 2017



<sup>7</sup>J. Hu et al, Squeeze-and-Excitation networks, arXiv:1709.01507, 2017

## Comparison of DNNs for image classification

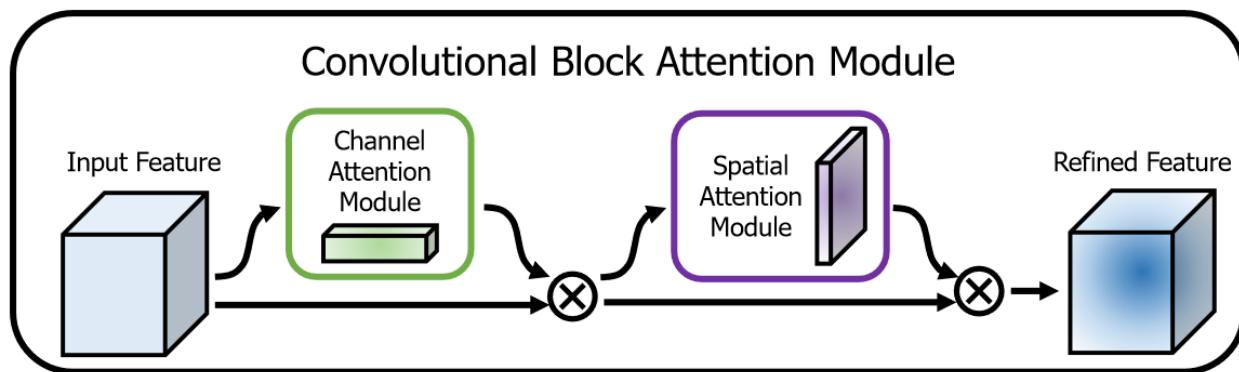
- A. Canziani et al (2016)<sup>8</sup>
- ILSVRC top-1 accuracy vs. memory usage and number of operations for inference



<sup>8</sup>A. Canziani et al, An analysis of deep neural network models for practical applications, arXiv:1605.07678, 2016

## Convolutional block attention module

- **Convolutional block attention module (CBAM)**: S. Woo et al (2018)<sup>9</sup>
- **channel attention**: compute weights  $\in \mathbb{R}^D$  to scale (emphasize) individual channels (feature maps)
- **spatial attention**: compute weights  $\in \mathbb{R}^{M \times N}$  to scale (emphasize) individual pixels and spatial regions
- parameters of both modules are learned from training data



<sup>9</sup>S. Woo et al, CBAM: Convolutional block attention module, arXiv:1807.06521, 2018

## MobileNet

- **MobileNet**: A. G. Howard Woo et al (2017)<sup>10</sup>
- use **depthwise separable convolution** (see ch. 7.2) consisting of
  - depthwise convolution: 2D spatial convolution for each feature map individually
  - pointwise convolution =  $1 \times 1$  convolution
- much less parameters and less operations at the price of a small accuracy loss
- suitable for low-cost mobile/embedded systems
- applicable for all feature extractors in image segmentation, object detection and image segmentation
- now MobileNetv2 and MobileNetv3 available

---

<sup>10</sup>A. G. Howard Woo et al, MobileNets: Efficient convolutional neural networks for mobile vision application, arXiv:1704.04861, 2017

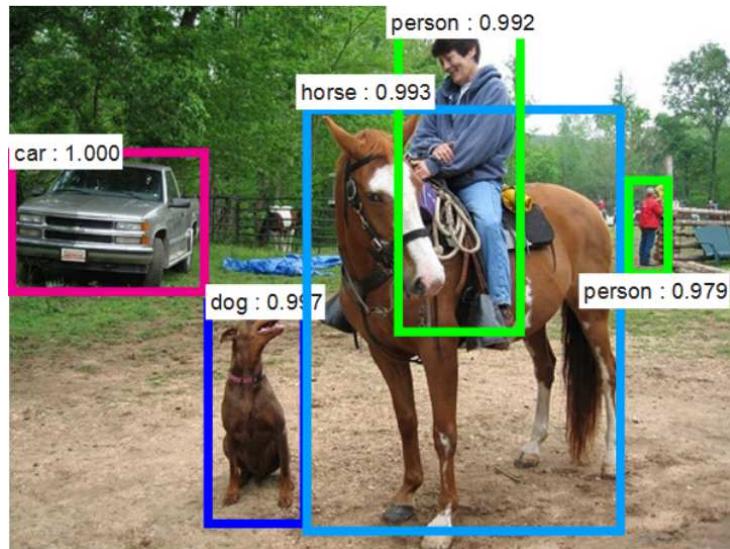
## Xception

- **Xception:** F. Collet from Google (2017)<sup>11</sup>
- combination of Inception and **modified depthwise separable convolution**
  - first pointwise convolution  $1 \times 1 \quad 1D$
  - then depthwise convolution  $2D$

## Object detection

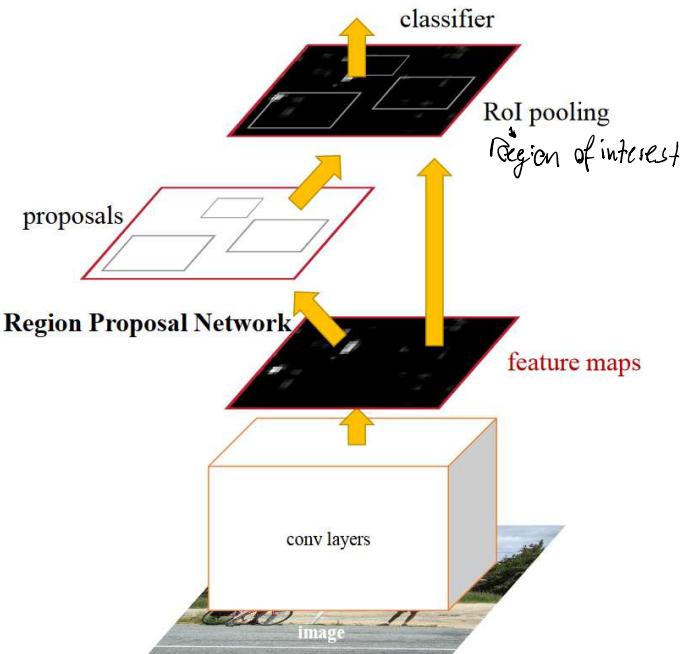
▽  
0

- detect, localize and classify objects in the input image
- a mixture of box position regression and box content classification
- feature extractors from image classification (VGG, Inception, ResNet, MobileNet, ...) used as shallow layers to extract features
- different meta structures for object detection based on the feature maps



## Faster R-CNN<sup>★</sup>

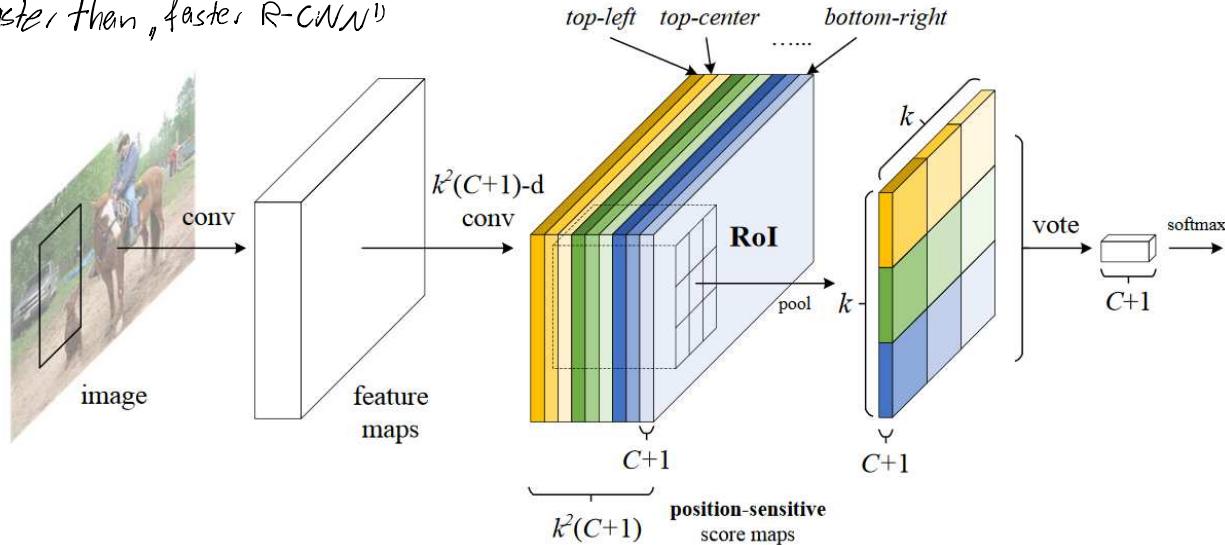
- **Faster R-CNN:** Shaoqing Ren et al (2016)<sup>12</sup>
- a region proposal network (RPN) generates region proposals from the input image
- bounding box estimation and box classification for each proposal in a 2nd step



<sup>12</sup>S. Ren et al, Faster R-CNN: Towards real-time object detection with region proposal networks, arXiv:1506.01497, 2016

# R-FCN<sup>★</sup>

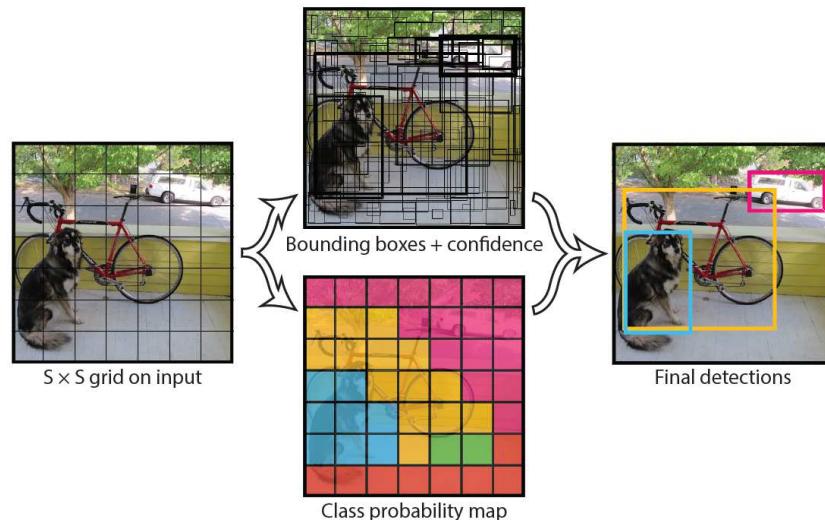
- Region-based fully convolutional networks (**R-FCN**): J. Dai et al (2016)<sup>13</sup>
- $k \times k \times (C + 1)$  position-sensitive score maps for  $C$  objects in a  $k \times k$  grid
- regions of interest (RoI) from a parallel region proposal network
- vote "yes" if enough object parts in a RoI
  - *faster than, faster R-CNN<sup>14</sup>*



<sup>13</sup>J. Dai et al, R-FCN: Object detection via region-based fully convolutional networks, arXiv:1605.06409, 2016

# YOLO<sup>★</sup>

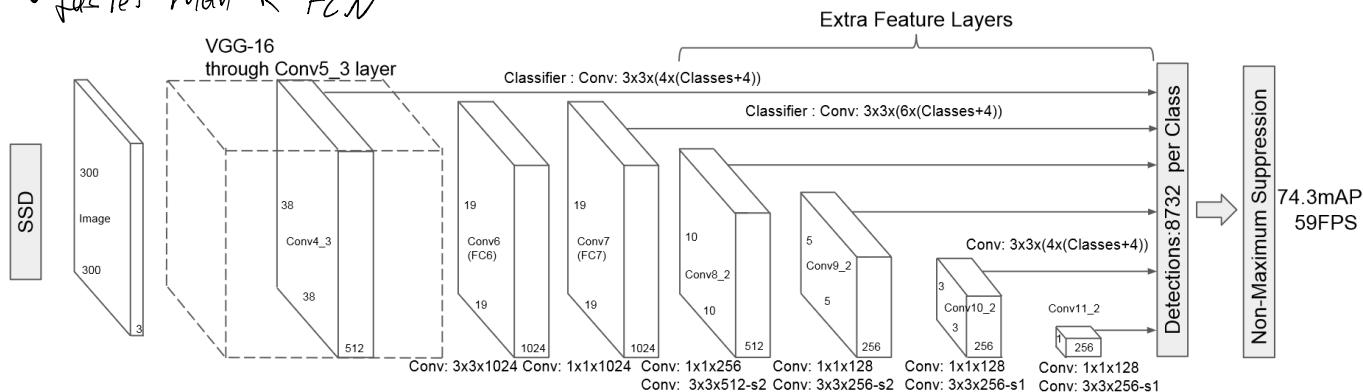
- You only look once (**YOLO**): Joseph Redmon et al (2016)<sup>14</sup>
- divides each input image into grids
- bounding box estimation and box classification in each grid
- delete boxes without objects and merge highly overlapping boxes
- faster than Faster R-CNN
- now YOLOv2, YOLOv3 and YOLO9000 available



<sup>14</sup>J. Redmon et al, You only look once: Unified, real-time object detection, arXiv:1506.02640, 2016

# SSD<sup>★</sup>

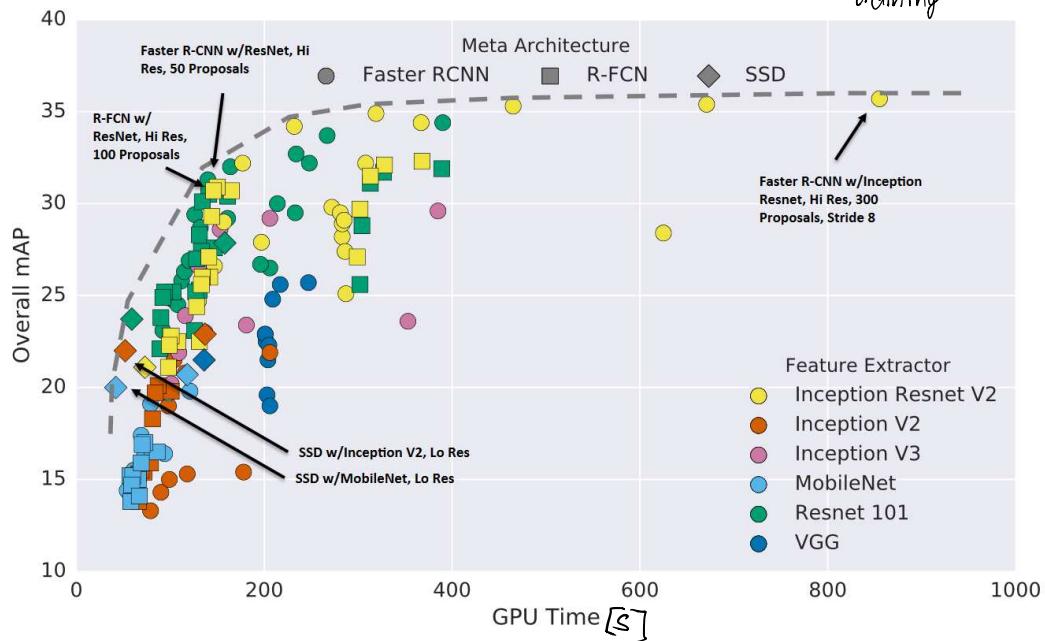
- **Single shot detector (SSD)**: W. Liu et al (2016)<sup>15</sup>
- predict at once all bounding boxes and the class probabilities
- input image passes through multiple convolutional layers (feature extractors)
- feature maps from different layers used to predict small sets of boxes with different aspect ratios
- during training, the box localizations are modified to best match the ground truth
- *faster than R-FCN*



<sup>15</sup>W. Liu et al, SSD: Single shot multibox detector, arXiv:1512.02325, 2016

## Comparison of DNNs for object detection

- J. Huang et al (2017)<sup>16</sup>
- based on the **COCO** Challenge for object detection
- mean average precision (mAP) vs. GPU time for inference → time to prediction after training

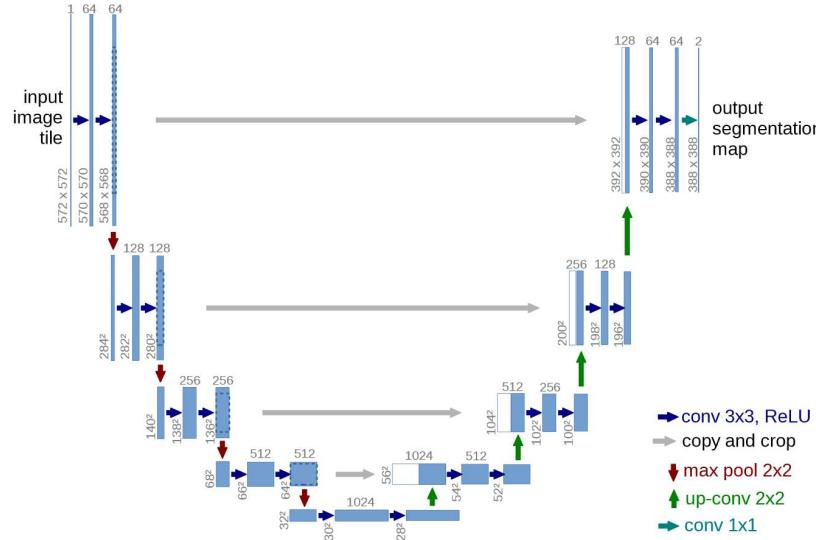


+

<sup>16</sup>J. Huang et al, Speed/accuracy trade-offs for modern convolutional object detectors, arXiv:1611.10012, 2017

## U-Net

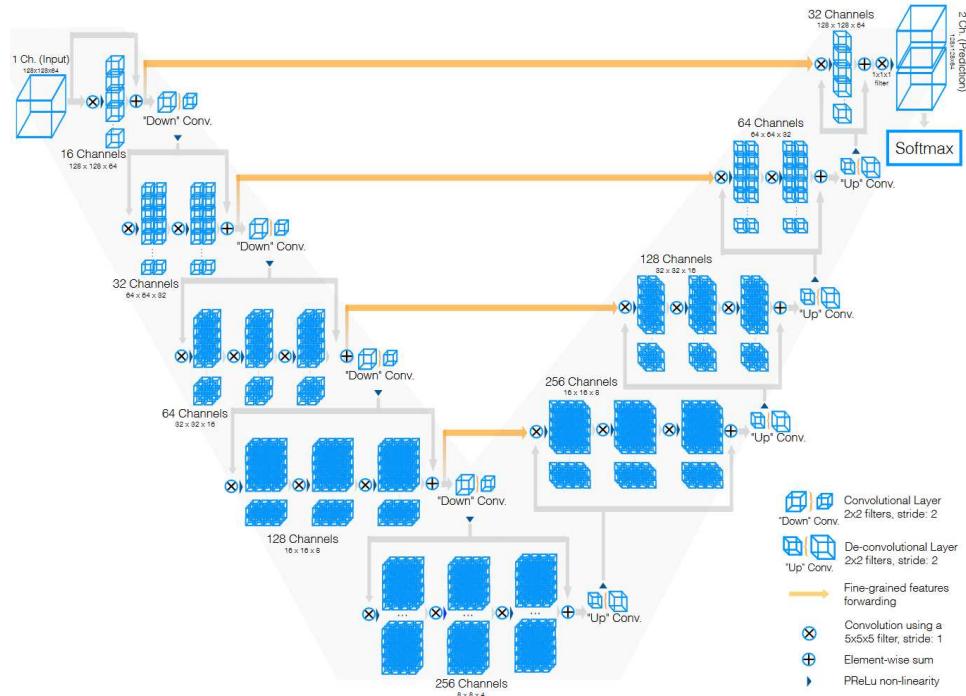
- **U-Net:** Olaf Ronneberger et al (2015) from Uni Freiburg for 2D medical image segmentation<sup>17</sup>
- symmetric arrangement of a downsampling (with increasing channel depth) and an upsampling part (with decreasing channel depth)
- shortcuts between downsampling and upsampling to avoid loss of features



<sup>17</sup>O. Ronneberger et al, U-Net: Convolutional networks for biomedical image segmentation, arXiv:1505.04597, 2015

## V-Net

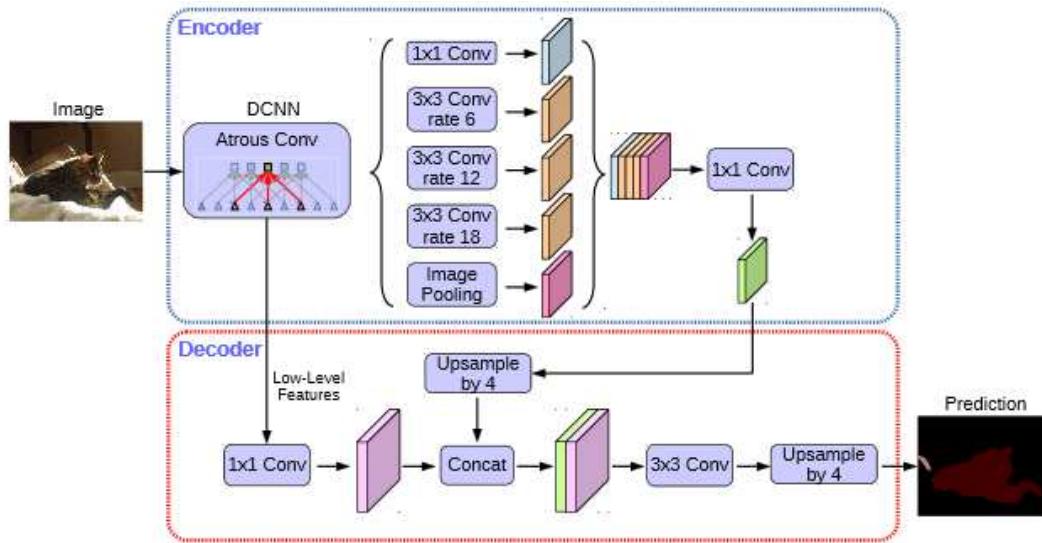
- **V-Net**: Fausto Milletari et al (2016) for 3D medical image segmentation<sup>18</sup>
- similar to U-Net with some differences



<sup>18</sup>F. Milletari et al, V-Net: Fully convolutional neural networks for volumetric medical image segmentation, arXiv:1606.04797, 2016

## DeepLabv3+★

- **DeepLabv3+:** L.-C. Chen et al (2016) for semantic image segmentation<sup>19</sup>
- encoder: a deep CNN for feature learning + atrous spatial pyramid pooling (dilated convolution at different dilations  $D$ )
- decoder: feeded by extracted features to predict the segmented image



+

<sup>19</sup>L.-C. Chen et al, Encoder-decoder with atrous separable convolution for semantic image segmentation, arXiv:1802.02611, 2018

## WaveNet (1)

Traditional **text-to-speech (TTS)** systems for speech synthesis:

- **parametric**: based on a parametric signal model for speech, learn parameters from data, synthesized speech quite unnatural
- **concatenative**: concatenation of recorded speech fragments from one speaker
  - more natural, but still far from human-like naturalness
  - hard to change voice (speaker)

**WaveNet:**

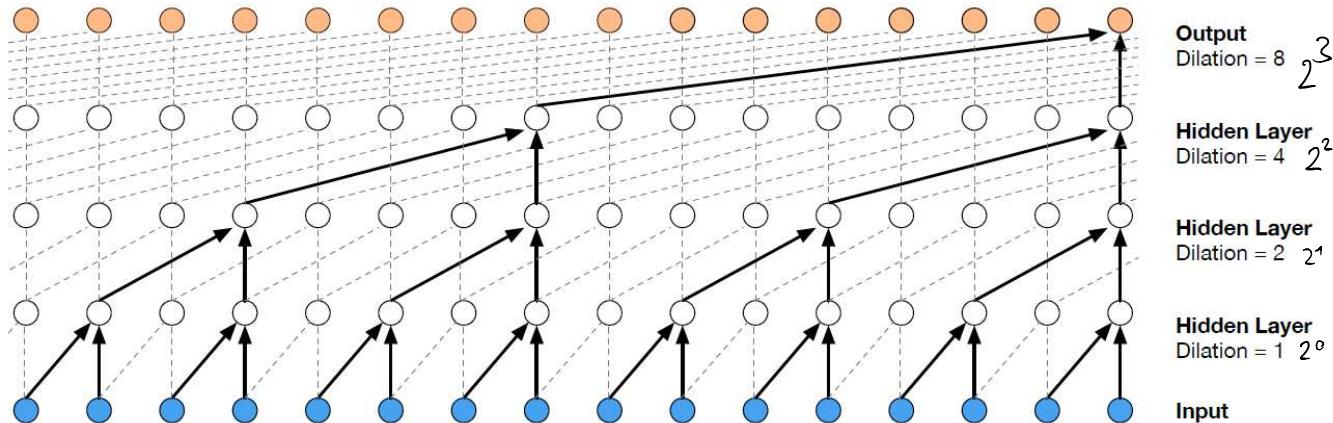
- A. Oord et al (2016) from Google DeepMind<sup>20</sup>
- direct modeling of acoustic waveform
- able to generate any audio signal (speech, music, acoustic events, ...)
- human-like naturalness for generated speech
- today's standard for speech synthesis

---

<sup>20</sup>A. Oord et al, WaveNet: A generative model for raw audio, arXiv:1609.03499, 2016

## WaveNet (2)

- generative model based on CNN instead of RNN/LSTM because of time dependency across more than 10k samples
- predict one sample at a time (from 256 possible values) based on the history
- use a stack of dilated causal 1D convolutional layers to reduce complexity
- **global conditioning** on a vector: affects all generated samples (e.g. speaker, gender, mood, ...)
- **local conditioning** on a time series: embedded linguistic information (e.g. text)



## Important but unaddressed topics

Up to now, only the fundamental concepts of deep learning have been presented. This chapter briefly mentions a large number of important but unaddressed topics:

- Explainable AI
  - visualization
  - causal inference
- Robust AI
  - adversarial attack
- New learning paradigms
  - deep reinforcement learning
  - meta-learning
    - \* transfer learning
    - \* continual learning
    - \* few-shot learning
    - \* neural architecture search
- Embedded implementation
  - model reduction

# Visualization

## Visualization techniques

- improve the understanding of DNN models
- enhances their acceptance by users

Two popular visualization techniques for two different purposes:

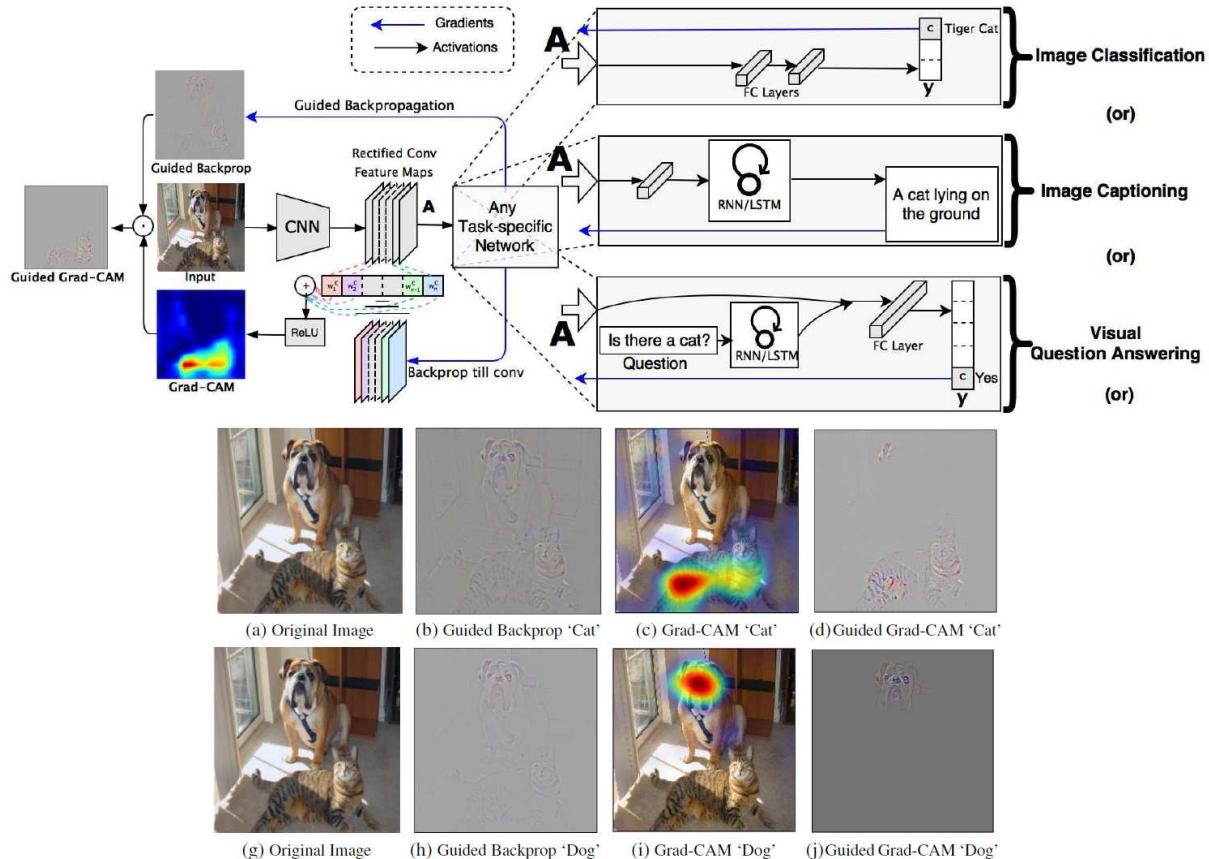
- Gradient-weighted class activation mapping (**Grad-CAM**) or **guided Grad-CAM**:  
R. R. Selvaraju et al (2017)<sup>1</sup>
  - visualize important regions (Grad-CAM) or pixels (guided Grad-CAM) of the input image for a certain target class
  - applicable to all DNNs without retraining
- t-distributed stochastic neighbor embedding (**t-SNE**): L. van der Maaten et al (2008)<sup>2</sup>
  - visualize the similarity of high-dimensional data in a low-dimensional space
  - more powerful than principal component analysis (PCA) due to a nonlinear mapping from high- to low-dimensional space

---

<sup>1</sup>R. R. Selvaraju et al, Grad-CAM: Visual explanations from deep networks via gradient-based localization, IEEE ICCV, 2017

<sup>2</sup>L. van der Maaten et al, Visualizing data using t-SNE, Journal of Machine Learning Research, 2008

# Grad-CAM and guided Grad-CAM (1)



## Grad-CAM and guided Grad-CAM (2)

Grad-CAM:

- Train a DNN for e.g. image classification
- Select the output  $\mathbf{X}_l \in \mathbb{R}^{M_l \times N_l \times D_l}$  of layer  $l$  consisting of  $D_l$  feature maps  $\mathbf{X}_{l,d} \in \mathbb{R}^{M_l \times N_l}$ . Typically, choose a deep layer (large  $l$ ) because a deep layer contains more class information than low-level image features like edges.
- Select the activation  $a_{L,c}$  (before softmax) of the output neuron corresponding to a wanted class  $c$  (e.g. cat).
- Calculate gradients  $\partial a_{L,c} / \partial [\mathbf{X}_l]_{mnd}$  and their average  $\alpha_d^c = \frac{1}{M_l N_l} \sum_m \sum_n \partial a_{L,c} / \partial [\mathbf{X}_l]_{mnd}$  as weighting factors.
- Calculate  $V_c = \text{ReLU} \left( \sum_d \alpha_d^c \mathbf{X}_{l,d} \right)$  as the visualization map for the class  $c$ .
- Normally, a deep layer has a much lower spatial resolution than the input image. Hence upsampling of  $V_c$  to the input resolution and overlay with the input image.

Guided Grad-CAM: Combination of Grad-CAM with **guided backpropagation**<sup>3</sup>

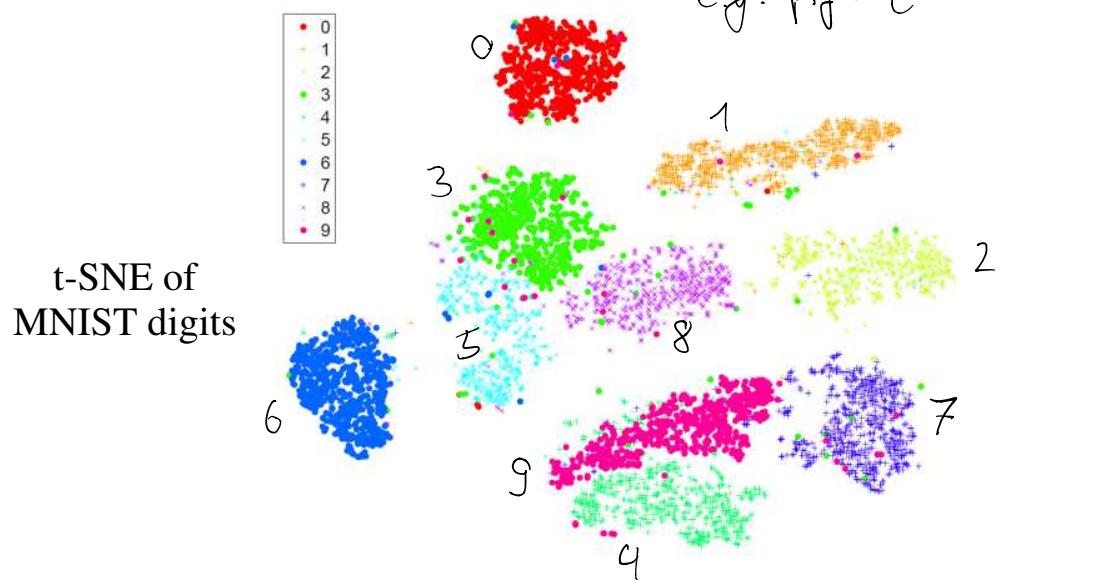
---

<sup>3</sup>J. T. Springenberg et al, Striving for simplicity: The all convolutional net, arXiv:1412.6806, 2014

## t-SNE

An unsupervised and nonlinear dimension reduction technique by representing each high-dimensional object  $\underline{x}_i$  (e.g. a 28x28 digit) by a 2- or 3-dimensional point  $\underline{y}_i$ :

- calculate all pairwise similarity probabilities  $p_{ij}$  between  $\underline{x}_i$  and  $\underline{x}_j$
- define pairwise similarity probabilities  $q_{ij}$  between  $\underline{y}_i$  and  $\underline{y}_j$  in a similar way
- choose  $\underline{y}_i$  to minimize the KLD between  $p_{ij}$  and  $q_{ij}$   
e.g.  $p_{ij} \approx e^{-(\alpha \|\underline{x}_i - \underline{x}_j\|^2)}$



## Causal inference (1)

In ML/DL, we are interested in making a prediction  $\hat{y}$  given an observation  $x$ . This relies on the conditional distribution  $p(y|x)$  which is unknown and estimated from data by using a (DNN) model. We say,  $x$  and  $y$  are (linear or nonlinear) correlated. Correlation is bidirectional:  $x \leftrightarrow y$ .

**Causal inference/reasoning:** Draw a conclusion about the *cause*  $y$  for an observation  $x$ . Causal connection is unidirectional  $x \rightarrow y$ . Correlation does not imply causality!

### E11.1: Correlation vs. causality

- a) Smoking and lung cancer are correlated. Smoking is one cause for lung cancer, but never vice versa: smoking  $\rightarrow$  lung cancer
- b) Once a correlation between yellow finger and early death was reported. Of course there is no causal connection. The common cause for both: yellow finger  $\leftarrow$  smoking  $\rightarrow$  early death
- c) Rain and low barometer value are correlated. One can predict rain from low barometer value and vice versa. But there is no causal connection between them. Again there is one common cause: rain  $\leftarrow$  low pressure  $\rightarrow$  low barometer value

## Causal inference (2)★

Mathematical tools for prediction and causal inference:

- Prediction: Observational  $p(y|x)$ . What is the distribution of  $y$  given observation  $x$ ?
- Causal inference: Interventional  $p(y|\text{do}(x))$  where  $\text{do}$  is the **do-operator**. What is the distribution of  $y$  if I intervened in the natural data generating process by artificially forcing  $X$  to take value  $x$ .

Which one do I want? Depending on the application.  $p(y|x)$  for observation and prediction,  $p(y|\text{do}(x))$  for proactive change and treatment.

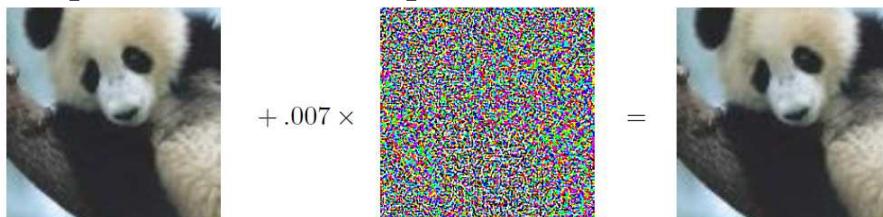
### E11.2: Causal inference "what if"

- a) "Would I not get lung cancer had I not smoked?"
- b) "Would my blood pressure be lower had I consumed less salt?"
- c)  $x = \text{medical treatment}$ ,  $y = \text{outcome}$
- d) software debugging
- e) how to prevent global warming?

We are at the beginning of research for causal inference with DL.

## Adversarial attack (1)

DNNs are powerful, but suffer from **adversarial attacks**: A small, imperceivable, but carefully designed perturbation to the input can fool a trained model.<sup>4</sup>



"panda" 57.7% confidence      noise      "gibbon" 99.3% confidence

- not all perturbations will fool the model, only carefully calculated ones
- imperceivable because the attacks are not visible for humans (more dangerous)
- attacks also to speech, audio, text, ...

How to calculate adversarial examples? It's an optimization problem.

Assume that a model  $f(\cdot)$  is trained to make the correct decision  $f(\underline{x}) = y$ . We look for a small perturbation  $\Delta \underline{x}$  such that  $f(\underline{x} + \Delta \underline{x}) = y' \neq y$ . This is formulated as a minimization problem and solved by gradient descent, using gradients over the input  $\underline{x}$  (by fixing the model), not over the model parameters as during training.

<sup>4</sup>I. Goodfellow et al, Explaining and harnessing adversarial examples, ICLR. 2015

## Adversarial attack (2)

Different types of adversarial attacks:

- **targeted attack**:  $f(\underline{x} + \Delta\underline{x}) = y'$  for a chosen  $y' \neq y$
- **untargeted attack**:  $f(\underline{x} + \Delta\underline{x}) = y'$  for any  $y' \neq y$
- **white-box attack**: model  $f(\cdot)$  well known and gradient search is applied
- **black-box attack**: more challenging due to unknown model  $f(\cdot)$ . No gradient information to guide the search of  $\Delta\underline{x}$ . Instead, multiple queries of  $f(\underline{x})$  are needed.

Intuitions for adversarial attack:

- DNNs have a huge-dimensional feature space
- most of the training data concentrated in a small region called manifold
- input perturbation may move the model to a region the model has never seen before with unpredictable results

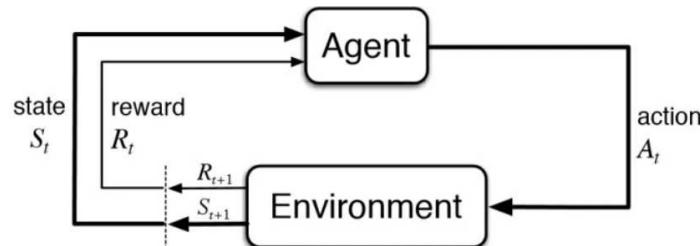
How to defend against adversarial attacks?

- some ideas, but no effective countermeasures yet
- open research problem

## Deep reinforcement learning

Traditional learning methods learn from examples, e.g. AlphaGo learns from old Go games of humans.

**Reinforcement learning (RL)** is a subarea of ML. It learns from **rewards** for sequential decision making. For example, AlphaGo Zero learns from games against computer (AlphaGo). To be more precise, based on the current state (game situation) and reward of an environment (Go game), an agent (AlphaGo Zero) optimizes his next action (which move?) to maximize the long-term reward (win of the game). RL can be interpreted as a data-driven control. It outperforms traditional model-based control in the lack of a good system model.



**Deep reinforcement learning (DRL)** combines RL with deep neural networks.

## Meta-learning

**Meta-learning:** Learn to learn<sup>5</sup>

- When we humans learn new skills, we start from skills learned earlier in related tasks. With every skill learned, learning new skills becomes easier and requires fewer examples.
- How to do that for DNNs? Currently, learn one model for one task from scratch using much training data.

Two major steps:

- Collect **meta-data** describing prior learning tasks and previously learned models (e.g. architecture, model parameters, hyperparameters, performance)
- Extract knowledge from meta-data to guide the search for new models for new tasks with less training data.

There are different related directions of meta-learning: transfer learning, continual learning, few-shot learning, neural architecture search, ....

---

<sup>5</sup>J. Vanschoren, Meta-Learning: A Survey, arXiv:1810.03548, 2018

## Transfer learning and continual learning

**Transfer learning:** Adapt a trained DNN for task A to a similar task B with little additional training data. The new model only solves task B.

**Continual learning:** Learn a sequence of similar or different tasks A, B, ... without forgetting the old tasks. The new model is able to solve all tasks. This corresponds to a constant expansion of skills like humans.

### E11.3: Delta Learning for automated driving

Transfer a DNN for

- $\Delta$ -sensor: from an old sensor to a new one (e.g. new resolution, field of view)
- $\Delta$ -country: from Germany to other countries (e.g. USA, China)
- $\Delta$ -environment: to new weather conditions (e.g. snow, fog)
- $\Delta$ -time: to new traffic situations (e.g. e-scooter)
- ...

## Few-shot learning

**Few-shot learning:** Train a DNN using only a few training examples, given prior experience with very similar tasks.

**One-shot learning:** Using only one training example.

Some key ideas for doing this:

- Learn a common feature space shared by all tasks. This facilitates a fast training for a new task by using a better model initialization and guided optimization of the model parameters.
- Use a memory component to store learned examples in a suitable feature space. Matching each new sample to the memorized ones is easier than learning many model parameters.

## Neural architecture search

Conventionally, the architecture of a DNN is hand-designed. Training means the optimization of the model parameters (and some hyperparameters).

The focus of **neural architecture search (NAS)** is an automated optimization of the DNN architecture. Three basic issues are:

- Search space: It defines the types of basic building blocks and connections to compose the DNN. The use of prior knowledge about typical properties of basic building blocks and connections will help to reduce the search space.
- Search strategy: It deals with exploration of the huge discrete search space. A challenge is the compromise between exploration (quick global search) and exploitation (detailed local search).
- Performance estimation strategy: It refers to the evaluation of a selected architecture. The simplest choice, a standard training and validation of the architecture, is too time-consuming. Methods to reduce the performance estimation cost are highly desirable.

## Model reduction

Today there is no theory to predict the right DNN size and capability for a given task. In order to solve the task with a satisfactory performance, typically DNNs with a larger capacity than required are used.

For an efficient implementation of trained DNNs on low-cost platforms in mobile (e.g. mobile phone) and embedded applications (e.g. car), it is highly desirable to reduce the computational and memory complexity as well as power consumption of a trained DNN without remarkable performance loss. This process is called **model reduction**.

Three important approaches for model reduction are:

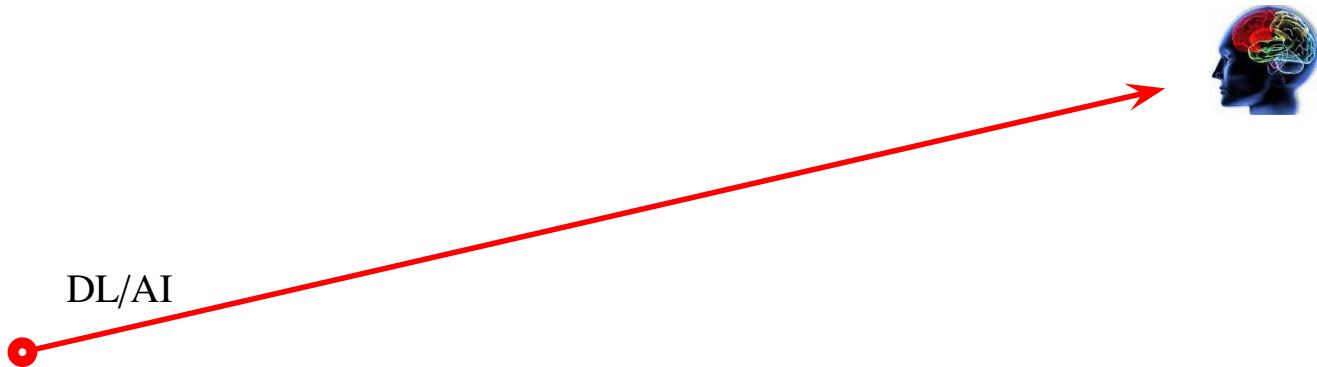
- **Low-rank factorization**: Factorize a large weight matrix  $\mathbf{W} \in \mathbb{R}^{M \times N}$  of a layer into a product of two low-rank matrices  $\mathbf{W} \approx \mathbf{AB}$  with  $\mathbf{A} \in \mathbb{R}^{M \times K}$ ,  $\mathbf{B} \in \mathbb{R}^{K \times N}$ ,  $K < \min(M, N)$ . The number of multiplications changes from  $MN$  to  $(M + N)K$ .
- **Pruning**: Force approximately zero columns or rows of  $\mathbf{W}$  to remove the corresponding input or output neurons.
- **Quantization**: Reduce word length of DNN for inference.

## Is AI already smarter than humans?

AI already beats humans in some tasks:

- Deep Blue for playing chess, IBM 1996
- Watson for question-answering (Jeopardy), IBM 2011
- ResNet for image recognition, Microsoft 2015
- AlphaGo for playing Go, DeepMind 2015-2017
- ...

*No, we are still at the beginning of a long way toward a human-like intelligence.*



## Artificial neural network vs. human brain

	Artificial neural network	Human brain
power consumption	kW–MW	W
synchronization	synchronized	non-synchronized
sampling rate	GHz	1-100Hz firing rate
precision	high	low (few bits)
number of neurons	millions	≈ 100 billions
architecture	layered	more general
network	fixed	evolutionary
learning	gradient-based	unknown
signal transport	electrical	biochemical electrical
intelligence	single domain	multi domains

Huge gap between artificial neural network and human brain!

## Limitations of AI

- Not energy efficient
- Not sample efficient
- It is a **weak/narrow AI**: Achieves or outperforms human's capability in some restricted tasks



**Strong AI**: Achieves or outperforms human's capability in all tasks.

*But we are still far from that.*

## Why weak AI?

- Single domain: AI not successful in one model for different tasks (e.g. speech recognition + automated driving + playing Go).
- Complete information: AI excellent in prepared knowledge, but weak common sense database. Not successful in uncertainty.
- Stationarity: AI not successful in time-varying environments
- Weak logical deduction

Q: "Who is the president of USA?"

A: "Trump."

Q: "Is Trump an American?"

A: "Sorry, I don't know."

- No symbolic processing: A child can distinguish a horse and an elephant by a symbolic comparison of their legs: "A horse has 4 long and thin legs, and an elephant has 4 short and thick legs" without a precise definition of "long/short" and "thin/thick". This is difficult for computers. They are numerical in nature.
- ...

## Which science fiction machines do represent AI?

Survey of Society Computer Science Germany (Gesellschaft Informatik), 2019



### Das sind die bekanntesten Science Fiction-Maschinen mit Künstlicher Intelligenz in Deutschland

■ Prozent der Bundesbürger ab 16 Jahre, die folgende Maschinen zumindest dem Namen nach kennen.



Allensbach-Umfrage im Auftrag der Gesellschaft für Informatik e.V.  
Basis: Bundesrepublik Deutschland, Bevölkerung ab 16 Jahre  
Quelle: Allensbacher Archiv, IfD-Umfrage 12003

Eine Initiative des Bundesministeriums  
für Bildung und Forschung

Wissenschaftsjahr 2019

**KÜNSTLICHE  
INTELLIGENZ**



Future of AI? Social impacts of AI?

## More serious questions

- **Singularity in AI:** Machines achieve human-like intelligence (strong AI).
  - Shall we accept it?
  - Shall we shut down all machines?
  - Can we still shut down the machines?
  - Shall we stop DL/AI research today?

This is not a technical, rather an open social and ethnic issue.

My opinions:

- *There is no way to stop the invention of a new technology* (e.g. fire, dynamite, automobile, nuclear energy, Internet, gene technology, AI, ...).
- *But we must regularize the use of new technologies.*

## My conclusions

- Today's AI is still far away from human-like intelligence.
- But it is a powerful tool for solving many challenging problems.
- Nevertheless, it is not the only tool. It is not recommended for problems with good signal models. In these cases, model-based signal processing without training is simpler and cheaper.
- The AI research will remain relevant for a long time.
  - More and more applications.
  - Increasing basic research to understand DL/AI.
  - Efforts toward strong AI.
- Next big step: brain-inspired **bioelectrical AI**

— Thank you —