Introduction to Neural Networks

Gitta Kutyniok

(Technische Universität Berlin and University of Tromsø)

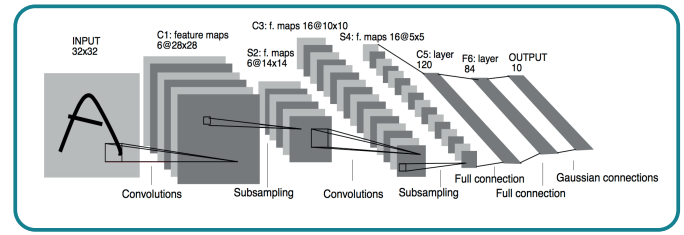
Banach Center – Oberwolfach Graduate Seminar:
Mathematics of Deep Learning
Polish Academy of Sciences, Będlewo, November 17 – 23, 2019

Let's start with some highlights...

The Dawn of Deep Learning

Self-Driving Cars

Surveillance



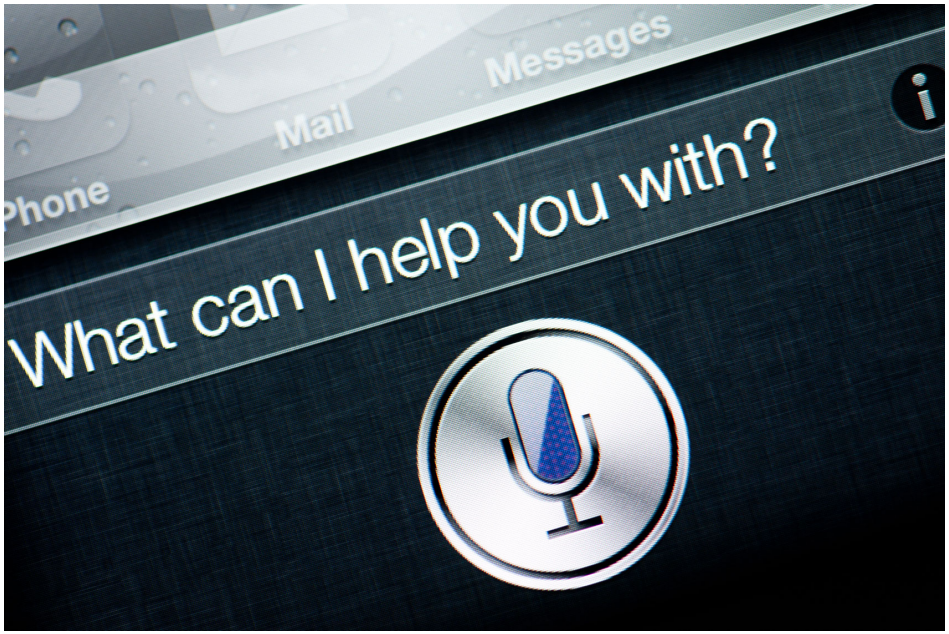
Legal Issues

Health Care

Speech Recognition

Application of deep learning to cell phones:

Speech recognition systems such as Siri belong to the current standard in numerous cell phones!



Deep Learning = Artificial Intelligence?

AlphaGo Zero Shows Machines Can Become Superhuman Without Any Help

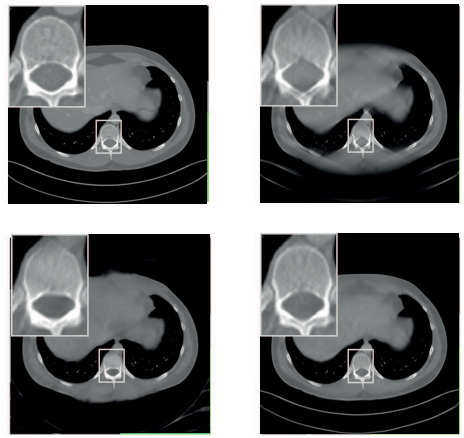
„AlphaGo wasn't the best Go player on the planet for very long. A new version of the masterful AI program has emerged, and it's a monster. In a head-to-head matchup, AlphaGo Zero defeated the original program by 100 games to none...
AlphaGo Zero...started with nothing but a blank board and the rules of the game. It learned simply by playing millions of games against itself, using what it learned in each game to improve."

MIT Technology Review (Oct. 2017)

Impact of Deep Learning on Mathematics

Some Examples:

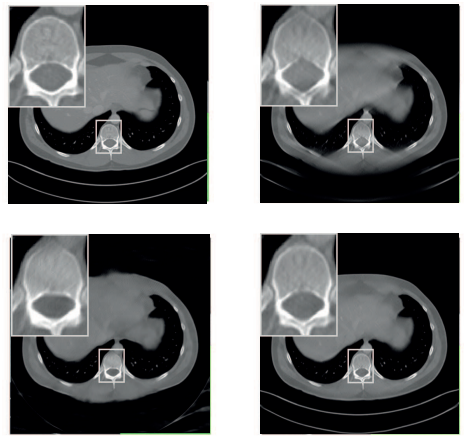
- Inverse Probleme/Imaging Science (2012–)
  - $\rightsquigarrow$ *Denoising*
  - $\rightsquigarrow$ *Edge Detection*
  - $\rightsquigarrow$ *Inpainting*
  - $\rightsquigarrow$ *Classification*
  - $\rightsquigarrow$ *Superresolution*
  - $\rightsquigarrow$ *Limited-Angle Computed Tomography*
  - $\rightsquigarrow$ ...



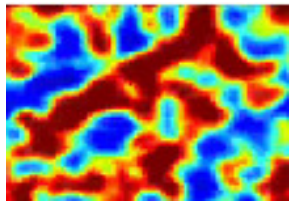
Impact of Deep Learning on Mathematics

Some Examples:

- Inverse Probleme/Imaging Science (2012–)
  - ⇝ *Denoising*
  - ⇝ *Edge Detection*
  - ⇝ *Inpainting*
  - ⇝ *Classification*
  - ⇝ *Superresolution*
  - ⇝ *Limited-Angle Computed Tomography*
  - ⇝ ...



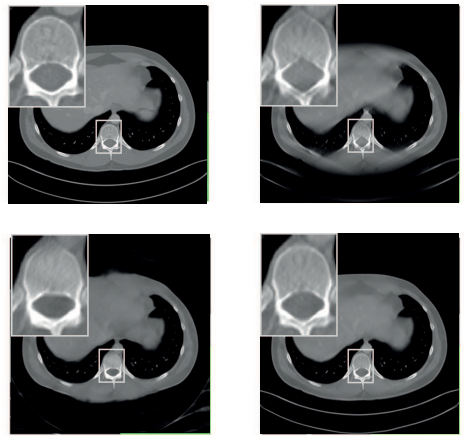
- Numerical Analysis of Partial Differential Equations (2017–)
  - ⇝ *Black-Scholes PDE*
  - ⇝ *Allen-Cahn PDE*
  - ⇝ *Parametric PDEs*
  - ⇝ ...



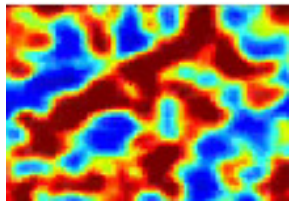
Impact of Deep Learning on Mathematics

Some Examples:

- Inverse Probleme/Imaging Science (2012–)
  - ⇝ *Denoising*
  - ⇝ *Edge Detection*
  - ⇝ *Inpainting*
  - ⇝ *Classification*
  - ⇝ *Superresolution*
  - ⇝ *Limited-Angle Computed Tomography*
  - ⇝ ...



- Numerical Analysis of Partial Differential Equations (2017–)
  - ⇝ *Black-Scholes PDE*
  - ⇝ *Allen-Cahn PDE*
  - ⇝ *Parametric PDEs*
  - ⇝ ...



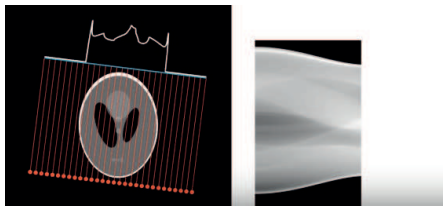
- Modelling (2018–)
  - ⇝ *Learning of equations from data*
  - ⇝ *Learning of PDEs*

Examples: Deep Learning changes Mathematical Methods

Limited-Angle Computed Tomography

A CT scanner samples the *Radon transform*

$$\mathcal{R}f(\phi, s) = \int_{L(\phi,s)} f(x) dS(x),$$



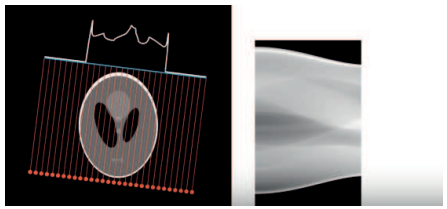
for $L(\phi, s) = \left\{x \in \mathbb{R}^2 : x_1 \cos(\phi) + x_2 \sin(\phi) = s\right\}$, $\phi \in [-\pi/2, \pi/2)$, and $s \in \mathbb{R}$.

Examples: Deep Learning changes Mathematical Methods

Limited-Angle Computed Tomography

A CT scanner samples the *Radon transform*

$$\mathcal{R}f(\phi, s) = \int_{L(\phi,s)} f(x) dS(x),$$



for $L(\phi, s) = \left\{x \in \mathbb{R}^2 : x_1 \cos(\phi) + x_2 \sin(\phi) = s\right\}$, $\phi \in [-\pi/2, \pi/2)$, and $s \in \mathbb{R}$.

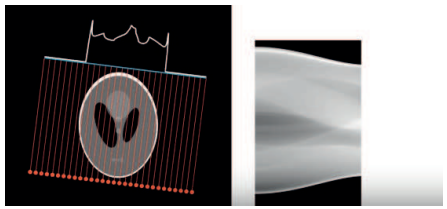
Challenging inverse problem if $\mathcal{R}f(\cdot, s)$ is only sampled on $[-\phi, \phi]$, $\phi < \pi/2$

Examples: Deep Learning changes Mathematical Methods

Limited-Angle Computed Tomography

A CT scanner samples the *Radon transform*

$$\mathcal{R}f(\phi, s) = \int_{L(\phi,s)} f(x) dS(x),$$



for $L(\phi, s) = \left\{x \in \mathbb{R}^2 : x_1 \cos(\phi) + x_2 \sin(\phi) = s\right\}$, $\phi \in [-\pi/2, \pi/2)$, and $s \in \mathbb{R}$.

Challenging inverse problem if $\mathcal{R}f(\cdot, s)$ is only sampled on $[-\phi, \phi]$, $\phi < \pi/2$

Learn the Invisible (Bubba, K, Lassas, März, Samek, Siltanen, Srinivan; 2018):

Step 1: Use model-based methods as far as possible

- Solve with sparse regularization using shearlets.

Step 2: Use data-driven methods where it is necessary

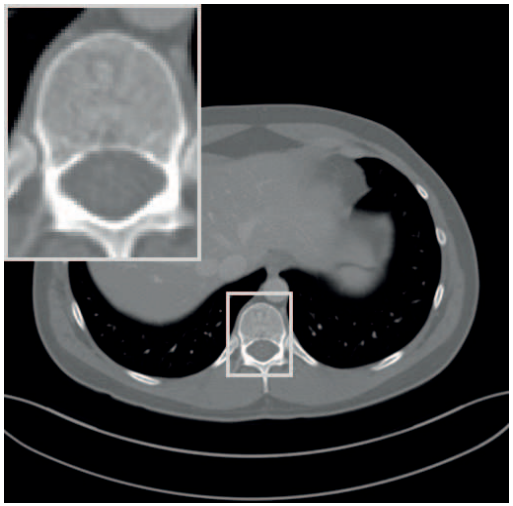
- Use a deep neural network to recover the missing components.

Step 3: Carefully combine both worlds

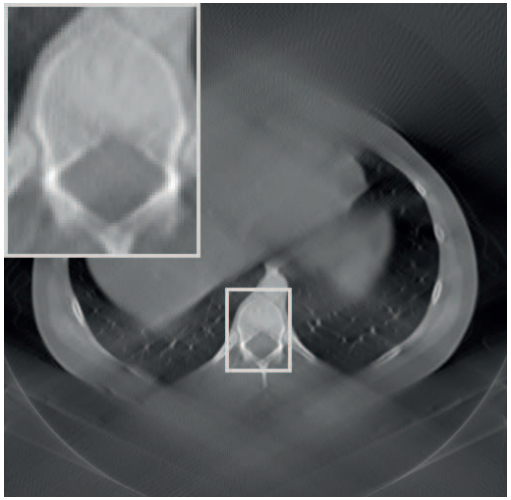
- Combine outcome of Step 1 and 2.

Learn the Invisible (LtI)

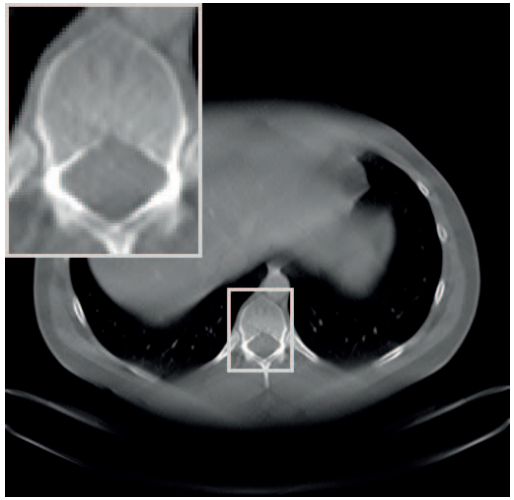
(Bubba, K, Lassas, März, Samek, Siltanen, Srinivan; 2018)



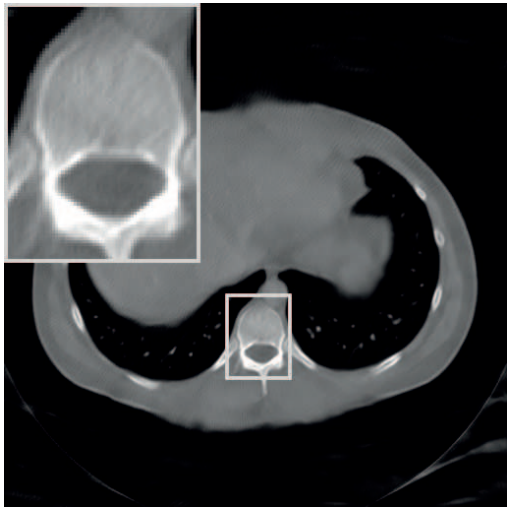
Original



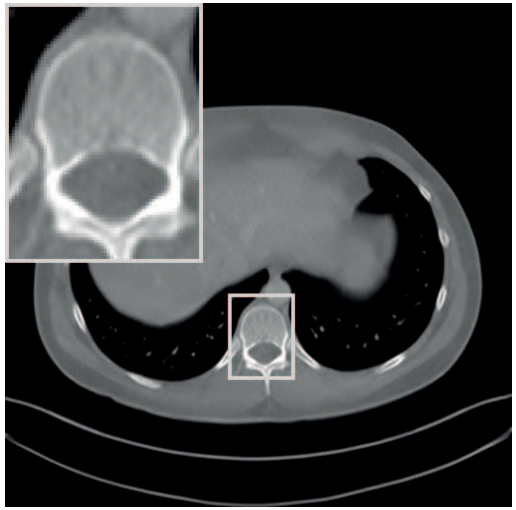
Filtered Backprojection



Sparse Regularization with Shearlets



[Gu & Ye, 2017]

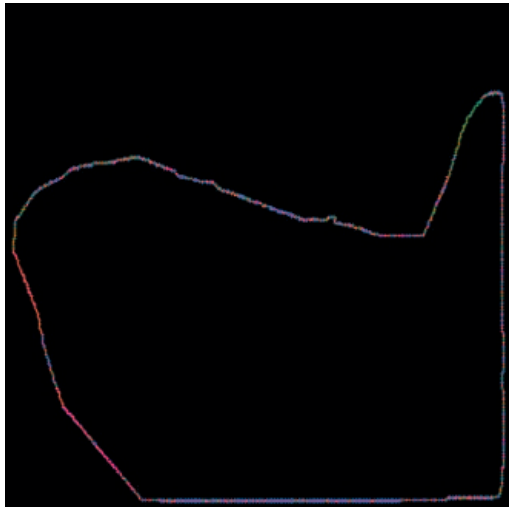


Learn the Invisible (LtI)

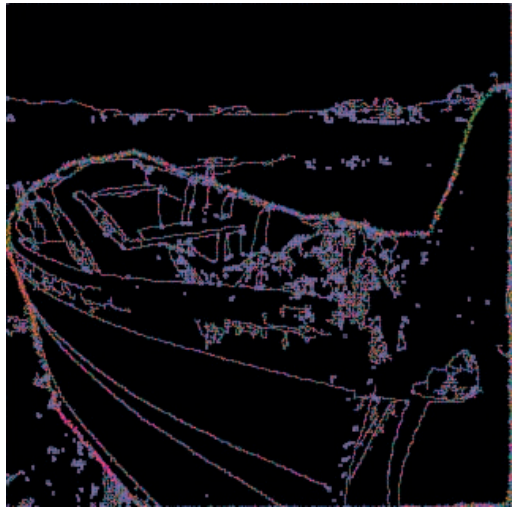
Deep Network Shearlet Edge Extractor (DeNSE)

(Andrade-Loarca, K, Öktem, Petersen; 2019)

Original

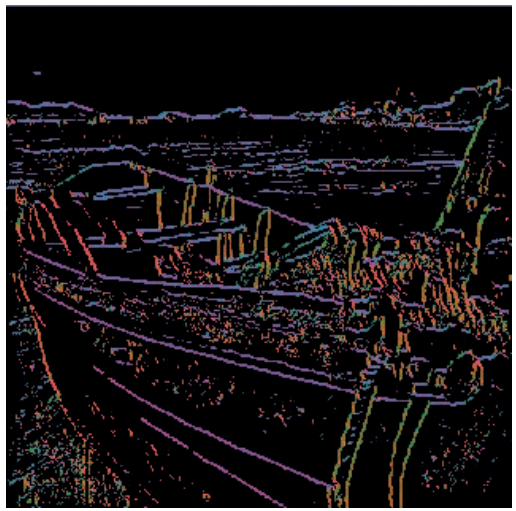


Human Annotation



SEAL [Yu et al; 2018]

CoShREM [Reisenhofer et al.; 2015]



DeNSE

Deep Learning changes Mathematical Methods

Optimal Balancing of
Classical Methods and Deep Learning!

Deep Neural Networks

Deep neural networks have recently shown impressive results in a variety of real-world applications and outformed already various mathematical based approaches.

Deep Neural Networks

Deep neural networks have recently shown impressive results in a variety of real-world applications and outformed already various mathematical based approaches.

Problem:

Very few theoretical results explaining their success!

Deep Neural Networks

Deep neural networks have recently shown impressive results in a variety of real-world applications and outformed already various mathematical based approaches.

Problem:

Very few theoretical results explaining their success!

Nice article:

Deep, Deep Trouble:

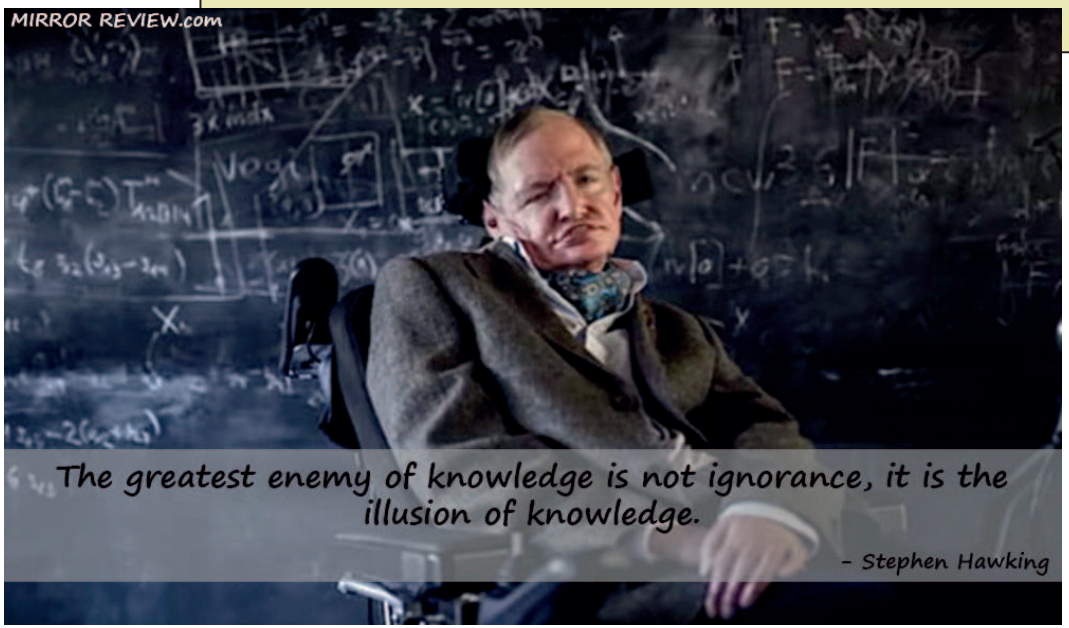
Deep Learnings Impact on Image Processing, Mathematics, and Humanity

Michael Elad (CS, Technion), SIAM News

Deep Learning = Alchemy?

„Ali Rahimi, a researcher in artificial intelligence (AI) at Google in San Francisco, California, took a swipe at his field last December—and received a 40-second ovation for it. Speaking at an AI conference, Rahimi charged that **machine learning algorithms, in which computers learn through trial and error, have become a form of „alchemy."** Researchers, he said, **do not know why some algorithms work and others don't, nor do they have rigorous criteria for choosing one AI architecture over another**...."

Science, May 2018



A Bit of History

First Appearance of Neural Networks

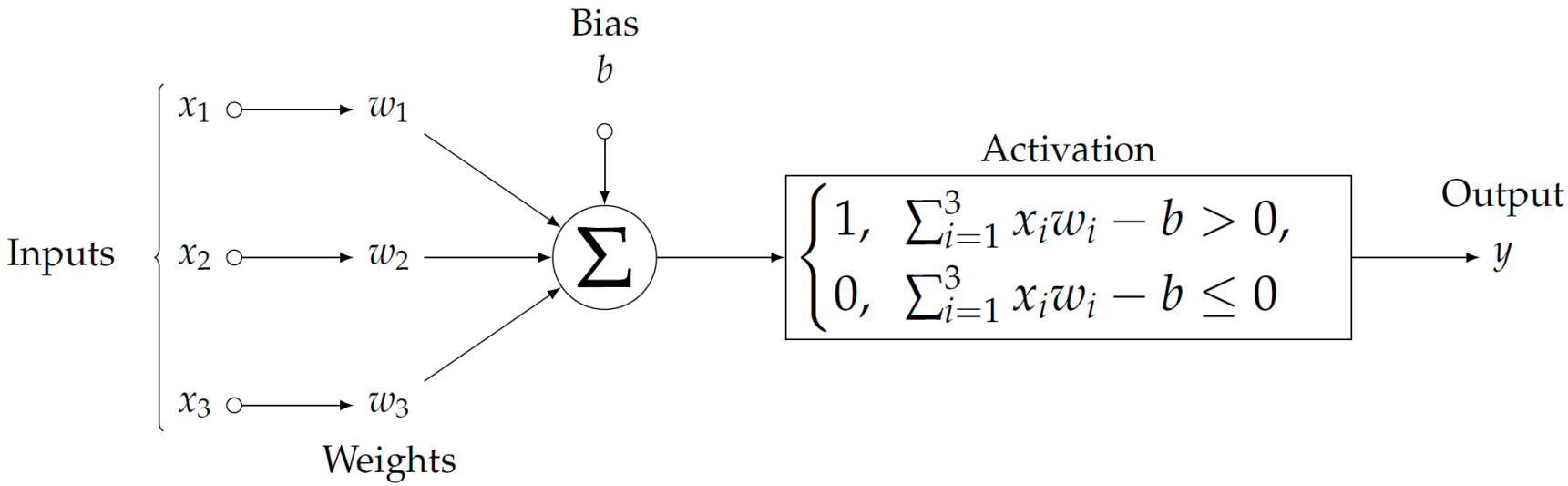
Key Task of McCulloch and Pitts (1943):

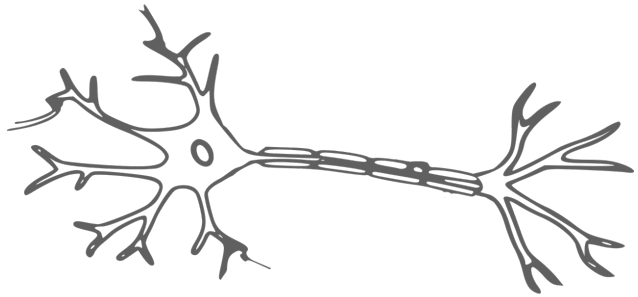
- Develop an algorithmic approach to learning.
- Mimicking the functionality of the human brain.

Goal: Artifical Intelligence!

Artificial Neurons

Mimic the human brain!





Artificial Neurons

Definition: An *artificial neuron* with *weights* $w_1, ..., w_n \in \mathbb{R}$, *bias* $b \in \mathbb{R}$ and *activation function* $\varrho : \mathbb{R} \to \mathbb{R}$ is defined as the function $f : \mathbb{R}^n \to \mathbb{R}$ given by

$$f(x_1, ..., x_n) = \varrho\left(\sum_{i=1}^{n} x_i w_i - b\right) = \varrho(\langle x, w \rangle - b),$$

where $w = (w_1, ..., w_n)$ and $x = (x_1, ..., x_n)$.

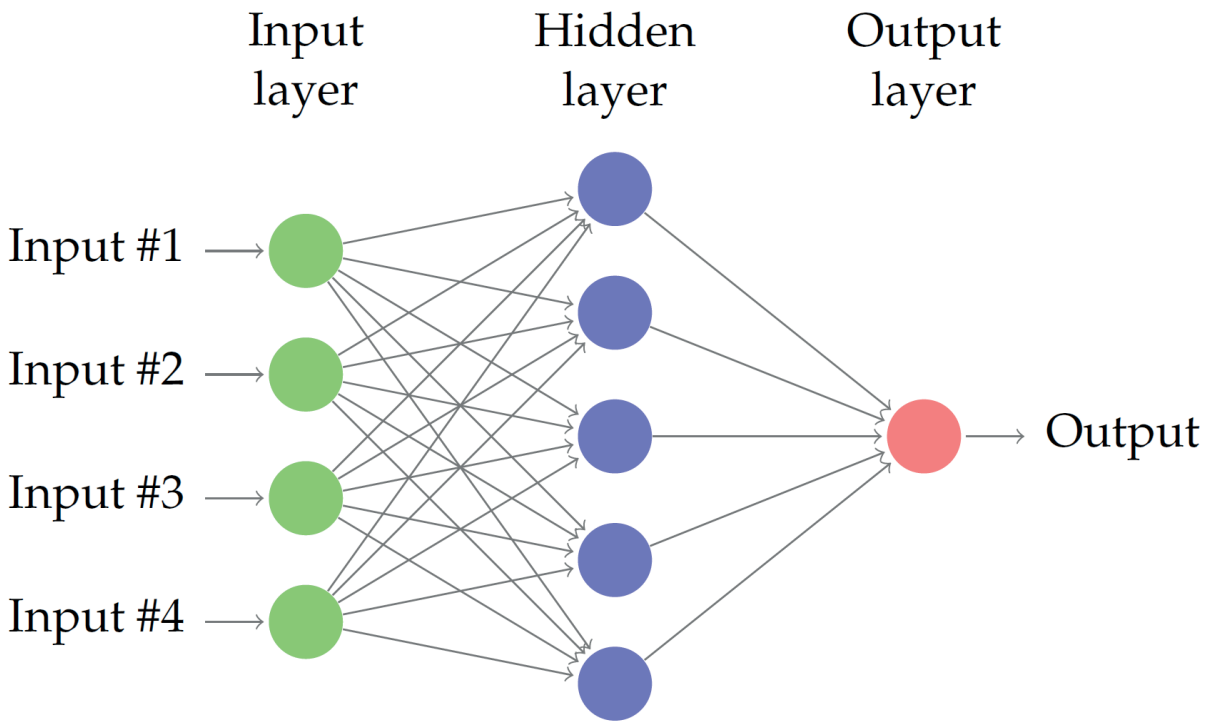
Examples of Activation Functions:

- Heaviside function $\varrho(x) = \begin{cases} 1, & x > 0, \\ 0, & x \leq 0. \end{cases}$
- Sigmoid function $\varrho(x) = \frac{1}{1+e^{-x}}$.
- Rectifiable Linear Unit (ReLU) $\varrho(x) = \max\{0, x\}$.
- Softmax function $\varrho(x) = \ln(1 + e^x)$.

Artificial Neural Network

Definition:

An *artificial neural network* is a graph which consists of artificial neurons. A *feed-forward neural network* is a directed, acyclic graph. All other neural networks are called *recurrent neural networks*.



Second Appearance of Neural Networks

Key Observations by Y. LeCun et al. (around 2000):

- Drastic improvement of computing power.
  - $\rightsquigarrow$ *Networks with hundreds of layers can be trained.*
  - $\rightsquigarrow$ *Deep Neural Networks!*
- Age of Data starts.
  - $\rightsquigarrow$ *Vast amounts of training data is available.*

Second Appearance of Neural Networks

Key Observations by Y. LeCun et al. (around 2000):

- Drastic improvement of computing power.
  $\rightsquigarrow$ *Networks with hundreds of layers can be trained.*
  $\rightsquigarrow$ *Deep Neural Networks!*
- Age of Data starts.
  $\rightsquigarrow$ *Vast amounts of training data is available.*

Current Situation:

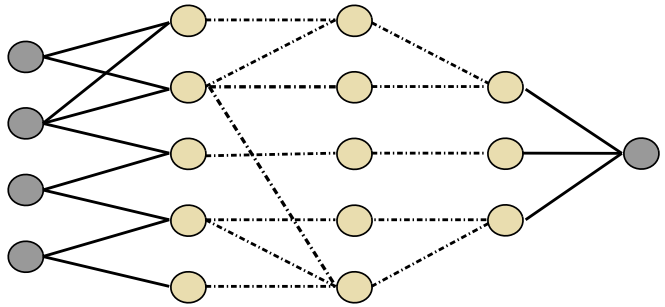
- Tremendous impact on science and public life.
- Development of theoretical foundation accelerating.
- Joint effort of mathematics and theoretical computer science.
- Paradigm change in mathematics.

Mathematics of Deep Neural Networks

The Mathematics of Deep Neural Networks

"Rough" Definition:

Assume the following notions:



- $d \in \mathbb{N}$: Dimension of input layer.
- L: Number of layers.
- N: Number of neurons.
- $\varrho : \mathbb{R} \to \mathbb{R}$: (Non-linear) function called *activation function*.
- $T_\ell : \mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell}$, $\ell = 1, \dots, L$: Affine linear maps.

Then $\Phi : \mathbb{R}^d \to \mathbb{R}^{N_L}$ given by

$$\Phi(x) = T_L \varrho(T_{L-1} \varrho(\dots \varrho(T_1(x))), \quad x \in \mathbb{R}^d,$$

is called *(deep) neural network (DNN)*.

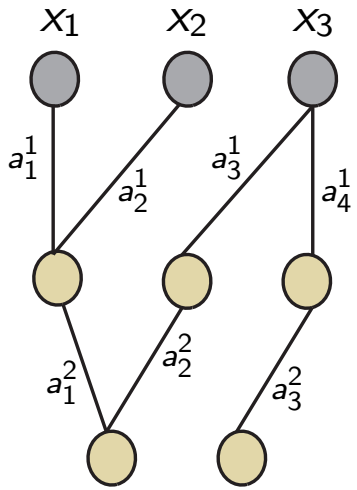
Affine Linear Maps and Weights

Remark: The affine linear map T_ℓ is defined by a matrix $A_\ell \in \mathbb{R}^{N_{\ell-1} \times N_\ell}$ and an affine part $b_\ell \in \mathbb{R}^{N_\ell}$ via

$$T_\ell(x) = A_\ell x + b_\ell.$$

$$A_1 = \begin{pmatrix} a_1^1 & a_2^1 & 0 \\ 0 & 0 & a_3^1 \\ 0 & 0 & a_4^1 \end{pmatrix}$$

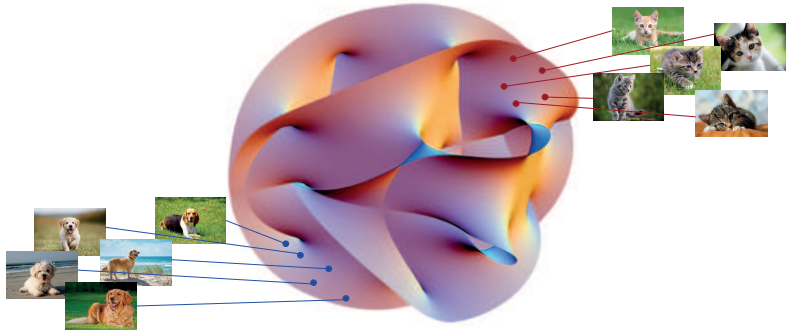
$$A_2 = \begin{pmatrix} a_1^2 & a_2^2 & 0 \\ 0 & 0 & a_3^2 \end{pmatrix}$$



Training of Deep Neural Networks

High-Level Set Up:

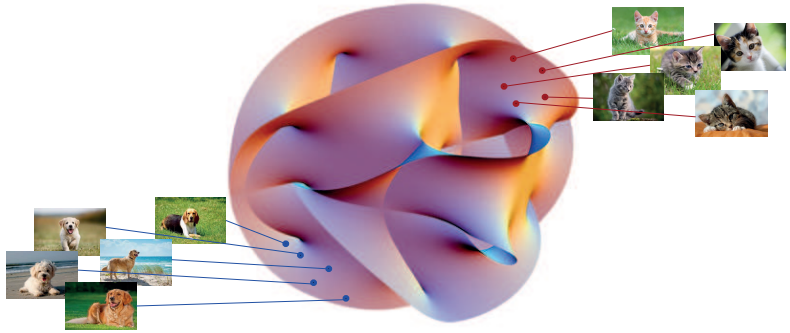
- Samples $(x_i, f(x_i))_{i=1}^m$ of a function such as $f : \mathcal{M} \to \{1, 2, \ldots, K\}$.



Training of Deep Neural Networks

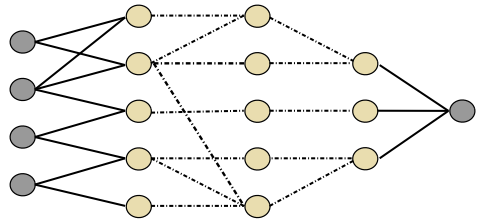
High-Level Set Up:

- Samples $(x_i, f(x_i))_{i=1}^m$ of a function such as $f : \mathcal{M} \to \{1, 2, \dots, K\}$.



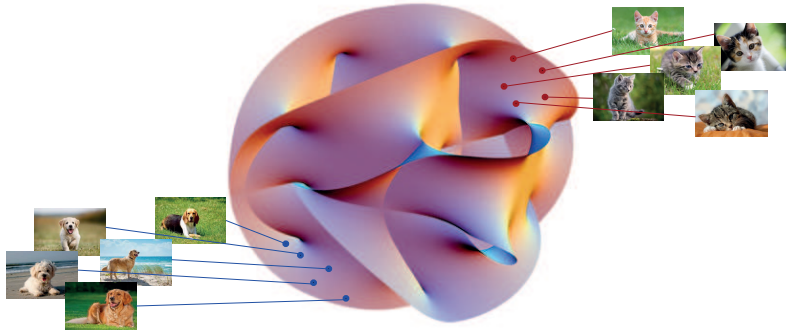
- Select an architecture of a deep neural network, i.e., a choice of d, L, $(N_\ell)_{\ell=1}^L$, and ϱ.

  Sometimes selected entries of the matrices $(A_\ell)_{\ell=1}^L$, i.e., weights, are set to zero at this point.

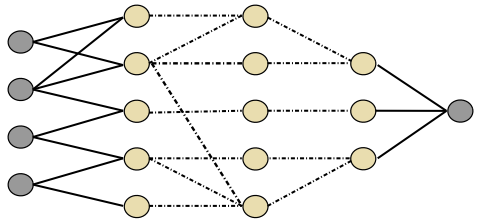


Training of Deep Neural Networks

High-Level Set Up:



- Samples $(x_i, f(x_i))_{i=1}^m$ of a function such as $f : \mathcal{M} \to \{1, 2, \dots, K\}$.



- Select an architecture of a deep neural network, i.e., a choice of d, L, $(N_\ell)_{\ell=1}^L$, and ϱ.

  Sometimes selected entries of the matrices $(A_\ell)_{\ell=1}^L$, i.e., weights, are set to zero at this point.

- Learn the affine-linear functions $(T_\ell)_{\ell=1}^L = (A_\ell \cdot + b_\ell)_{\ell=1}^L$ by

$$\min_{(A_\ell, b_\ell)_\ell} \sum_{i=1}^m \mathcal{L}(\Phi_{(A_\ell, b_\ell)_\ell}(x_i), f(x_i)) + \lambda \mathcal{R}((A_\ell, b_\ell)_\ell)$$

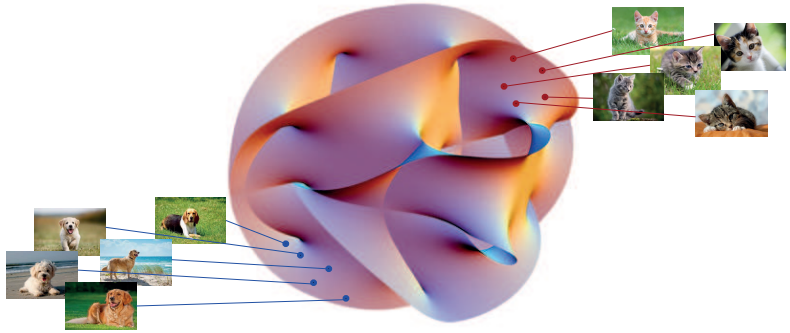
  yielding the network $\Phi_{(A_\ell, b_\ell)_\ell} : \mathbb{R}^d \to \mathbb{R}^{N_L}$,

$$\Phi_{(A_\ell, b_\ell)_\ell}(x) = T_L \varrho(T_{L-1} \varrho(\dots \varrho(T_1(x))).$$

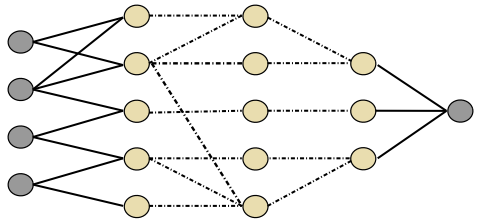
  This is often done by stochastic gradient descent.

Training of Deep Neural Networks

High-Level Set Up:



- Samples $(x_i, f(x_i))_{i=1}^m$ of a function such as $f : \mathcal{M} \to \{1, 2, \dots, K\}$.



- Select an architecture of a deep neural network, i.e., a choice of d, L, $(N_\ell)_{\ell=1}^L$, and ϱ.

  Sometimes selected entries of the matrices $(A_\ell)_{\ell=1}^L$, i.e., weights, are set to zero at this point.

- Learn the affine-linear functions $(T_\ell)_{\ell=1}^L = (A_\ell \cdot + b_\ell)_{\ell=1}^L$ by

$$\min_{(A_\ell, b_\ell)_\ell} \sum_{i=1}^m \mathcal{L}(\Phi_{(A_\ell, b_\ell)_\ell}(x_i), f(x_i)) + \lambda \mathcal{R}((A_\ell, b_\ell)_\ell)$$

  yielding the network $\Phi_{(A_\ell, b_\ell)_\ell} : \mathbb{R}^d \to \mathbb{R}^{N_L}$,

$$\Phi_{(A_\ell, b_\ell)_\ell}(x) = T_L \varrho(T_{L-1} \varrho(\dots \varrho(T_1(x))).$$

This is often done by stochastic gradient descent.

Goal: $\Phi_{(A_\ell, b_\ell)_\ell} \approx f$

Fundamental Questions concerning Deep Neural Networks

- *Expressivity:*
  - How powerful is the network architecture?
  - Can it indeed represent the correct functions?

  $\rightsquigarrow$ *Applied Harmonic Analysis, Approximation Theory, ...*

Fundamental Questions concerning Deep Neural Networks

- *Expressivity:*
  - How powerful is the network architecture?
  - Can it indeed represent the correct functions?

  $\rightsquigarrow$ *Applied Harmonic Analysis, Approximation Theory, ...*
- *Learning:*
  - Why does the current learning algorithm produce anything reasonable?
  - What are good starting values?

  $\rightsquigarrow$ *Differential Geometry, Optimal Control, Optimization, ...*

Fundamental Questions concerning Deep Neural Networks

- *Expressivity:*
  - How powerful is the network architecture?
  - Can it indeed represent the correct functions?

  $\rightsquigarrow$ *Applied Harmonic Analysis, Approximation Theory, ...*
- *Learning:*
  - Why does the current learning algorithm produce anything reasonable?
  - What are good starting values?

  $\rightsquigarrow$ *Differential Geometry, Optimal Control, Optimization, ...*
- *Generalization:*
  - Why do deep neural networks perform that well on data sets, which do not belong to the input-output pairs from a training set?
  - What impact has the depth of the network?

  $\rightsquigarrow$ *Learning Theory, Optimization, Statistics, ...*

Fundamental Questions concerning Deep Neural Networks

- *Expressivity:*
  - How powerful is the network architecture?
  - Can it indeed represent the correct functions?

  $\rightsquigarrow$ *Applied Harmonic Analysis, Approximation Theory, ...*
- *Learning:*
  - Why does the current learning algorithm produce anything reasonable?
  - What are good starting values?

  $\rightsquigarrow$ *Differential Geometry, Optimal Control, Optimization, ...*
- *Generalization:*
  - Why do deep neural networks perform that well on data sets, which do not belong to the input-output pairs from a training set?
  - What impact has the depth of the network?

  $\rightsquigarrow$ *Learning Theory, Optimization, Statistics, ...*
- *Interpretability:*
  - Why did a trained deep neural network reach a certain decision?
  - Which components of the input do contribute most?

  $\rightsquigarrow$ *Information Theory, Uncertainty Quantification, ...*

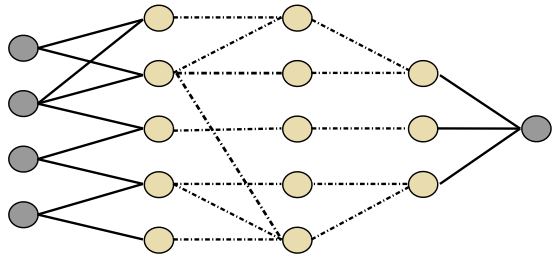
Topics of this Seminar

- Basics of Statistical Learning Theory
- Training of Deep Neural Networks
- Mathematical Theory of Deep Learning:
  - Expressivity
  - Optimization
  - Generalization
  - Interpretability
- Deep Learning in Mathematical Approaches:
  - Inverse Problems/Imaging Science
  - Numerical Analysis of Partial Differential Equations

Deep Neural Networks Enter the Stage:

Let's be more concrete....

Key Notions, I



Definition: Let

- $d \in \mathbb{N}$ be the input dimension,
- $L \in \mathbb{N}$ be the number of layers,
- $N_0, N_1, ..., N_L$ the number of neurons in each layer and $N_0 := d$,
- $A_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}, \ell = 1, ..., L$ be the weights of the edges
- $b_\ell \in \mathbb{R}^{N_\ell}, \ell = 1, ..., L$ be the biases, and
- $\varrho : \mathbb{R} \to \mathbb{R}$ be the (non-linear) activation function.

Then

$$\Phi = ((A_\ell, b_\ell))_{\ell=1}^{L}$$

is called *neural network* ("*architecture*") and the map

$$R_\varrho(\Phi) : \mathbb{R}^d \to \mathbb{R}^{N_L}, \quad R_\varrho(\Phi)(x) := x_L,$$

where $x_0 := x$, $x_\ell := \varrho(A_\ell x_{\ell-1} - b_\ell)$, $\ell = 1, ...L-1$, and $x_L := A_L x_{L-1} - b_L$ is called the *realization of Φ with activation function ϱ*.

Key Notions, II

Definition (continued):

We further call

- $N(\Phi) := d + \sum_{\ell=1}^{L} N_L$ the total number of neurons,
- $L(\Phi) := L$ the number of layers,
- $M(\Phi) := \sum_{\ell=1}^{L} \|A_\ell\|_0 + \|b_\ell\|_0$ the number of weights (edges), where $\|\cdot\|_0$ is the number of non-zero entries.

We say that

- Φ is *sparsely connected*, if $M(\Phi)$ is small,
- Φ is a *shallow neural network*, if $L(\Phi)$ is small,
- Φ is a *deep neural network*, if $L(\Phi)$ is large.

Key Notions, II

Definition (continued):

We further call

- $N(\Phi) := d + \sum_{\ell=1}^{L} N_L$ the total number of neurons,
- $L(\Phi) := L$ the number of layers,
- $M(\Phi) := \sum_{\ell=1}^{L} \|A_\ell\|_0 + \|b_\ell\|_0$ the number of weights (edges), where $\|\cdot\|_0$ is the number of non-zero entries.

We say that

- Φ is *sparsely connected*, if $M(\Phi)$ is small,
- Φ is a *shallow neural network*, if $L(\Phi)$ is small,
- Φ is a *deep neural network*, if $L(\Phi)$ is large.

For $d \in \mathbb{N}$ and $M, L, N \in \mathbb{N} \cup \{\infty\}$, we denote by

$$\mathcal{NN}_{d,M,N,L}$$

the set of neural networks Φ with input dimension d, $N_L = 1$ and

$$M(\Phi) \leq M, N(\Phi) \leq N, L(\Phi) \leq L.$$

Key Notions, III

Definition (continued):

If the size of the weights are a concern, we denote by

$$\mathcal{NN}^R_{d,M,N,L}$$

the set of neural networks Φ with input dimension d, $N_L = 1$, with

$$M(\Phi) \leq M, N(\Phi) \leq N, L(\Phi) \leq L,$$

and with all weights bounded by R.

Getting Familiar with the Notions...

Examples:

(1) Let Φ be given by

$$A_1 = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 0 & 5 \\ 0 & 0 & 4 \end{pmatrix}, \quad b_1 = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 2 & 6 & 0 \\ 0 & 0 & 7 \end{pmatrix}, \quad b_1 = \begin{pmatrix} 8 \\ 8 \end{pmatrix}.$$

Then $\Phi \in \mathcal{NN}_{d,M,N,L}$ with

Getting Familiar with the Notions...

Examples:

(1) Let Φ be given by

$$A_1 = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 0 & 5 \\ 0 & 0 & 4 \end{pmatrix}, \quad b_1 = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 2 & 6 & 0 \\ 0 & 0 & 7 \end{pmatrix}, \quad b_1 = \begin{pmatrix} 8 \\ 8 \end{pmatrix}.$$

Then $\Phi \in \mathcal{NN}_{d,M,N,L}$ with

$$d = 3, \ M = 12, \ L = 2, \text{ and } N = 8.$$

Getting Familiar with the Notions...

Examples:

(1) Let Φ be given by

$$A_1 = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 0 & 5 \\ 0 & 0 & 4 \end{pmatrix}, \quad b_1 = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 2 & 6 & 0 \\ 0 & 0 & 7 \end{pmatrix}, \quad b_1 = \begin{pmatrix} 8 \\ 8 \end{pmatrix}.$$

Then $\Phi \in \mathcal{NN}_{d,M,N,L}$ with

$$d = 3, \ M = 12, \ L = 2, \text{ and } N = 8.$$

(2) An artificial neuron is a realization with activation function ϱ of a neural network $\Phi \in \mathcal{NN}_{d,M,N,L}$ with

Getting Familiar with the Notions...

Examples:

(1) Let Φ be given by

$$A_1 = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 0 & 5 \\ 0 & 0 & 4 \end{pmatrix}, \quad b_1 = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 2 & 6 & 0 \\ 0 & 0 & 7 \end{pmatrix}, \quad b_1 = \begin{pmatrix} 8 \\ 8 \end{pmatrix}.$$

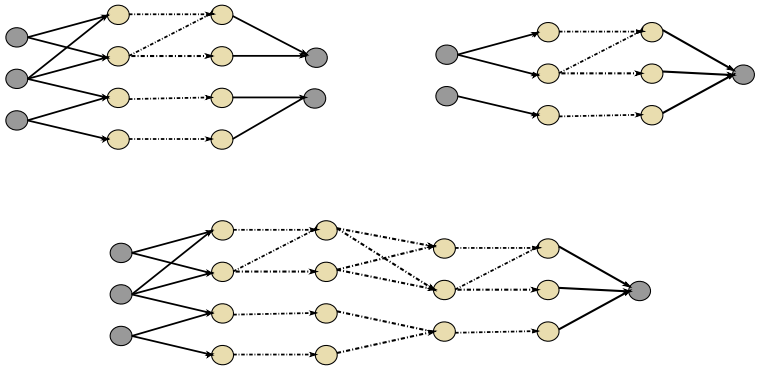
Then $\Phi \in \mathcal{NN}_{d,M,N,L}$ with

$$d = 3, \ M = 12, \ L = 2, \text{ and } N = 8.$$

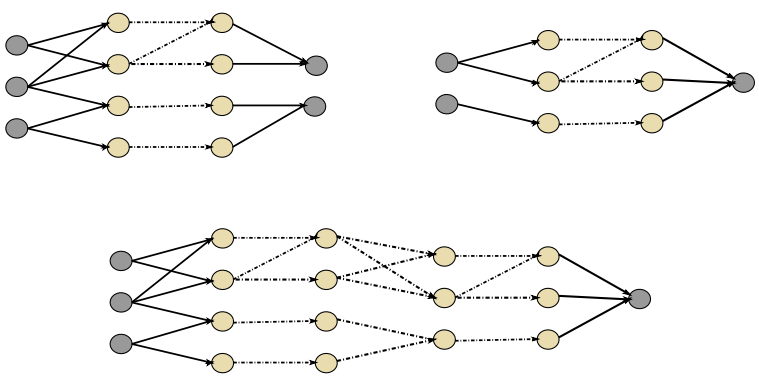
(2) An artificial neuron is a realization with activation function ϱ of a neural network $\Phi \in \mathcal{NN}_{d,M,N,L}$ with

$$M = d, \ L = 1, \text{ and } N = d + 1.$$

Basic Operations: Concatenation



Basic Operations: Concatenation



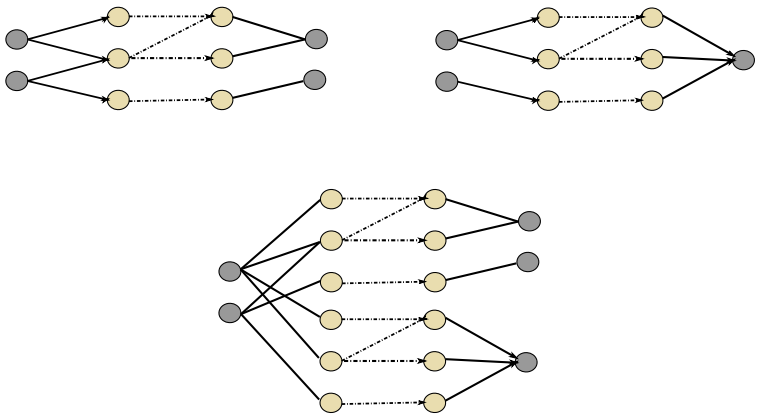
Lemma: Let $L_1, L_2 \in \mathbb{N}$ and $\Phi^i = ((A_1^i, b_1^i), ..., (A_{L_i}^i, b_{L_i}^i)), i = 1, 2$, be neural networks such that the input layer of Φ^1 has the same dimension as the output layer of Φ^2. Then, for any activation function ϱ,

$$R_\varrho(\Phi^1 \circ \Phi^2) = R_\varrho(\Phi^1) \circ R_\varrho(\Phi^2) \quad \text{and} \quad L(\Phi^1 \circ \Phi^2) = L(\Phi^1) + L(\Phi^2) - 1,$$

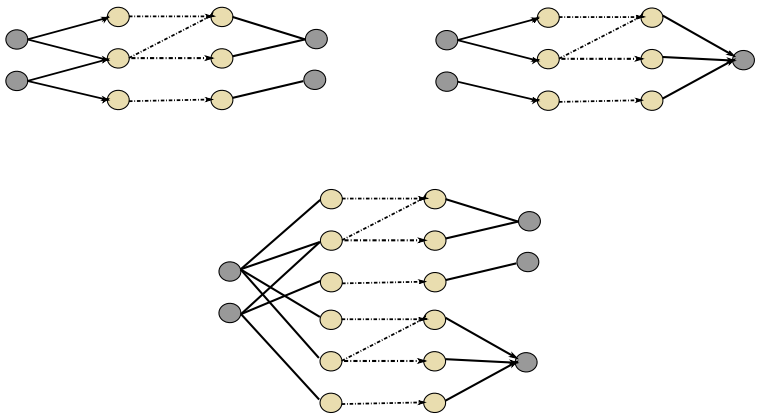
where $\Phi^1 \circ \Phi^2$ denotes the *concatenation of Φ^1 and Φ^2*:

$$((A_1^2, b_1^2), ...(A_{L_2-1}^2, b_{L_2-1}^2), (A_1^1 A_{L_2}^2, A_1^1 b_{L_2}^2 + b_1^1), (A_2^1, b_2^1), ...(A_{L_1}^1, b_{L_1}^1)).$$

Basic Operations: Parallelization



Basic Operations: Parallelization



Lemma: Let $L \in \mathbb{N}$ and $\Phi^i = ((A_1^i, b_1^i), ..., (A_L^i, b_L^i)), i = 1, 2....$ Then, for any activation function ϱ and any $x \in \mathbb{R}^d$,

$$\begin{aligned} (R_\varrho(P(\Phi^1, \Phi^2)))(x) &= (R_\varrho(\Phi^1)(x), R_\varrho(\Phi^2)(x)), \\ L(P(\Phi^1, \Phi^2)) &= \max\{L(\Phi^1), L(\Phi^2)\}, \\ M(P(\Phi^1, \Phi^2)) &= M(\Phi^1) + M(\Phi^2) - d, \end{aligned}$$

where $P(\Phi^1, \Phi^2)$ denotes the *parallelization of* Φ^1 *and* Φ^2:

$$P(\Phi^1, \Phi^2) := ((\hat{A}_1, \hat{b}_1), ..., (\hat{A}_L, \hat{b}_L)) \text{ with}$$

$$\hat{A}_1 = \begin{pmatrix} A_1^1 \\ A_1^2 \end{pmatrix}, \hat{b}_1 = \begin{pmatrix} b_1^1 \\ b_1^2 \end{pmatrix}, \hat{A}_\ell = \begin{pmatrix} A_\ell^1 & 0 \\ 0 & A_\ell^2 \end{pmatrix}, \hat{b}_\ell = \begin{pmatrix} b_\ell^1 \\ b_\ell^2 \end{pmatrix}, 1 \leq \ell \leq L.$$

Doing Nothing...

Lemma: Define

$$\Phi^{\mathrm{Id}} := ((A_1, b_1), (A_2, b_2))$$

with

$$A_1 = \begin{pmatrix} \mathrm{Id}_{\mathbb{R}^d} \\ -\mathrm{Id}_{\mathbb{R}^d} \end{pmatrix}, \quad b_1 = b_2 = 0, \quad A_2 = (\mathrm{Id}_{\mathbb{R}^d}, -\mathrm{Id}_{\mathbb{R}^d}).$$

Then

$$R_{ReLU}(\Phi^{\mathrm{Id}})(x) = x \quad \text{for all } x \in \mathbb{R}^d.$$

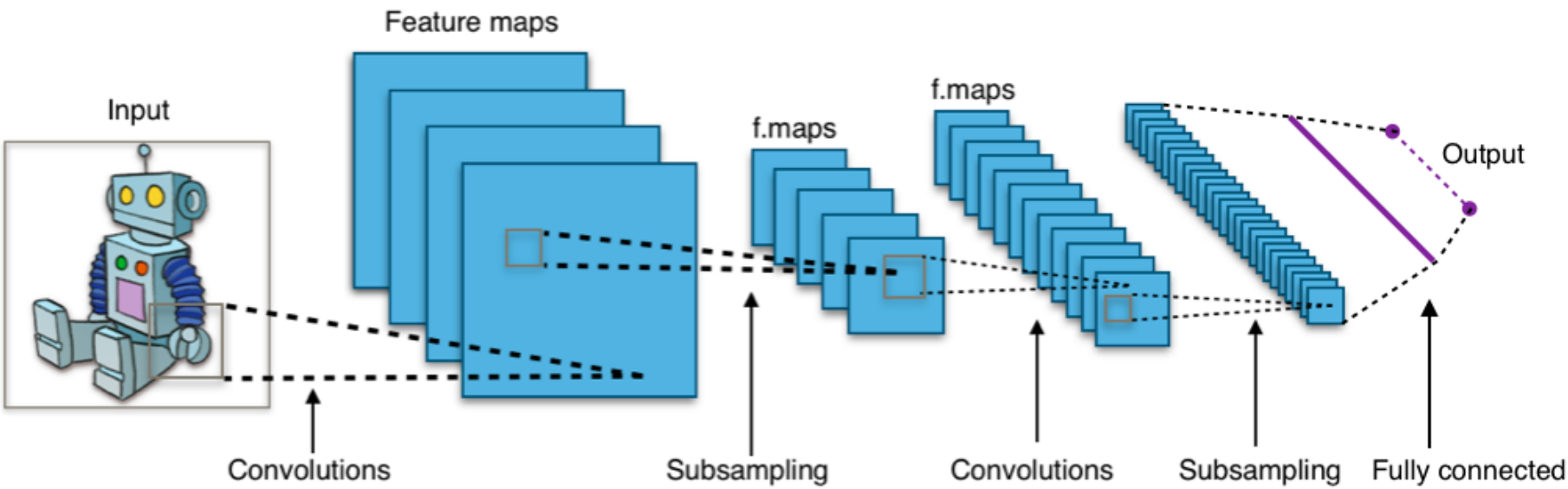
Remark: Let Φ be a neural network with input dimension d. Then

$$R_{ReLU}(\Phi) = R_{ReLU}(\Phi \circ \Phi^{\mathrm{Id}}),$$

i.e., different architectures can lead to the same realization.

Convolutional Neural Networks

Typical Convolutional Neural Network



Convolutional Layer

Definition:

- A *convolutional node* has a stack of images $X \in \mathbb{R}^{n_1,n_2,S}$ as input. Given a filter $W \in \mathbb{R}^{F,F,S}$, where F is the spatial extend of the filter, and a bias $b \in \mathbb{R}$, it computes

$$\begin{aligned} Z[i,j] &= (W *_{12} X)[i,j] + b \\ &= \sum_{k=1}^{S} \underbrace{W[\cdot,\cdot,k] * X[\cdot,\cdot,k]}_{\text{convolution for each image}} + b \end{aligned}$$

- A *convolutional layer* consists of K convolutional nodes $((W_k, b_k))_{k=1}^{K} \subseteq \mathbb{R}^{F,F,S} \times \mathbb{R}$ and produces as output a stack

$$Z[i,j,k] := W_k *_{12} X + b_k$$

Pooling Layer

Definition:

A *pooling operator* R acts layerwise on each $Z \in \mathbb{R}^{n_1,n_2,S}$ and results in

$$R(Z) \in \mathbb{R}^{m_1,m_2,S}, m_1 < n_1, m_2 < n_2.$$

Examples:

1. *Sub-Sampling*:

$$R(Z)[i,j,k] = Z[s_1 i, s_2 j, k]$$

with $\frac{s_1}{n_1}, \frac{s_2}{n_2}$. Then $m_1 = \frac{n_1}{s_1}, m_2 = \frac{n_2}{s_2}$.

2. *Averaging*:

$$R(Z)[i,j,k] = \left(\sum_{l=s_1 \cdot i}^{s_1 i + s_1 - 1} \sum_{t=s_2 \cdot j}^{s_2 j + s_2 - 1} Z[l,t,k] \right) \frac{1}{s_1 + s_2}$$

3. *Max-Pooling*:

$$R(Z)[i,j,k] = \max_{l=s_1 \cdot i, \ldots s_1 \cdot i + s_1 - 1, t = s_2 \cdot i, \ldots s_2 \cdot j + s_2 - 1} |Z[l,t,k]|$$

Overall Architecture

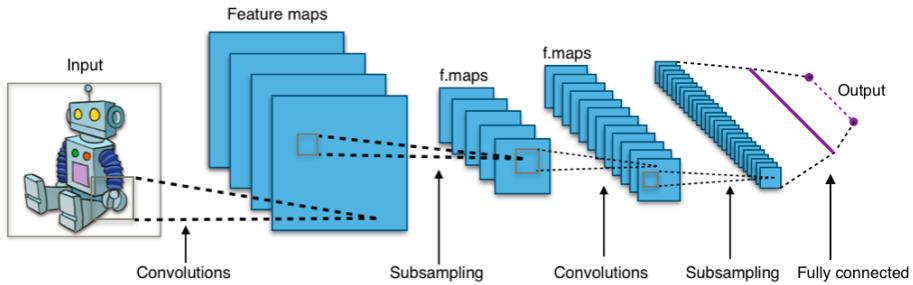
Definition:

A *convolutional neural network (CNN)* with L layers consists of L iterated applications of a convolutional layer followed by an activation layer and possibly a pooling layer.

Remark:

A typical architecture consists of the following components (e.g., LeNet (LeCun et. al 1998)):

- First a CNN as feature-extractor.
- Second a fully connected NN as classifier.



Deep Neural Networks: Training

Problem Setting

Let $d = N_0 \in \mathbb{N}$ and $N_1, ..., N_L, L \in \mathbb{N}$ and let ϱ be an activation function. Then consider the hypothesis space

$$\mathcal{H} := \{R_\varrho(\Phi) : \Phi = ((A_1, b_1), ..., (A_L, b_L)),\ A_\ell \in \mathbb{R}^{N_{\ell-1}, N_\ell},\ b_\ell \in \mathbb{R}^{N_\ell}\}.$$

Task: Given samples $z = ((x_i, y_i))_{i=1}^m \subseteq \mathbb{R}^d \times \mathbb{R}^{N_L}$, find the empirical target function

$$f_z := f_{\mathcal{H},z} = \text{argmin}_{f \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} (f(x_i) - y_i)^2.$$

Problem Setting

Let $d = N_0 \in \mathbb{N}$ and $N_1, ..., N_L, L \in \mathbb{N}$ and let ϱ be an activation function. Then consider the hypothesis space

$$\mathcal{H} := \{R_\varrho(\Phi) : \Phi = ((A_1, b_1), ..., (A_L, b_L)),\ A_\ell \in \mathbb{R}^{N_{\ell-1}, N_\ell},\ b_\ell \in \mathbb{R}^{N_\ell}\}.$$

Task: Given samples $z = ((x_i, y_i))_{i=1}^m \subseteq \mathbb{R}^d \times \mathbb{R}^{N_L}$, find the empirical target function

$$f_z := f_{\mathcal{H},z} = \text{argmin}_{f \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m (f(x_i) - y_i)^2.$$

More General Task: One can also consider a more general case such as

$$f_z = \text{argmin}_{f \in \mathcal{H}} \sum_{i=1}^m \mathcal{L}(f, x_i, y_i)$$

where $\mathcal{L} : C(\mathbb{R}^d, \mathbb{R}^{N_L}) \times \mathbb{R}^d \times \mathbb{R}^{N_L} \to \mathbb{R}$ is a *loss function*.

Gradient Descent

Optimization Approach: A simple optimization method is *gradient descent*. For $F : \mathbb{R}^N \to \mathbb{R}$, this amounts to

$$u_{n+1} \leftarrow u_n - \eta \nabla F(u_n) \text{ for all } n \in \mathbb{N}$$

where $\nabla F(u) = (\frac{\partial F}{\partial u_1}(u), \dots \frac{\partial F}{\partial u_n}(u))$ and η is the step size.

Gradient Descent

Optimization Approach: A simple optimization method is *gradient descent*. For $F : \mathbb{R}^N \to \mathbb{R}$, this amounts to

$$u_{n+1} \leftarrow u_n - \eta \nabla F(u_n) \text{ for all } n \in \mathbb{N}$$

where $\nabla F(u) = (\frac{\partial F}{\partial u_1}(u), \dots \frac{\partial F}{\partial u_n}(u))$ and η is the step size.

In our problem... we have

$$F = \sum_{i=1}^{m} \mathcal{L}(f, x_i, y_i) \text{ and } u = ((A_\ell, b_\ell))_{\ell=1}^{L}.$$

Since

$$\nabla_{((A_\ell, b_\ell))_{\ell=1}^{L}} F = \sum_{i=1}^{m} \nabla_{((A_\ell, b_\ell))_{\ell=1}^{L}} \mathcal{L}(f, x_i, y_i),$$

we need to compute

$$\frac{\partial \mathcal{L}(f, x, y)}{\partial (A_\ell)_{i,j}} \quad \text{and} \quad \frac{\partial \mathcal{L}(f, x, y)}{\partial (b_\ell)_i} \quad \text{for all } i, j, \ell.$$

Backpropagation

Data: A neural network f, a loss function $\mathcal{L}$, points x, y.

Result: The matrices $\nabla_{(A_\ell, b_\ell)_{\ell=1}^L} \mathcal{L}(f, x, y)$.

Algorithm:

Compute a_ℓ, z_ℓ for $\ell = 0, \ldots L$;

Set $\delta_L := 2(f(x) - y)$;

Then $\frac{\partial \mathcal{L}(f,x,y)}{\partial A_L} = \delta_L \cdot a_{L-1}^T$ and $\frac{\partial \mathcal{L}(f,x,y)}{\partial b_L} = \delta_L$;

for $\ell = L - 1$ to 1 **do**

$$\delta_\ell := \operatorname{diag}(\varrho'(z_\ell)) A_{\ell+1}^T \cdot \delta_{\ell+1};$$

$$\text{Then } \frac{\partial \mathcal{L}(f, x, y)}{\partial A_\ell} = \delta_\ell a_{\ell-1}^T \text{ and } \frac{\partial \mathcal{L}(f, x, y)}{\partial b_\ell} = \delta_\ell;$$

return $\nabla_{(A_\ell, b_\ell)_{\ell=1}^L} \mathcal{L}(f, x, y)$.

Stochastic Gradient Descent

Goal: Find a stationary point of

$$F = \sum_{i=1}^{m} F_i : \mathbb{R}^N \to \mathbb{R} \text{ where } F_i = \mathcal{L}(f, x_i, y_i).$$

Data: A neural network f, a loss function $\mathcal{L}$.

Result: A point u_n.

Algorithm:

- Set starting value u_0 and $n = 0$.
- **while** (error is large), **do**
  - Pick $i^* \in \{1, ..., m\}$ uniformly at random;
  - Update $u_{n+1} \leftarrow u_n - \eta \nabla F_{i^*}$;
  - Set $n + 1 \leftarrow n$;

return u_n.

$\rightsquigarrow$ *Mini-Batch!*

AFG
Applied Functional Analysis Group
Technische Universität Berlin

THANK YOU!

References available at:

www.math.tu-berlin.de/∼kutyniok

Code available at:

www.ShearLab.org