

# Deep Learning-Framework PyTorch und Anwendungen in der Medizin



Linda Stemmler, Tim Hubert, Patrice Bender, Rico Zieger, Sebastian Grüb

# Vorstellung & Motivation

# Zeitplan

09:00 Uhr	Motivation
09:15 Uhr	Einführung Python / PyTorch
10:30 Uhr	Kaffeepause
10:45 Uhr	Neuronale Netze Teil 1
12:00 Uhr	Mittagspause
12:45 Uhr	Neuronale Netze Teil 2
14:00 Uhr	Verschiedene Frameworks
14:10 Uhr	Kaffeepause
14:20 Uhr	Anwendungsbeispiele in der Medizin
14:35 Uhr	Retrospektive

# Das Team

Linda Stemmle, Tim Hubert, Patrice  
Bender, Rico Zieger, Sebastian Grüb

*Gruppenbild entfernt*



Menti.com

 **Mentimeter**

Code → 44 46 51

# Was ist PyTorch?

- eine auf Maschinelles Lernen ausgerichtete Open-Source-Bibliothek
- für Programmiersprache Python
- basiert auf der Bibliothek Torch
- erstes Release war im Oktober 2016
- entwickelt von Facebooks Forschungsteam für künstliche Intelligenz

# Wieso PyTorch?

- große Community
- zwei herausstechende Merkmale:
  - mit GPUs beschleunigte Tensor-Analysen
  - Deep Neural Networks auf Basis eines Autograd-Systems
- dient als numpy Ersatz wenn GPU Beschleunigung gewünscht ist
- bewährte Bibliotheken lassen sich problemlos verwenden:
  - numpy, scipy, cython etc.
- sehr flexibel und hohe Performance
  - Beispiel: CPU 25 min. , GPU 1 min.

# Motivation - Wieso überhaupt Deep Learning in der Medizin?

Künstliche neuronale Netze helfen, das Gehirn zu

01.03.2003 | Heimbeatmung | Ausgabe

Künstliche neuronale Netze  
von Heimbeatmung

01.11.2003 | Trends und Medizinökonomie |  
Künstliche neuronale Netze

Theorie und  
und K

Zeitschrift: Monatsschrift

Autor:

G. Wol

KÜNSTLICHE INTELLIGENZ IN DER DIAGNOSTIK

## KI könnte Heilungschancen massiv verbessern

von Andreas Menn  
11. September 2018

Medizin

3 Kommentare

Monatsschrift  
Kinder



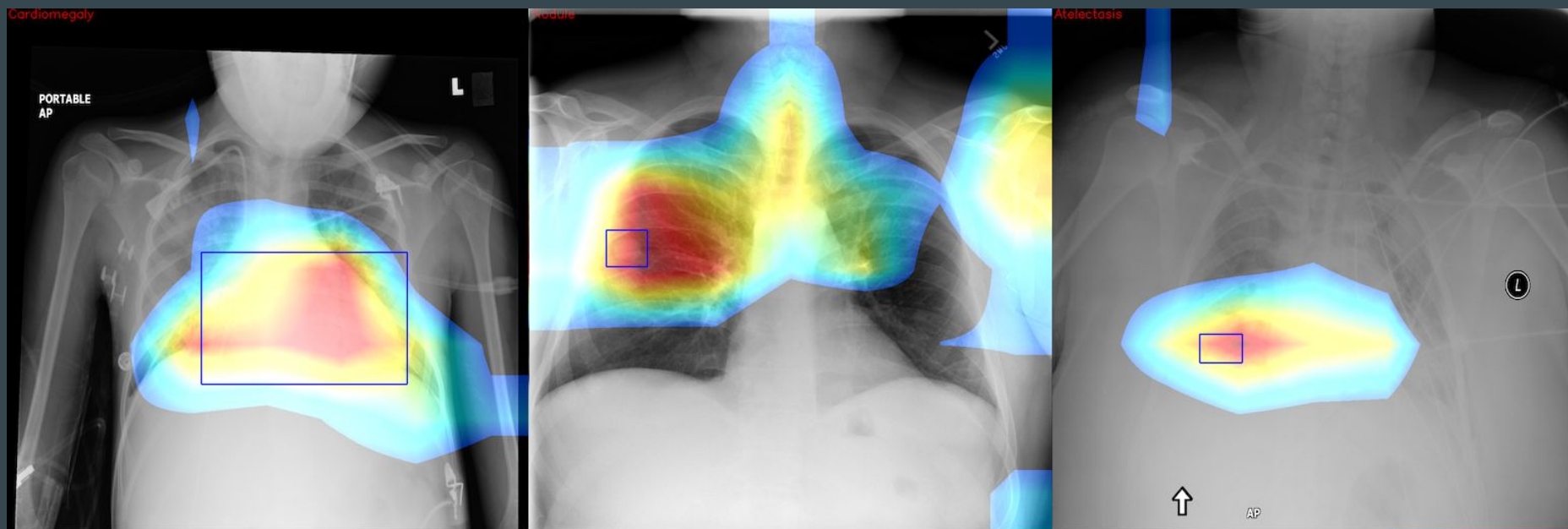
M. Morin, C. Putzke, H. Wulf, PD Dr. L. H. J.



# Motivation - Wieso überhaupt Deep Learning in der Medizin?

- Diagnostik
- Epidemiologie
- Biometrie
- Entscheidungsunterstützung
- Prognosen

# ChexNet



# ChexNet

- trainiert auf ChestX-ray14
  - 112.120 Röntgenbilder
- 121 Schichten
- Datensatz mit 420 Bildern verwendet um Netz zu trainieren
- in Zusammenarbeit mit vier Radiologen der Uni Stanford



## Input

Chest X-Ray Image

## CheXNet

121-layer CNN

## Output

Pneumonia Positive (85%)



# Einführung in Python & PyTorch

# Python

- Scriptsprache
  - Script -> ByteCode -> Interpreter
- objektorientiert
- dynamische Typisierung
- automatische Speicherverwaltung

# In clojure wird alles geklammert, in Python (fast) nichts

```
print("Namenskonventionen 1: 404 - camelNotFound" )  
print("Namenskonventionen 2: python_is_a_snake" )
```

```
if language == "clojure":  
    use_brackets_for_everything()  
elif language == "Python":  
    use_indents()  
    use_snake_case()  
else:  
    print("Nebenbei wisst ihr jetzt, wie man ein if-elif... schreibt" )
```

# Schleifen in Python - while

```
grund_studium = True
while grund_studium:
    print("Wo ist die if-Schleife?" )
    print("Noten zählen erst im Hauptstudium" )
```

# Schleifen in Python - for i in range

```
for i in range(5):  
    # 0 to 4 [5 is not included]  
    print(i)
```

```
for i in range(len(liste)):  
    # iterate using the list's length  
    do_something(liste[i])
```



# Schleifen in Python - for element in collection

```
elements = [0,1,2,3,4]
for element in elements:
    print(element)
```

```
for _ in range(len(elements)):
    print("iteration")
```



codewars

<https://www.codewars.com/collections/pytorch-workshop>

# numpy.ndarray

- (effiziente) Implementierung mehrdimensionaler (n-dimensionaler) Arrays
  - basierend auf C Arrays
- mathematische Grundrechenarten (und weitere Operatoren)
- Initialisierung
- Auswertung

# numpy - Arrays initialisieren

```
import numpy as np
```

```
shape = (2, 2)
```

```
np.zeros(shape)          # => [[0,0],[0,0]]
```

```
value = 5
```

```
np.full(shape, value)    # => [[5,5],[5,5]]
```

```
np.array([[5,5],[5,5]])  # Python-Liste zu Array
```

```
np.arange(4).reshape(shape) # [0,1,2,3] wird zu [[0,1],[2,3]]
```

# numpy - Arrays auswerten

```
my_arr = np.arange(6).reshape((2, 3)) # =>
```

```
axis=0  0 1 2  
axis=1  3 4 5  
axis=1
```

```
np.argmax(my_arr)           # = 5 <=> Index von max für eindimensionales Array  
np.argmax(my_arr, axis=0)   # = [1,1,1] <=> Index von max entlang axis=0 (Spalten)  
np.argmax(my_arr, axis=1)   # = [2,2] <=> Index von max entlang axis=1 (Reihen)
```

```
my_arr.max()                # 5  
my_arr.max(axis=0)          # [3, 4, 5]  
my_arr.min()                # 0  
my_arr.mean()               # 2,5 <=> Mittelwert
```

# numpy - Python Index Slicing

```
# 4x4 Array mit Werten 0-15
arr4x4 = np.arange(16).reshape((4, 4))

# Bereich in Zeile [0;2) und Spalte [0;2)
topLeft = arr4x4[0:2, 0:2]

# Bereich in Zeile [0;2) und Spalte [2;4)
topRight = arr4x4[0:2, 2:4]
```

[[ 0	1	2	3]
[ 4	5	6	7]
[ 8	9	10	11]
[12	13	14	15]]

[[0	1]
[4	5]]

[[2	3]
[6	7]]



codewars

<https://www.codewars.com>

# PyTorch - torch.Tensor

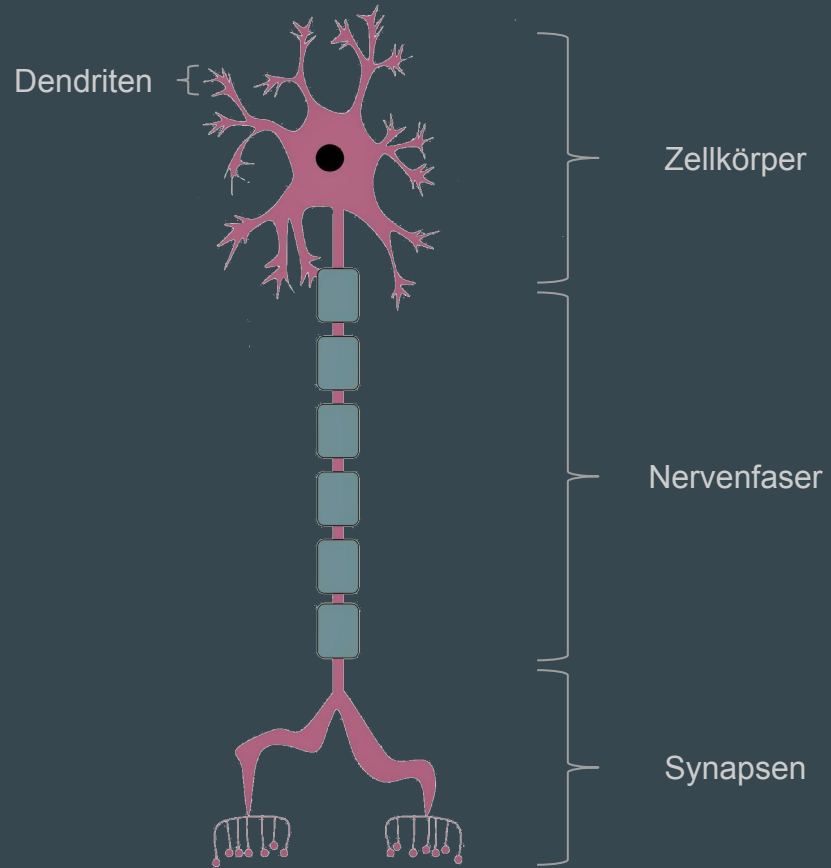
- basiert auf numpy Arrays
  - verwendet numpy API weitgehend wieder
- automatisches Differenzieren (Autograd)
  - verfolgt Änderungen eines Tensors
  - Tensor + Operation = neuer Tensor (mit Operationsreferenz)
  - ist Optional  $\Leftrightarrow$  muss explizit aktiviert werden
- kann auf GPUs berechnet werden



# Einführung in Neuronale Netze

# Neuronen in der Biologie

1. An den Dendriten kommen Signale von anderen Neuronen an. (Durch die dort angedockten Synapsen)
2. Die Signale werden zu **einem** Signal aufsummiert und über die Nervenfasern an die Synapsen weitergeleitet.
3. Die Synapsen sind wieder an den Dendriten anderer Neuronen angedockt und leiten die Signale weiter.

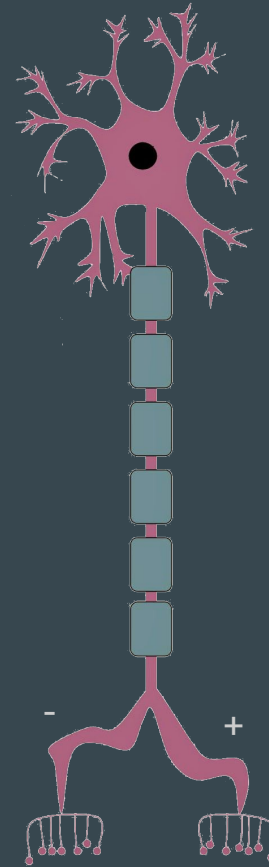


# Neuronen in der Biologie

Vereinfacht gesagt gibt es erregende und hemmende Synapsen, die die summierte Signalstärke aller Eingänge positiv oder negativ beeinflussen können.

Eine Weiterleitung erfolgt jedoch nur dann, wenn die summierte Signalstärke größer einem Schwellenwert / Aktionspotenzials ist.

Binär ausgedrückt wird entweder 0 oder 1 ausgegeben, je nachdem wie groß die aufsummierte Eingabe ist.



Aufsummierung  
der Eingänge

Weiterleitung, wenn  
Schwellenwert  
überschritten ist



Frage 1

# Hands-on #1

## Aufgabe:

Bilden Sie in Python die Aktivierungsfunktion eines Neurons nach. Diese bestimmt, ob ein Neuron für einen gegebenen Input anspricht, oder nicht.

Vervollständigen Sie hierzu die Datei Hands-on 1.py

## Zur Erinnerung:

- Ein Neuron kann beliebig viele Signale als Eingaben haben
- Eine Ausgabe liegt nur an, wenn die Summe aller Eingaben einen Schwellenwert übersteigt

Als Eingaben und Ausgaben liegen an einem Neuron Spannungen an. Wir vereinfachen wie folgt:

- Eingaben sind Floats
- Ausgaben sind Binär

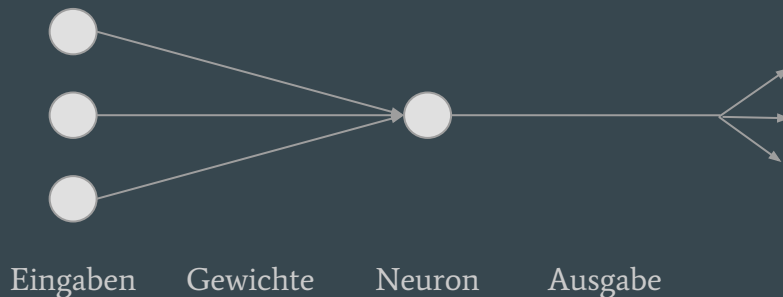
# Gewichtete Eingänge

Um zu lernen, müssen Neuronen unterschiedlich auf Eingaben reagieren können.

→ In der Biologie können Neuronen die hemmende bzw. erregende Wirkung der Synapsen anpassen.

Das bedeutet für uns:

Die Eingänge gewichten, um so einzelne Werte abschwächen oder verstärken zu können.



# Hands-on #2

Aufgabe:

Erweitern Sie nun die Aktivierungsfunktion des gegebenen Neurons um die Verwendung von Gewichten.

Lassen Sie das Neuron “lernen”, indem Sie die Gewichte manuell so anpassen, dass es nur für jeweils eine der drei Eingaben anspricht.

Vervollständigen Sie hierzu die Datei Hands-on 2.py

Zur Erinnerung:

- Jeder Eingabewert eines Neurons kann durch Gewichte abgeschwächt oder verstärkt werden.
- Ein Neuron hat immer so viele Gewichte, wie Eingabewerte



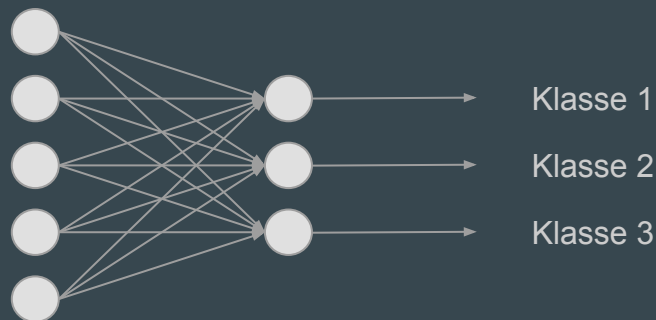
Frage 2



# Mehrere Klassen unterscheiden

Ein Neuron allein kann nur zwei Klassen unterscheiden (0 oder 1). Darüber hinaus gilt: Um  $n$  Klassen unterscheiden zu können, werden auch  $n$  Ausgabe-Neuronen benötigt.

- Eingaben liegen an allen Neuronen an.
- Alle Neuronen haben eigene Gewichte.
- Schwellenwert-Aktivierungsfunktionen sind hier oftmals nicht geeignet



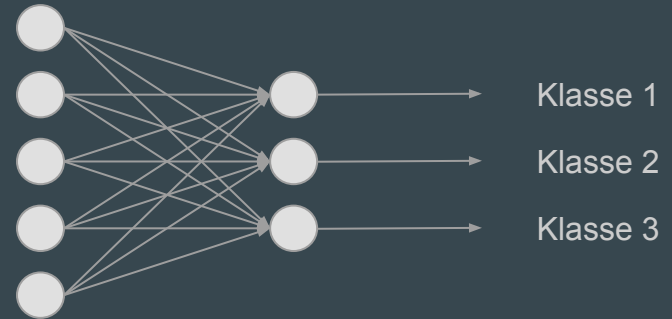
# Hands-on #3

Aufgabe:

Erstellen Sie ein einschichtiges, neuronales Netz mit fünf Eingängen sowie drei Ausgabe-Neuronen.

Vollziehen Sie die Funktionsweise des Netzes nach, indem Sie verschiedene Eingaben klassifizieren lassen. Finden Sie eine Eingabe, die der Klasse 2 zugeordnet wird?

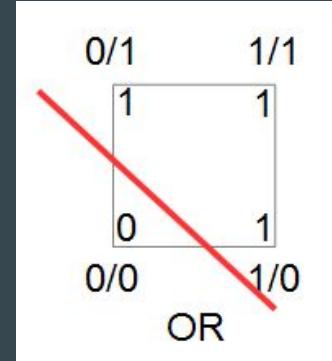
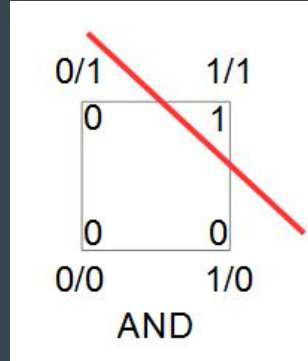
Vervollständigen Sie hierzu die Datei Hands-on 3.py



# Beschränkungen einschichtiger Netze

Bisher haben wir ein einschichtiges Netz geschaffen.

Einschichtige Netze können jedoch nur zur Unterscheidung linear separierbarer Mengen eingesetzt werden.





Frage 3

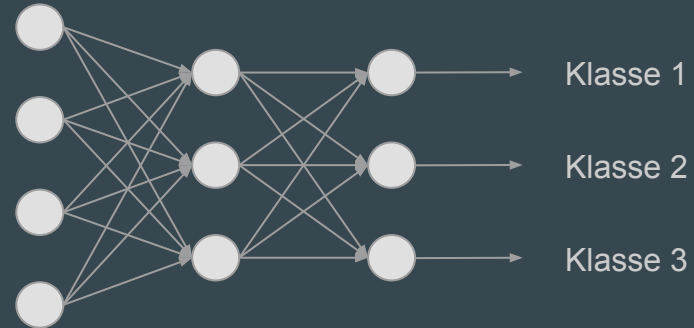
# Lösung ?

## Mehrschichtige Netze

### Zwischenschichten einführen

Ziel der zusätzlichen Schichten ist es, die Daten durch verschiedene Gewichte so zu verschieben, dass sie in der Ausgabeschicht wieder linear trennbar sind.

Die lineare Algebra zeigt, dass bereits zwei Schichten ausreichen, um beliebig komplex getrennte Daten linear trennbar zu machen. Je nach Problem ist das aber nicht zwingend die beste / performanteste Lösung.



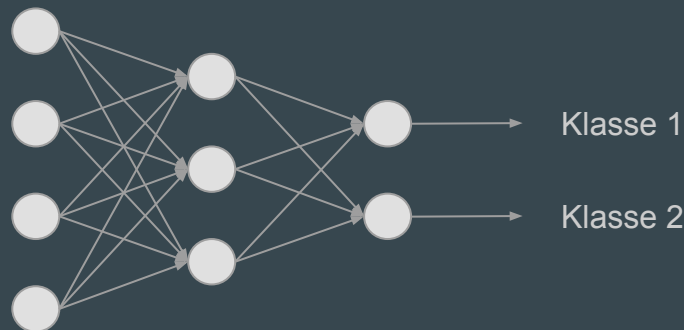
# Hands-on #4

Aufgabe:

Erstellen Sie eine Klasse “Layer”, die eine Schicht eines mehrschichtigen, neuronalen Netzes darstellt und beliebige Dimensionen einnehmen kann.

Verwenden Sie diese Klasse, um ein mehrschichtiges Netz mit vier Inputs, drei Hidden-Neuronen sowie zwei Ausgabe-Neuronen zu erstellen.

Vervollständigen Sie hierzu die Datei Hands-on 4.py



# Neuronale Netze trainieren

Bisher haben wir die Gewichte eines Netzes von Hand gesetzt.

→ Netze sollen die Gewichte aber selbst durch Training lernen.

Lernen ist nichts anderes, als der Versuch, den Fehler, den das Netz macht, zu minimieren.

→ Der Fehler lässt sich numerisch durch sogenannte Fehlerfunktionen beschreiben

Bsp.: Quadratische Fehlerfunktion

- Ein Fehler liegt vor, wenn die Ausgabe nicht der erwarteten Ausgabe entspricht
- Fehler sollten nicht negativ sein → Differenz quadrieren

Es gibt darüber hinaus viele weitere Fehlerfunktionen, wie z.B. Cross Entropy, Negative Log Likelihood, ...

# Neuronale Netze trainieren

Nachdem wir den Fehler durch eine Funktion beschreiben können, versuchen wir diese zu minimieren.

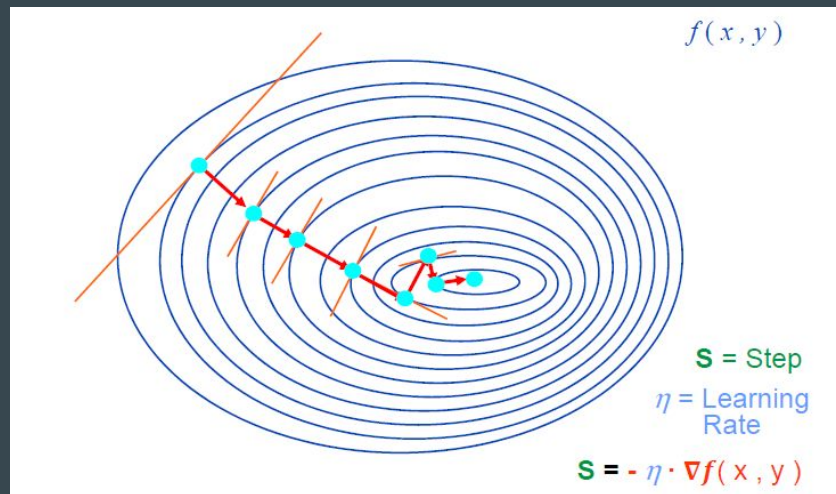
→ Dazu wird in der Regel eine Form des Gradientenabstiegsverfahrens eingesetzt

## Zur Erinnerung:

Der Gradient (die Ableitung) einer Funktion zeigt in die Richtung des größten Anstiegs.

→ Mal Minus 1 und sie zeigt in Richtung der größten Abnahme

Typischerweise wird noch eine Lernrate eingeführt, um zu verhindern, dass das Minimum übersprungen wird.





# Neuronale Netze trainieren

Für das Gradientenabstiegsverfahren müssen wir die Ableitung einer Funktion berechnen

→ In diesem Fall unsere Fehlerfunktion

Wir wollen wissen, wie die Gewichte angepasst werden müssen, damit der Fehler kleiner wird.

→ Ableitung nach den Gewichten

Durch Anwendung der Kettenregel ergeben sich daraus im Wesentlichen folgende Schritte:

- Der Fehler der Ausgabeschicht = Ausgabewert - Zielwert
- Der Fehler von Hidden-Schichten = Fehler der vorigen Schicht \* Gewichte der vorigen Schicht \*  
Ableitung Aktivierungsfunktion für den Ausgabewert
- Das Gewichtsdelta für eine Schicht = Fehler der Schicht \* Eingaben der Schicht

Nach dieser Vorgehensweise trainiert man ein Netz so lange, bis der Fehler klein genug, oder die gewünschte Anzahl an Durchläufen erreicht ist.

# Hands-on #5

## Aufgabe:

Setzen Sie nun das Training eines mehrschichtigen, neuronalen Netzes mit 2 Eingabewerten, 2 Hidden-Neuronen sowie 2 Ausgabe-Neuronen gemäß diesen Lernregeln um.

Vervollständigen Sie hierzu die Datei Hands-on 5.py

## Zur Erinnerung:

- Fehler Ausgabe-Neuronen =  $\text{Ausgabewert} - \text{Zielwert}$
- Fehler Hidden-Neuronen = Fehler vorige Schicht \* Gewichte vorige Schicht \* Ableitung Aktivierungsfunktion für den Ausgabewert
- Gewichtsdelta = Fehler der Schicht \* Eingaben der Schicht

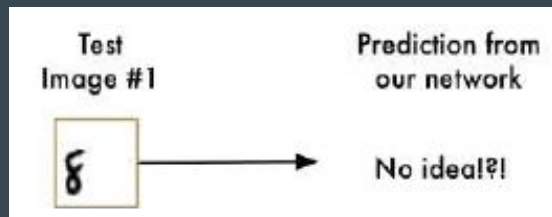
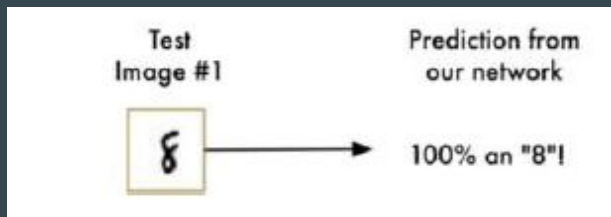
# Convolutional Networks

Wir wollen ein neuronales Netz verwenden, um Bilddaten zu trainieren.

Wie könnte das funktionieren?

→ Jeder Pixel ist eine Eingabe von 0-255 (z.B. der Weiß-Anteil bei Schwarz-Weiß Bildern)

Dabei stehen wir jedoch vor folgendem Problem:



→ Verwendung einer speziellen Variante von neuronalen Netzen, den **Convolutional Networks**

# Convolutions

Zunächst werden die Eingaben / Bilder mithilfe einer Faltungsmaske geglättet (Convolution)

1	2	7	0	3
5	0	1	4	0
9	5	1	5	7
4	1	6	3	3
2	9	3	1	2

Eingaben

1	2	1
2	4	2
1	2	1

Gaußmaske

	2,75	2,5		

Ausgaben

Summe von Maske x Pixelwerte

Summe der Maske

Neuer Wert für grünen Pixel =  $(1*2+2*7+1*0+2*0+4*1+2*4+1*5+2*1+1*5) : (1+2+1+2+4+2+1+2+1) = 40:16 = 2,5$

Es gibt keine klassischen Gewichte mehr; Die Faltungsmaske wird trainiert.

# Hands-on #6

Aufgabe:

Erstellen Sie eine Funktion, die für einen gegebenen Input sowie mit einer gegebenen Faltungsmaske eine Convolution durchführt.

Überprüfen Sie ihre Funktion mithilfe der gegebenen Ein- und Ausgabedaten.

Vervollständigen Sie hierzu die Datei  
Hands-on 6.py

# Pooling

Anschließend an eine Convolution findet i.d.R. ein Pooling statt.

→ Üblicherweise wird dafür ein **Max-Pooling** verwendet

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

6	8
3	4

Ein Filter (hier der Größe 2 bzw. 2x2) wird über das Bild bewegt und der jeweils maximale Wert bestimmt.

→ Diese Werte zusammen bilden dann das Ergebnis

Somit wird die Anzahl an Parametern / Pixeln reduziert und die Eingabe wird schrittweise generalisiert.

(Es ist jetzt egal, ob das Maximum oben rechts oder unten links steht.)

# Hands-on #7

Aufgabe:

Erstellen Sie eine Funktion, die ein Max-Pooling für eine gegebene Eingabe sowie eine gegebene Filtergröße realisiert.

Überprüfen Sie ihre Funktion mithilfe der gegebenen Ein- und Ausgabedaten.

Vervollständigen Sie hierzu die Datei Hands-on 7.py



Frage 4



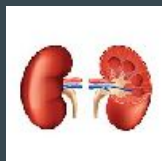
# Erstellung eines Klassifikators mit PyTorch

# Auswahl des Klassifikators



Klassifikator für komplexe Aufgaben wie ChexNet?

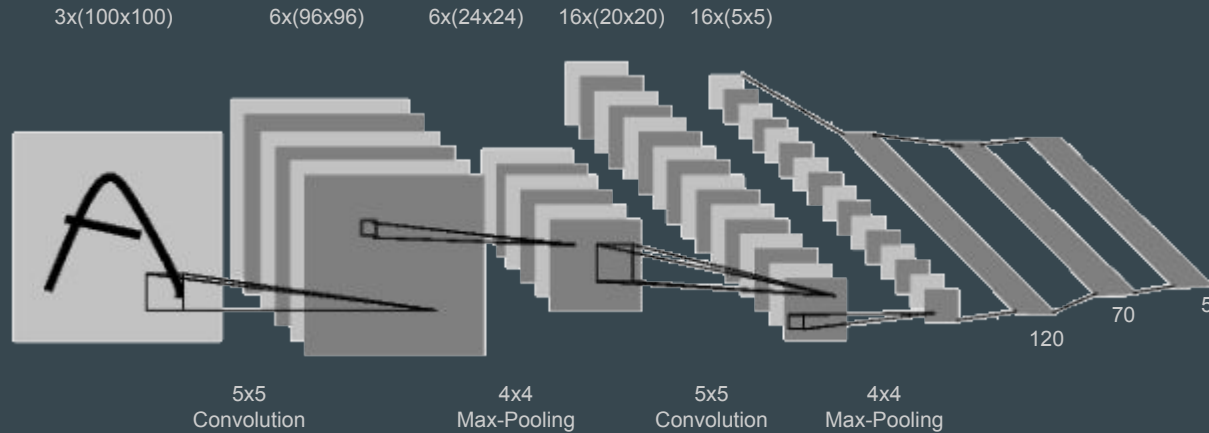
- Training dauert sehr lange (1 Monat)
  - Training erfordert sehr viele Trainingsdaten
- Für diesen Workshop nicht geeignet



Einfacher Organ-Klassifikator

- Unterscheidung von Leber, Lunge, Darm, Magen und Nieren
  - Bilder zeigen ausschließlich das jeweilige Organ
  - Funktionsweise aber prinzipiell die gleiche, wie bei komplexeren Aufgaben
- Lässt sich im Rahmen des Workshops umsetzen

# Aufbau des Klassifikators



Der Aufbau unseres Klassifikators ist dem des LeNet, einem der ersten bekannten Klassifikatoren, nachempfunden.

# Umsetzung mit PyTorch 1

Netz aufbauen mit:

- nn.Conv2d für Convolution-Schichten
- nn.MaxPool2d für Max-Pooling-Schichten
- nn.Linear für “normale” Schichten

Nach den Convolutions mittels view die Ausgabe (Bilder) in ein Array überführen

→  $5 \times 5 \Rightarrow 1 \times 25$

Linear-Schichten liefern die reine Netzwerkschritte ohne Anwendung einer Aktivierungsfunktion.

→ muss separat darauf angewendet werden.

```
1 import torch.nn as nn
2 import torch.nn.functional as F
3
4 class MyClassifier(nn.Module):
5     def __init__(self):
6         super(MyClassifier, self).__init__()
7         self.conv = nn.Conv2d(1, 2, 3)
8         self.pool = nn.MaxPool2d(2, 2)
9         self.fc = nn.Linear(10, 5)
10
11     def forward(self, x):
12         x = self.pool(self.conv(x))
13         x = x.view(-1, 10)
14         x = F.relu(self.fc(x))
15         return x
```

# Umsetzung mit PyTorch 2

```
1 import torch.nn as nn
2 import torch.optim as optim
3
4 net = MyClassifier()
5
6 criterion = nn.CrossEntropyLoss()
7 optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Eine Fehlerfunktion wählen, z.B. CrossEntropyLoss

→ eignet sich für Klassifikationsaufgaben besser als die quadratische Fehlerfunktion

Ein Optimierungsverfahren wählen, z.B. StochasticGradientDescent (SDG)

Beim Training:

1. Auf dem Ergebnis der Fehlerfunktion die Ableitung mittels `.backwards()` bestimmen
2. Mittels `.step()` passt das Optimierungsverfahren dann die Parameter so an, dass der Fehler minimiert wird

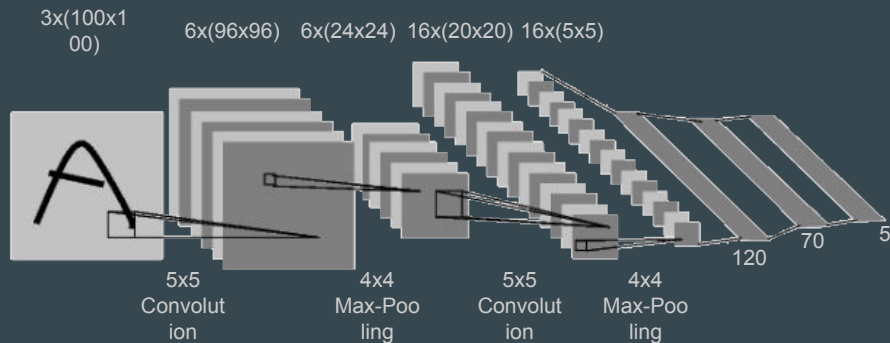
# Hands-on #8

Aufgabe:

Implementieren Sie nun den vorgestellten Klassifikator.

Trainieren Sie anschließend ihren Klassifikator. Welche Genauigkeit erreicht ihr Netz nach 5, 10 bzw. 20 Trainingsdurchläufen?

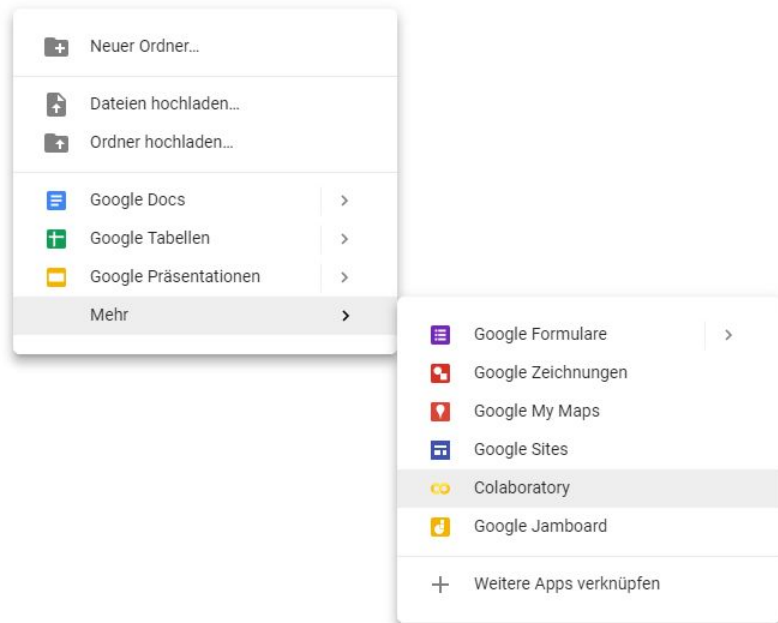
Vervollständigen Sie hierzu die Datei Hands-on 8.py



# PyTorch mit Google Colaboratory

- Ist eine Webanwendung für maschinelles Lernen, die auf Jupyter basiert.
- Anbindung einfach über Google Drive
- Erlaubt das Ausführen von Python-Code auf CPU (Xeon E-Series) oder GPU (Tesla K80)

Die Teilnehmeraccounts sind bereits mit Colab verknüpft



# Hands-on #9

Aufgabe:

Importieren Sie Ihren zuvor erstellten Klassifikator nun in Colaboratory.

Die dafür benötigten Daten finden Sie bereits in dem Drive-Speicher Ihrer Google-Accounts. Sie müssen lediglich die Pfade in Ihrem Klassifikator entsprechend anpassen. Führen Sie den Klassifikator aus, um sicherzugehen, dass alle Anpassungen korrekt vorgenommen wurden.

Vervollständigen Sie hierzu die Datei Hands-on 9.ipynb



# PyTorch mit GPU

Genauso wie man einen Tensor zur Berechnung auf die GPU verschiebt, kann man auch ein komplettes Netz auf GPU auslagern.

Dann müssen beim Training des Netzes aber auch die Eingaben und Zielwerte auf die GPU verschoben werden.

Als Ergebnis erhält man in diesem Fall aber einen neuen Tensor, der dann wieder einer Variable zugewiesen werden muss.

```
device = torch.device("cuda:0")  
model.to(device)
```

```
mytensor = my_tensor.to(device)
```

# Hands-on #10

Aufgabe:

Passen Sie nun ihren Klassifikator so an, dass er auf einer GPU ausgeführt werden kann.

Fügen Sie anschließend außerdem eine Zeitmessung ein und lassen Sie sich ausgeben, wie lange das Training des Klassifikators auf der GPU dauert. Vergleichen Sie den Wert mit der Trainingsdauer auf einer CPU.

Erweitern Sie hierzu Ihr bestehendes Jupyter-Notebook (Hands-on 9.ipynb)

```
import time

#Anzahl Sekunden seit 01.01.1970 als Float
print(time.time())
```

# Weitere Features von PyTorch

PyTorch erlaubt sowohl das Speichern und Laden von ganzen Netzen als auch von den Parametern eines Netzes.

→ `torch.save()`, `torch.load()`

Bietet Implementierungen für viele andere Fehlerfunktionen oder Optimierungsverfahren an

→ z.b. die dynamische Anpassung der Lernrate

Bietet Implementierungen zu Dropout-Layern oder Data-Augmentation an, um Overfitting zu verhindern

→ Data-Augmentation insbesondere bei Klassifikation von Bildern, wie etwa in der Medizin wichtig

→ Erzeugt leicht veränderte Bilder aus den Original-Bildern

→ Netz kann besser lernen, worauf es bei der Klassifikation ankommt

# Hands-on #11

Aufgabe:

Erweitern Sie abschließend ihren Klassifikator, sodass die Parameter nach dem Training abgespeichert werden. Sie können diese auch direkt in Drive hochladen.

Optional:

Ändern Sie testweise die Fehlerfunktion des Netzes, z.B. auf NLLLoss oder fügen Sie dem Netz eine Dropout-Schicht hinzu. Wie beeinflusst dies ihre Erkennungsrate?

Erweitern Sie hierzu Ihren zuvor erstellten Klassifikator

# Gegenüberstellung verschiedener Frameworks

# Was gibt es für Deep-Learning Frameworks?

PYTORCH

theano

Caffe

APACHE  
mxnet™

Caffe2

neon

Microsoft  
CNTK

Chainer

TensorFlow™

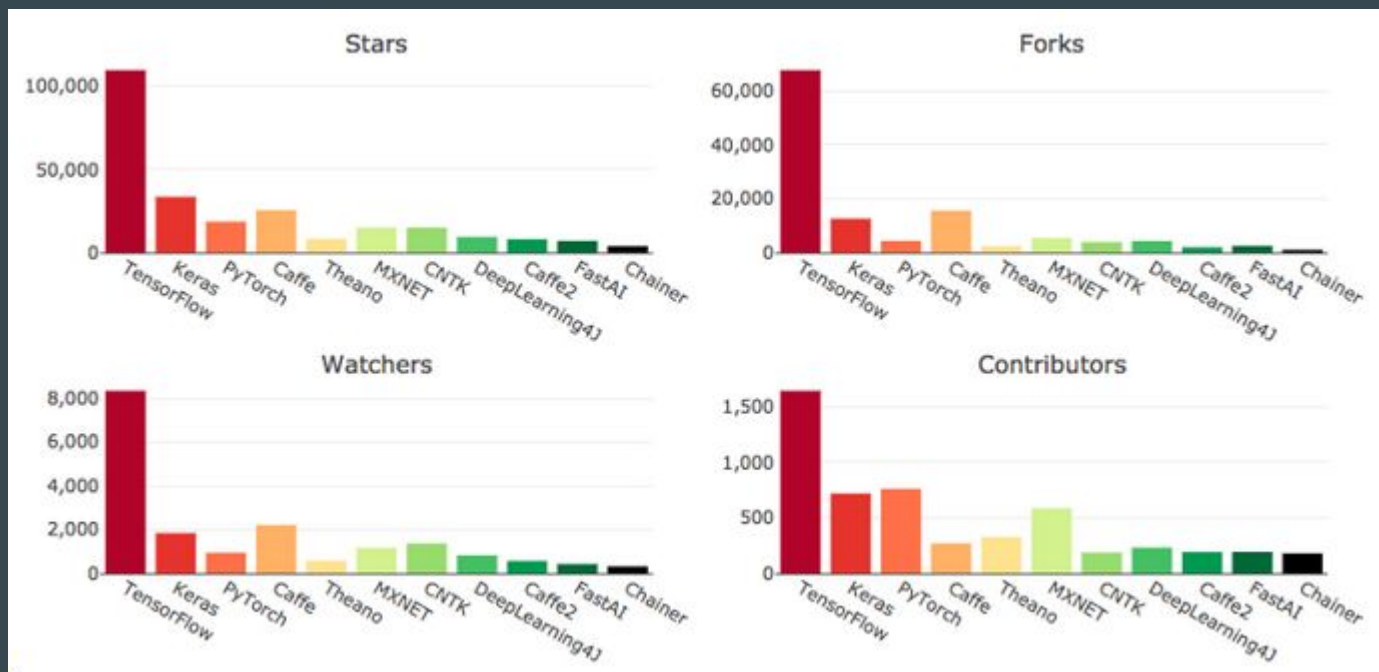
Lasagne

K Keras

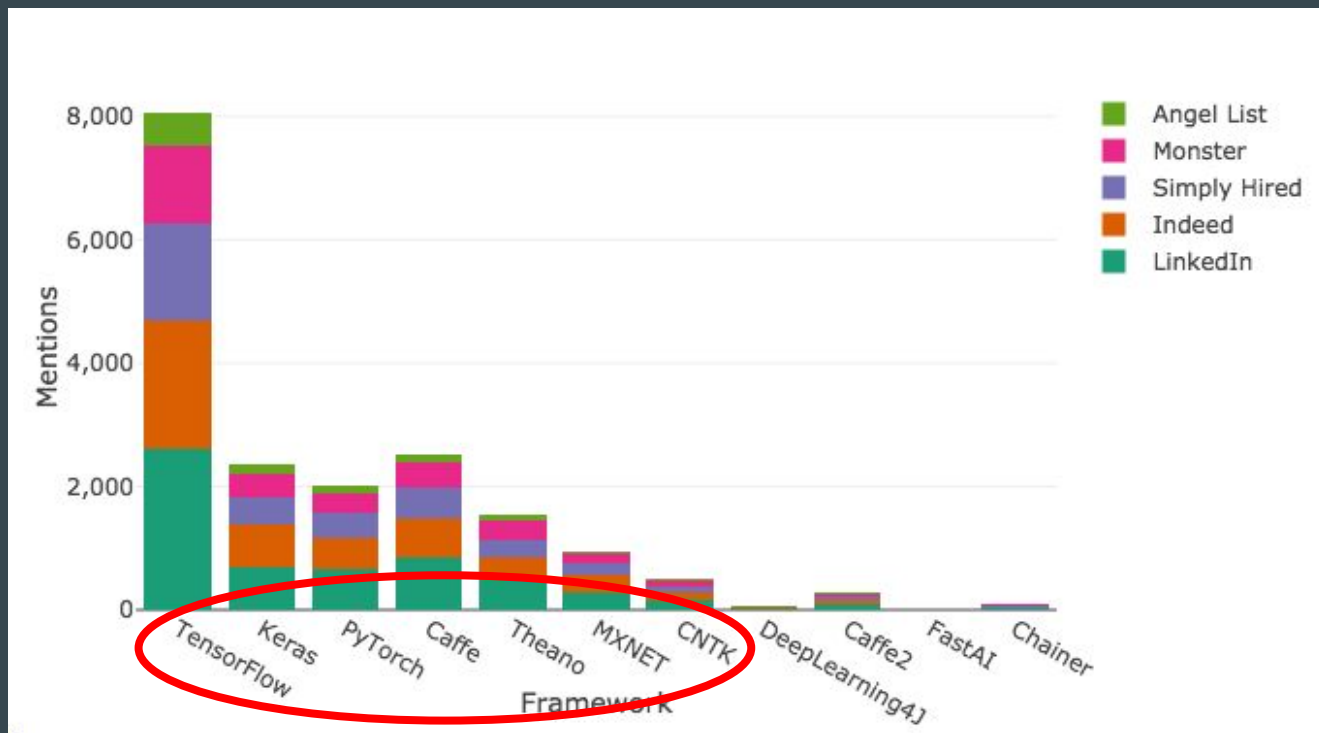


DEEPLARNING4J

# GitHub Aktivitäten



# Online-Job Listings



<https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a>



# PyTorch



- von Facebook
- Veröffentlichung: 2016
- open source
- basiert auf Torch
- MedicalTorch
  - Framework für PyTorch
  - Bereitstellung umfangreicher medizinischer Tools und Datensätze
- Interface: Python

- Caffe besitzt unter anderem einen fork “Caffe-NVIDIA”
- Veröffentlichung: 2013
- schwierigere Implementierung
- Caffe2 verbessert Caffe 1.0 in folgenden Punkten:
  - umfangreicheres verteiltes Training
  - neue Hardware-Unterstützung
  - mobile Bereitstellung
  - Flexibilität bei neuen Technologien
- Interfaces: Python, C++

# MXNet



- von Apache
- u.a. Amazon verwendet MXNet für AWS
- Veröffentlichung: 2015
- schnell (Benchmark-Vergleiche für Netzwerk-Training)
- sehr flexibel einsetzbar
  - auch für low-end-Umgebungen
- “eigenartig” im Vergleich zu anderen Frameworks
- Interfaces: Python, C++, JavaScript, Go, R, Scala, Perl, Matlab, Julia

# TensorFlow



- von Google
- Veröffentlichung: 2015
- große Community
- open source
- low-level- and high-level-Interfaces für Netzwerk-Training
- Interfaces: Python (Keras), C/C++, Java, Go, JavaScript, R, Julia, Swift

# Theano



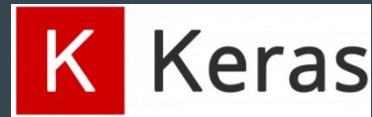
- Python-Bibliothek
- Veröffentlichung: 2007
- kann mit weiteren nützlichen Bibliotheken wie “Lasagne” verwendet werden
- Interface: Python (Keras)

# Microsoft Cognitive Toolkit (CNTK)



- von Microsoft
- Veröffentlichung: 2016
- open source
- hohe Performance
- hohe Skalierbarkeit
- Interfaces: Python (Keras), C++, Command line, BrainScript, (.NET on roadmap)

# Keras



- “on-Top” für TensorFlow, CNTK oder Theano
- Veröffentlichung: 2015
- leicht zu Lernen
- nicht flexibel einsetzbar
- Interfaces: Python, R

# Anwendungsbeispiele in der Medizin



# Alzheimer Diagnose

Erkennt Alzheimer anhand struktureller MRT-Scans des Gehirns

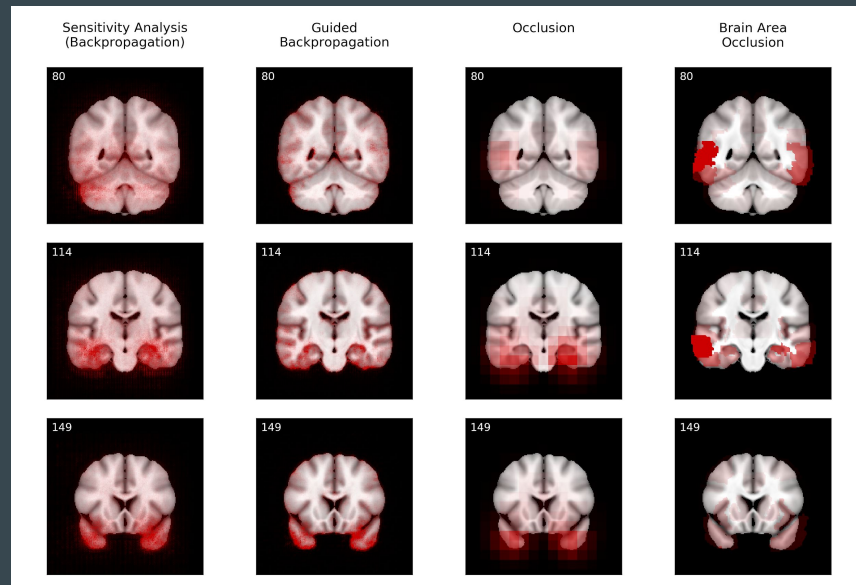
CNN (convolutional neural network) entscheidet, ob Alzheimer vorliegt.

→ Da CNN schwer zu interpretieren sind, werden 4 verschiedene Visualisierungsmethoden bereitgestellt, um Entscheidung nachzuvollziehen.

Erkennungsrate: 77%

Implementiert mit PyTorch

<https://github.com/jrieke/cnn-interpretability>



# TernaryNet

Ermöglicht die voxelgenaue Segmentierung der Bauchspeicheldrüse in CT-Scans

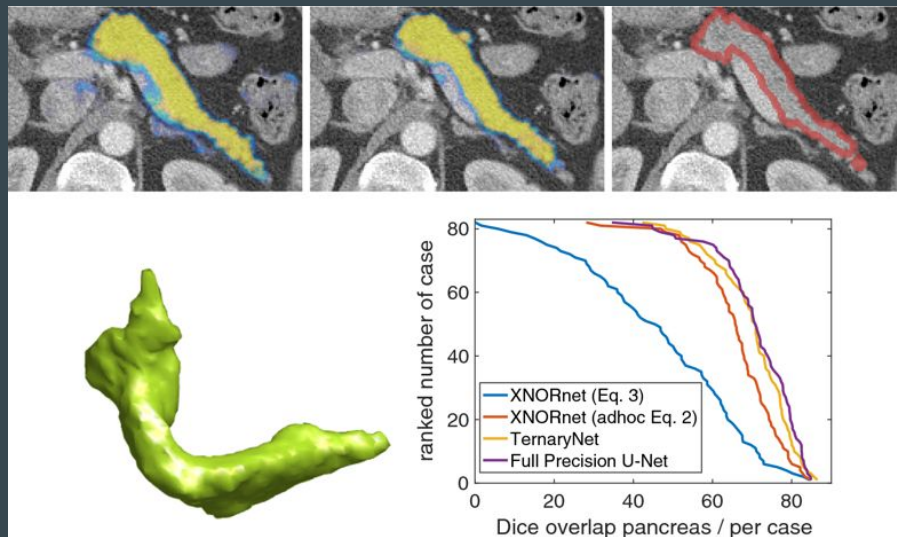
→ Für die Diagnose von Entzündungen oder Krebs

TernaryNet arbeitet schneller und ressourcenschonender als bisherige computergestützte Verfahren

71.0% Überlappung

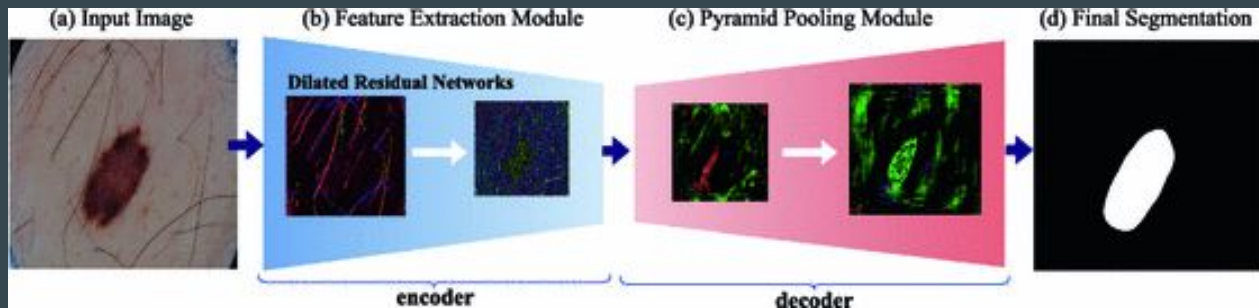
Implementiert mit PyTorch

<https://github.com/mattiaspaul/TernaryNet>



# SLSDeep

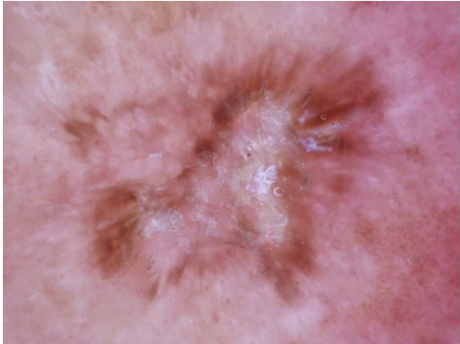
- Dient der Klassifikation von Hautläsionen (Skin lesion segmentation)
- Auch erfahrenen Dermatologen unterlaufen Fehler
  - ca. 5% aller Klassifikationen sind falsch
- Neue Fehlerfunktion durch Kombination von zwei verschiedenen Fehlerfunktionen um Ergebnis zu optimieren



# Aryan Misras Skin Lesion Classifier

- Alternatives Netz zur Erkennung von Hautkrebs-Erkrankungen
- verwendet HAM10000
- Implementiert mit Keras, bietet aber eine interessante Web-Schnittstelle, auf der man Bilder hochladen und klassifizieren lassen kann:

<https://aryanmisra.com/skinpredictapp.html>

Predictions	Image
1. akiec, Actinic Keratoses (Solar Keratoses) or intraepithelial Carcinoma (Bowen's disease): 0.990488 2. df, Dermatofibroma: 0.008266 3. bcc, Basal Cell Carcinoma: 0.001005	

# Retrospektive

# Retrospektive des heutigen Tages

- Motivation: Wieso überhaupt Deep Learning in der Medizin?
- Grundlagen Python & PyTorch
- Grundlagen Neuronale Netze
  - Funktionsweise Neuronen
  - Aufbau & Einsatz einschichtiger Netze
  - Aufbau & Einsatz mehrschichtiger Netze
  - Training mehrschichtiger Netze
  - Aufbau & Einsatz Convolutional Networks
- Überblick der meist genutzten Deep-Learning-Frameworks
- Anwendungsbeispiele in der Medizin

# Vielen Dank für Ihre Aufmerksamkeit!



# Quellen

- Codewars-Logo Folie 18, 23:
  - <https://www.codewars.com/assets/logos/logo-square-red-big-d261c083345ce22e7749dc6b69549398.png> über <https://www.codewars.com/about> (06.02.2019)
- GitHub-Aktivitäten & Online Job-Listings Folie 63f:
  - <https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a> über <https://towardsdatascience.com>
- Alzheimer Diagnose Folie 73:
  - <https://github.com/jrieke/cnn-interpretability>
- TernaryNet Folie 74:
  - <https://arxiv.org/pdf/1801.09449.pdf>
- ChexNet Folie 10f:
  - <https://stanfordmlgroup.github.io/projects/chexnet/> (06.02.2019)
  - <https://github.com/brucechou1983/CheXNet-Keras> (06.02.2019)
- SLSDeep Folie 75:
  - [https://link.springer.com/chapter/10.1007%2F978-3-030-00934-2\\_3](https://link.springer.com/chapter/10.1007%2F978-3-030-00934-2_3) (06.02.2019)
- Aryan Misra's Skin Lesion Classifier Folie 76:
  - <https://aryanmisra.com/skinpredictapp.html> (06.02.2019)
- Abschluss-Bild Folie 79:
  - [https://www.scoopnest.com/user/Independent\\_ie/596279608674246656](https://www.scoopnest.com/user/Independent_ie/596279608674246656) (06.02.2019)



# Quellen

- Überlappende Bildausschnitte Folie 8 :
  - <https://www.aerzteblatt.de/nachrichten/74352/Kuenstliche-Intelligenz-unterstuetzt-Aerzte-bei-Auswertung-von-Roentgenbildern> (06.02.2019)
  - <https://biermann-medizin.de/kuenstliche-neuronale-netze-helfen-das-gehirn-zu-kartieren/> (06.02.2019)
  - <https://www.heise.de/newsticker/meldung/Neuronales-Netz-erkennt-Herzinfarkte-so-gut-wie-erfahrene-Kardiologen-4107042.html> (06.02.2019)
  - <https://entwickler.de/online/development/machine-learning-kuenstliche-intelligenz-medizin-246715.html> (06.02.2019)
  - <https://www.springermedizin.de/kuenstliche-neuronale-netze-zur-steuerung-von-heimbeatmungsgeraet/8045664> (06.02.2019)
  - <https://www.springermedizin.de/kuenstliche-neuronale-netze/8002090> (06.02.2019)
  - <https://www.wiwo.de/technologie/forschung/kuenstliche-intelligenz-in-der-diagnostik-ki-koennte-heilungschancen-massiv-verbessern/23054930.html> (06.02.2019)
- Neuron Bild Folie 26f :
  - <https://hannahelpbiologya2.blogspot.com/2015/02/the-structure-of-myelinated-motor.html> (06.02.2019)
- Mentimeter-Logo:
  - [http://www.codlearningtech.org/wp-content/uploads/2018/06/Mentimeter\\_Logo.png](http://www.codlearningtech.org/wp-content/uploads/2018/06/Mentimeter_Logo.png) (06.02.2019)
- Gradientenabstieg Folie 40:
  - <https://na-mic.org/w/images/b/ba/Insight-Registration.ppt> (06.02.2019)
- Convolutional Networks Folie 43:
  - <https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721> (06.02.2019)

# Quellen

- Organ-Bilder Folie 50:
  - <http://www.netanimations.net/Moving-picture-breathing-lungs-animated-gif.gif> (06.02.2019)
  - <http://editimage.club/newakc37127.html> (06.02.2019)
  - <https://de.clipartlogo.com/free/kidney-dialysis.html> (06.02.2019)
- Klassifikator-Bild Folie 51:
  - Le Cun et al., Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE (1998), 86(11):2278-2324
- FFramework Logos Folie 62:
  - [https://pytorch.org/tutorials/\\_images/pytorch-logo-flat.png](https://pytorch.org/tutorials/_images/pytorch-logo-flat.png) (06.02.2019)
  - <https://www.exastax.com/wp-content/uploads/2018/02/theano-1.png> (06.02.2019)
  - <https://www.paperspace.com/images/logos/ml/caffe-logo.png> (06.02.2019)
  - <https://user-images.githubusercontent.com/591887/30987393-ba66e610-a44b-11e7-8226-da1711dc5bdc5.jpg> (06.02.2019)
  - [https://caffe2.ai/static/og\\_image\\_caffe2.png](https://caffe2.ai/static/og_image_caffe2.png) (06.02.2019)
  - <https://www.microway.com/wp-content/uploads/Deep-Learning-Frameworks-masthead.png> (06.02.2019)
  - <https://multicorewareinc.com/wp-content/uploads/cntk-logo.png> (06.02.2019)
  - <http://chainer.org/images/logo.png> (06.02.2019)
  - [https://wiki.tum.de/download/attachments/25009442/tensor-flow\\_opengraph\\_h.png?version=1&modificationDate=1485888308193&api=v2](https://wiki.tum.de/download/attachments/25009442/tensor-flow_opengraph_h.png?version=1&modificationDate=1485888308193&api=v2) (06.02.2019)
  - [https://cdn-images-1.medium.com/max/1200/1\\*4xMHPimPID74b6pGnnbgeA.png](https://cdn-images-1.medium.com/max/1200/1*4xMHPimPID74b6pGnnbgeA.png) (06.02.2019)
  - [https://l.bp.blogspot.com/-dkgTIGvckXQ/WIIMcvIC0yI/AAAAAAAAAEcU/xG\\_wTsHOx2Y6j6GiTdQTEv4umKZS0IVCwCLcBGAs/s1600/keras-logo-2018-1-arge-1200.png](https://l.bp.blogspot.com/-dkgTIGvckXQ/WIIMcvIC0yI/AAAAAAAAAEcU/xG_wTsHOx2Y6j6GiTdQTEv4umKZS0IVCwCLcBGAs/s1600/keras-logo-2018-1-arge-1200.png) (06.02.2019)
  - [https://deeplearning4j.org/assets/themes/thedocs/img/logo\\_white.png](https://deeplearning4j.org/assets/themes/thedocs/img/logo_white.png) (06.02.2019)