Recap

- Neural networks
  - Layered architecture
  - Output of one layer fed into the next
  - Layer contains neurons, a neuron represents a single calculation
  - Activation functions
- Word2Vec training
  - Two architectures
  - Train NN to predict words in contexts
  - Use learned weights as word vectors
  - From Scratch Guide

Neural Networks, Part 2

Sprachverarbeitung (VL + Ü)

Nils Reiter

June 27, 2024

UNIVERSITÄT ZU KÖLN

INSTITUT FÜR DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

Practical Deep Learning

Libraries

- Deep learning in python rests on several independent libraries
  - `numpy` Provides efficient matrices and arrays
  - `pandas` Convenient working with tabular data (inspired by `data.frames` in R)
  - `scikit-learn` ›Classical‹ machine learning (not deep learning)
  - `tensorflow` Basic, low-level machine learning and math
  - `keras` High-level deep learning (built on top of tensorflow)
  - `pytorch` Newer alternative to tensorflow
- Libraries are well integrated

Libraries

- Deep learning in python rests on several independent libraries
  - `numpy` Provides efficient matrices and arrays
  - `pandas` Convenient working with tabular data (inspired by `data.frames` in R)
  - `scikit-learn` ›Classical‹ machine learning (not deep learning)
  - `tensorflow` Basic, low-level machine learning and math
  - `keras` High-level deep learning (built on top of tensorflow)
  - `pytorch` Newer alternative to tensorflow
- Libraries are well integrated
- Documentation is fragmented – important links:
  - `https://keras.io/api/`
  - `https://pandas.pydata.org/docs/reference/index.html`
  - `https://scikit-learn.org/stable/modules/classes.html`

keras

- `https://keras.io`
- High-level Python API for deep learning
- Built on top of tensorflow /pytorch
- Pattern
  1. Layout the network
  2. Set hyper parameters
  3. Run training
  4. Coffee drinks
  5. Evaluation

Configuration

Listing 1: Sequential API

```
# model layout
model = Sequential()
model.add(...)
model.add(...)

# hyperparameter specification
model.compile(loss=...,
  optimizer=...)

# training  Data
model.fit(..., epochs=...,
  batch_size=...)
```

Configuration

Two most basic layer types

- Dense: »Just your regular densely-connected NN layer.«
  - https://keras.io/api/layers/core_layers/dense/

```
layer = Dense(3, # number of neurons
  activation = activations.sigmoid, # activation function
  name = "dense layer 7" # useful for debugging/visualisation
  ... # more options, see docs
)
```

- Input: Marks layers to accept data
  - https://keras.io/api/layers/core_layers/input/

```
layer = Input(shape=(15,) # number of input dimensions/features
  name = "input layer", # useful for debugging/visualisation
  ... # see docs
)
```

Shape

- Description of the dimensionality of the data
- A vector of numbers, giving the number of elements for each dimension
- Python tuple
  - List with fixed length: `x = (5,3,1) #a tuple`
  - ⚠ Tuple with one element printed as `(5,)` or `5`

Shape

- Description of the dimensionality of the data
- A vector of numbers, giving the number of elements for each dimension
- Python tuple
  - List with fixed length: `x = (5,3,1) #a tuple`
  - ⚠ Tuple with one element printed as `(5,)` or `5`

```
x = np.zeros(5) # array([0., 0., 0., 0., 0.])
x.shape # returns (5,)
x = np.zeros((3,5))
# array([[0., 0., 0., 0., 0.],
#        [0., 0., 0., 0., 0.],
#        [0., 0., 0., 0., 0.]])
x.shape # returns (3,5)
```

Section 2

Overfitting

Introduction

- ›Fitting‹: Train a model on data (= »fit« it to the data)
  - Underfitting: The model is not well fitted to the data, i.e., accuracy is low
  - Overfitting: The model is fitted too well to the data, i.e., accuracy is high

Introduction

- ›Fitting‹: Train a model on data (= »fit« it to the data)
  - Underfitting: The model is not well fitted to the data, i.e., accuracy is low
  - Overfitting: The model is fitted too well to the data, i.e., accuracy is high

Why is overfitting a problem?

Introduction

- ›Fitting‹: Train a model on data (= »fit« it to the data)
  - Underfitting: The model is not well fitted to the data, i.e., accuracy is low
  - Overfitting: The model is fitted too well to the data, i.e., accuracy is high

Why is overfitting a problem?

- We want to the model to behave well »in the wild«
- It needs to generalize from training data
- If it is overfitted, it works very well on training data, and very badly on test data

Intuition

$\simeq$ Learning by heart

Example

- Learning by heart gets you through the test
  - I.e., systems achieve high performance

Intuition

$\simeq$ Learning by heart

Example

- Learning by heart gets you through the test
  - I.e., systems achieve high performance
- You are unable to apply your knowledge to situations not exactly as in the test
  - I.e., system performance is lower in the wild

Intuition

≃ Learning by heart

Example

- Learning by heart gets you through the test
  - I.e., systems achieve high performance
- You are unable to apply your knowledge to situations not exactly as in the test
  - I.e., system performance is lower in the wild

Real-World Examples

- Machine learning for COVID-19 detection on chest scans Roberts et al. (2021)
  - »none of the models identified are of potential clinical use due to methodological flaws and/or underlying biases« Roberts et al. (2021, 200)
  - »Using a public dataset alone without additional new data can lead to community-wide overfitting on this dataset. Even if each individual study observes sufficient precautions to avoid overfitting, the fact that the community is focused on outperforming benchmarks on a single public dataset encourages overfitting.« Roberts et al. (2021, 212)

Real-World Examples

- Machine learning for COVID-19 detection on chest scans — Roberts et al. (2021)
  - »none of the models identified are of potential clinical use due to methodological flaws and/or underlying biases« — Roberts et al. (2021, 200)
  - »Using a public dataset alone without additional new data can lead to community-wide overfitting on this dataset. Even if each individual study observes sufficient precautions to avoid overfitting, the fact that the community is focused on outperforming benchmarks on a single public dataset encourages overfitting.« — Roberts et al. (2021, 212)
- Collection of real-world examples of overfitting: `https://stats.stackexchange.com/questions/128616/whats-a-real-world-example-of-overfitting`
  - Also note the comments and discussions

Overfitting and Neural Networks

⚠ Overfitting is not a purely technical problem – no purely technical solution

Classical machine learning

- Feature selection can avoid relying on irrelevant features
- But this is only one source for overfitting

Overfitting and Neural Networks

⚠ Overfitting is not a purely technical problem – no purely technical solution

Classical machine learning

- Feature selection can avoid relying on irrelevant features
- But this is only one source for overfitting

Neural networks are overfitting machines

- Layered architecture $\Rightarrow$ Any relation between x and y can be learned
  - including a fixed set of if/else rules

Techniques against overfitting (besides critical thinking and use of brain)

- Regularization
- Dropout

Subsection 1

Regularization

Intuition

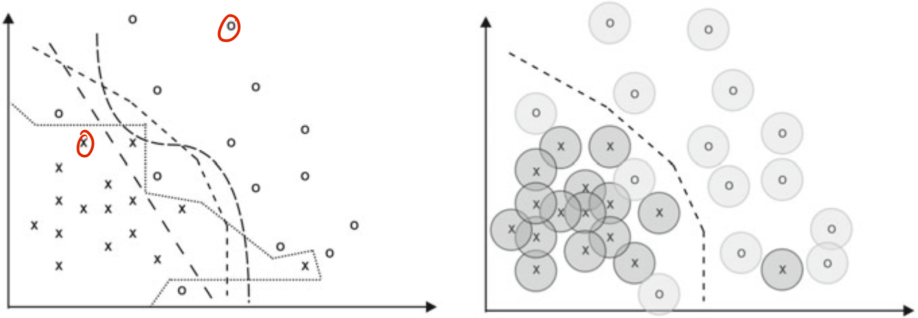


Figure: Visual representation of regularization results (Skansi, 2018, 108)

Formalization

- Formally, regularization is a parameter added to the loss

$$J(\vec{w}) = J_{\text{original}}(\vec{w}) + R$$

L^2-Regularization

L^2-Norm (a. k. a. Euclidean norm) Tikhonov (1963)

- Given a vector $\vec{x} = (x_1, x_2, \ldots, x_n)$,
  its L^2 norm is $L^2(\vec{x}) = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = ||\vec{x}||_2$

L^2-Regularization

L^2-Norm (a. k. a. Euclidean norm) Tikhonov (1963)

- Given a vector $\vec{x} = (x_1, x_2, \dots, x_n)$,
  its L^2 norm is $L^2(\vec{x}) = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = ||\vec{x}||_2$
- In practice, we drop the square root and calculate L^2 norm of the weight vector during training:

$$(||\vec{w}||_2)^2 = \sum_{i=0}^{n} w_i^2$$

L^2-Regularization

L^2-Norm (a. k. a. Euclidean norm) Tikhonov (1963)

- Given a vector $\vec{x} = (x_1, x_2, \ldots, x_n)$,
  its L^2 norm is $L^2(\vec{x}) = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = ||\vec{x}||_2$
- In practice, we drop the square root and calculate L^2 norm of the weight vector during training:

$$(||\vec{w}||_2)^2 = \sum_{i=0}^{n} w_i^2$$

- Regularization rate λ: Factor that expresses how much we want (another hyperparameter)

$$J(\vec{w}) = J_{\text{original}}(\vec{w}) + \frac{\lambda}{n}||w||_2^2 \qquad \text{with } n \text{ for the batch size}$$

L_2-Regularization

- What does it do?

L_2-Regularization

- What does it do?
  - If weights $\vec{w}$ are large: Loss is increased more
  - Large weights are only considered if the increased loss is »worth it«, i.e., if it is counterbalanced by a real error reduction
  - Small weights are preferred

Subsection 2

Dropout

Dropout

- Regularization: Numerically combatting overfitting
- Dropout: Structurally combatting overfitting Hinton et al. (2012)

Dropout

- Regularization: Numerically combatting overfitting
- Dropout: Structurally combatting overfitting — Hinton et al. (2012)
  - A new hyperparameter $\pi = [0; 1]$
  - In each epoch, every weight is set to zero with a probability of π

[Dropout] prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors. Instead, each neuron learns to detect a feature that is generally helpful for producing the correct answer given the combinatorially large variety of internal contexts in which it must operate.

Hinton et al. (2012, 1)

Dropout

Example

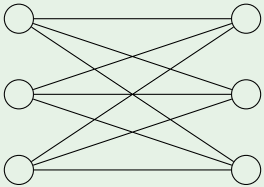


Figure: Dropout $\pi = 0.5$, visualized

Dropout

Example

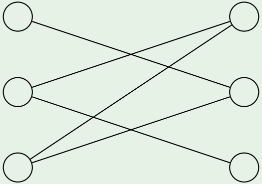


Figure: Dropout $\pi = 0.5$, visualized, Epoch 0

Dropout

Example

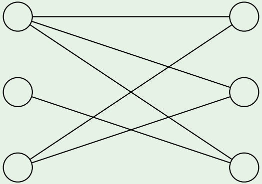


Figure: Dropout $\pi = 0.5$, visualized, Epoch 1

Dropout

Example

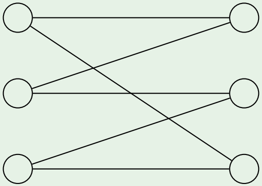


Figure: Dropout $\pi = 0.5$, visualized, Epoch 2

Section 3

Sequence Labeling

Motivation

- Language works sequentially
  - Word meaning depends on context

Motivation

- Language works sequentially
  - Word meaning depends on context
- Feedforward neural networks
  - One instance at a time
  - E.g., one sentence with four tokens ➔ positive/negative

Motivation

- Language works sequentially
  - Word meaning depends on context
- Feedforward neural networks
  - One instance at a time
  - E.g., one sentence with four tokens ➔ positive/negative
- Conceptually not adequate for natural language
- Length of influencing context varies

Motivation

- Language works sequentially
  - Word meaning depends on context
- Feedforward neural networks
  - One instance at a time
  - E.g., one sentence with four tokens ➔ positive/negative
- Conceptually not adequate for natural language
- Length of influencing context varies
- Recurrent neural networks are one solution to this problem

Sequence Labeling

- So far: Classification
- Sequence labeling
  - Special case of classification
  - Instances are organized sequentially and not independent of each other
    - I.e.: The prediction of a class for one item influences the next

Sequence Labeling

- So far: Classification
- Sequence labeling
  - Special case of classification
  - Instances are organized sequentially and not independent of each other
    - I.e.: The prediction of a class for one item influences the next

Example (Part of speech tagging)

- »the dog barks« $\rightarrow$ »DET NN VBZ«
- Predicting »DET VBZ NN« is extremely unlikely, because verbs usually don't follow determiners

Towards Recurrent Neural Networks

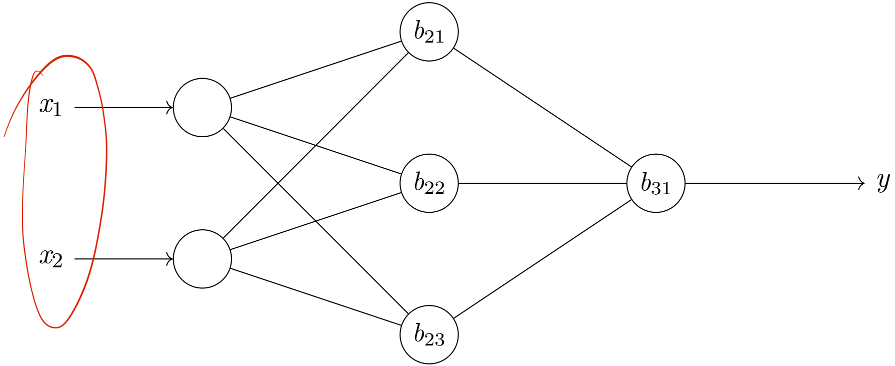


Figure: A feedforward neural network with 1 hidden layer (same picture as before)

Towards Recurrent Neural Networks

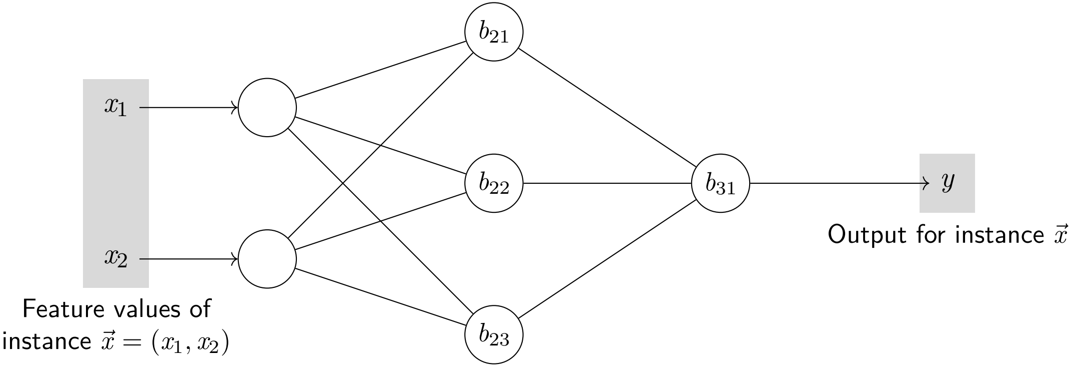


Figure: A feedforward neural network with 1 hidden layer (same picture as before)

Towards Recurrent Neural Networks

To work with sequences, we need to include the sequence into the model

Notation

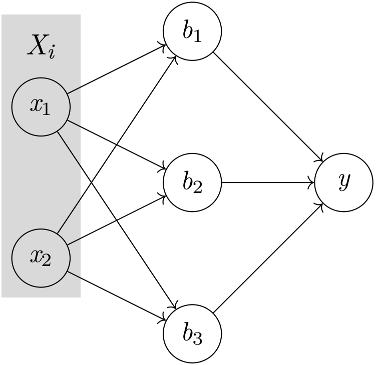
$X = (\vec{X}_1, \vec{X}_2, \ldots)$ The input data set containing a sequence of instances (e.g., a sequence of words)

$\vec{X}_i = (x_1, x_2, \ldots)$ One instance with feature values (e.g., embedding dimensions)

Y_i Output for instance X_i

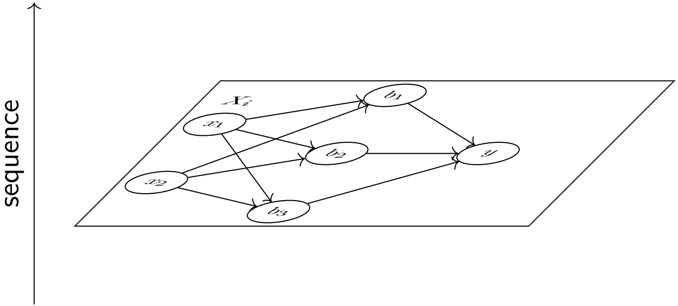
Recurrent Neural Networks

Example



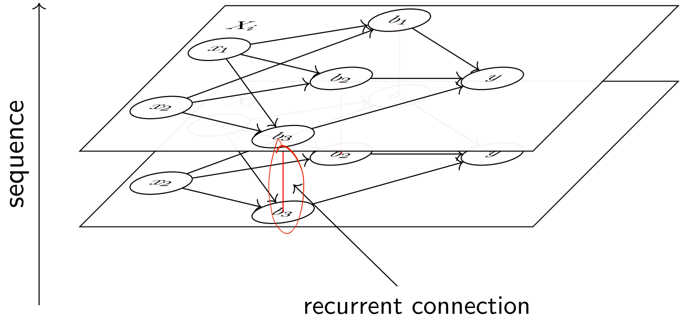
Recurrent Neural Networks

Example



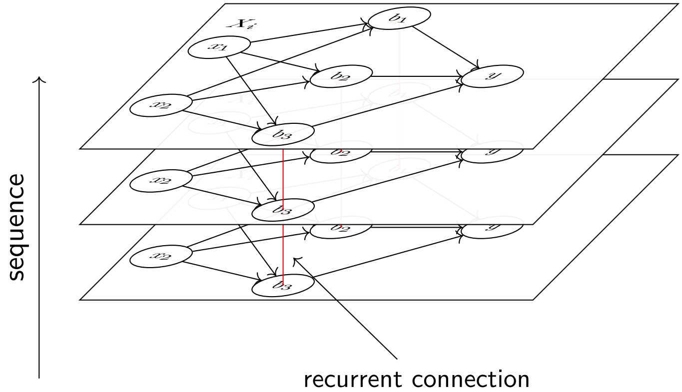
Recurrent Neural Networks

Example



Recurrent Neural Networks

Example



Recurrent Neural Networks

- FFNN, CNN: Weights between neurons
- RNN: Additional weights for recurrent connections

Recurrent Neural Networks

- FFNN, CNN: Weights between neurons
- RNN: Additional weights for recurrent connections

Input shape

- Before: Network gets at one object at a time, potentially with multiple features
- Now: Network gets sequence of objects at a time, each one potentially with multiple features
- RNN layers need 2D input:
  - Length of input sequences (if needed, padded)
  - Number of features (dimensions)
    - (this is where embeddings would go)
- For training, we need multiple sequences, making the training data 3D

Demo

- Simple task: Learn to count distances
  - Given a sequence of 1s and 0s, predict a 1 two steps after an input-1
  - E.g.: »010010001« becomes »000100100«
  - Model has to learn to count the distance
  - Training data can easily be generated

Demo

- Simple task: Learn to count distances
  - Given a sequence of 1s and 0s, predict a 1 two steps after an input-1
  - E.g.: »010010001« becomes »000100100«
  - Model has to learn to count the distance
  - Training data can easily be generated

demo

Implementation in keras

- `tf.keras.layers.SimpleRNN`
  - Documentation: `https://keras.io/api/layers/recurrent_layers/simple_rnn/`
    Selected parameters:
  - `recurrent_dropout=0.0` Dropout for recurrent links
  - `return_sequences=False` Wether to continue the network with the entire sequence or just the last element

```
model.add(layers.SimpleRNN(...))
```

BIO Scheme

- POS-Tagging is easy, because structurally simple: Each token is assigned to one class
- Named entity recognition (and many other tasks) is complicated
  - Not every token is part of a named entity (NE)
  - Many named entities span multiple tokens
  - We distinguish NEs based on the ontological type of the referent
    - PERson, ORGanization, LOCation, ...

–	–	+	+	+	+	–	–	–	–
Gestern	hat	Andreas	Müller	Sarah	Friedrichs	ein	Buch	geliehen	.
O	O	B	I	B	I	O	O	O	O
		B-PER	I-PER	B-PER	I-PER				

LOC

ORG

BIO Scheme

- POS-Tagging is easy, because structurally simple: Each token is assigned to one class
- Named entity recognition (and many other tasks) is complicated
  - Not every token is part of a named entity (NE)
  - Many named entities span multiple tokens
  - We distinguish NEs based on the ontological type of the referent
    - PERson, ORGanization, LOCation, …
- BIO scheme
  - How to represent NE annotations token-wise
  - Each token gets a label
    - B: Beginning of a NE
    - I: Inside of a NE
    - O: Outside of a NE (the majority of tokens)

B-PER I-PER I-PER

B-ORG I-ORG I-ORG I-ORG

Die Oswald-von-Wolkenstein-Gesellschaft

B-LOC

BIO Scheme

- POS-Tagging is easy, because structurally simple: Each token is assigned to one class
- Named entity recognition (and many other tasks) is complicated
  - Not every token is part of a named entity (NE)
  - Many named entities span multiple tokens
  - We distinguish NEs based on the ontological type of the referent
    - PERson, ORGanization, LOCation, …

B-PER
I-PER
B-LOC
O

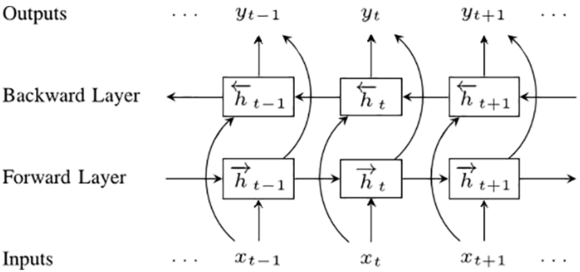
- BIO scheme
  - How to represent NE annotations token-wise
  - Each token gets a label
    - B: Beginning of a NE
    - I: Inside of a NE
    - O: Outside of a NE (the majority of tokens)
- Why B: Marking the beginning allows to recognize multiple multi-word NEs in direct sequence
  - »…hat Peter Paulus Maria Müller geküsst« → »O B-PER I-PER B-PER I-PER O«

Directions

- In a regular RNN, the sequence is processed in one direction
- Simple extension: two recurrent layers for both directions

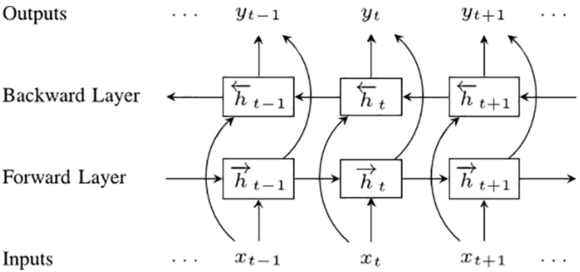
Directions

- In a regular RNN, the sequence is processed in one direction
- Simple extension: two recurrent layers for both directions



Directions

- In a regular RNN, the sequence is processed in one direction
- Simple extension: two recurrent layers for both directions




```
model.add(layers.Bidirectional(layers.SimpleRNN(...)))
```

References I

Hinton, Geoffrey E./Nitish Srivastava/Alex Krizhevsky/Ilya Sutskever/Ruslan R. Salakhutdinov (2012). *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv: 1207.0580 [cs.NE].

References II

Roberts, Michael/Derek Driggs/Matthew Thorpe/Julian Gilbey/Michael Yeung/Stephan Ursprung/Angelica I. Aviles-Rivero/Christian Etmann/Cathal McCague/Lucian Beer/Jonathan R. Weir-McCall/Zhongzhao Teng/Effrossyni Gkrania-Klotsas/Alessandro Ruggiero/Anna Korhonen/Emily Jefferson/Emmanuel Ako/Georg Langs/Ghassem Gozaliasl/Guang Yang/Helmut Prosch/Jacobus Preller/Jan Stanczuk/Jing Tang/Johannes Hofmanninger/Judith Babar/Lorena Escudero Sánchez/Muhunthan Thillai/Paula Martin Gonzalez/Philip Teare/Xiaoxiang Zhu/Mishal Patel/Conor Cafolla/Hojjat Azadbakht/Joseph Jacob/Josh Lowe/Kang Zhang/Kyle Bradley/Marcel Wassin/Markus Holzer/Kangyu Ji/Maria Delgado Ortet/Tao Ai/Nicholas Walton/Pietro Lio/Samuel Stranks/Tolou Shadbahr/Weizhe Lin/Yunfei Zha/Zhangming Niu/James H. F. Rudd/Evis Sala/Carola-Bibiane Schönlieb/AIX-COVNET (2021). »Common pitfalls and recommendations for using machine learning to detect and prognosticate for COVID-19 using chest radiographs and CT scans«. In: *Nature Machine Intelligence* 3.3, pp. 199–217. DOI: 10.1038/s42256-021-00307-0.

References III

Skansi, Sandro (2018). *Introduction to Deep Learning*. Undergraduate Topics in Computer Science. Cham: Springer.

Tikhonov, A. N. (1963). »Solution of incorrectly formulated problems and the regularization method«. In: *Soviet Math* 4, pp. 1035–1038.