Neuronale Netze

Prof. Dr. Rudolf Kruse

Computational Intelligence
Institut für Intelligente Kooperierende Systeme
Fakultät für Informatik
rudolf.kruse@ovgu.de

Lernen Tiefer Neuronaler Netze (Deep Learning)

Deep Learning

Wiederholung
- Lernen Rekurrenter Netze durch *ausfalten*
- Problem des verschwindenden Gradienten

Varianten des Neurons

Autoencoder

Hybrider Lernalgorithmus

Faltung

Pooling

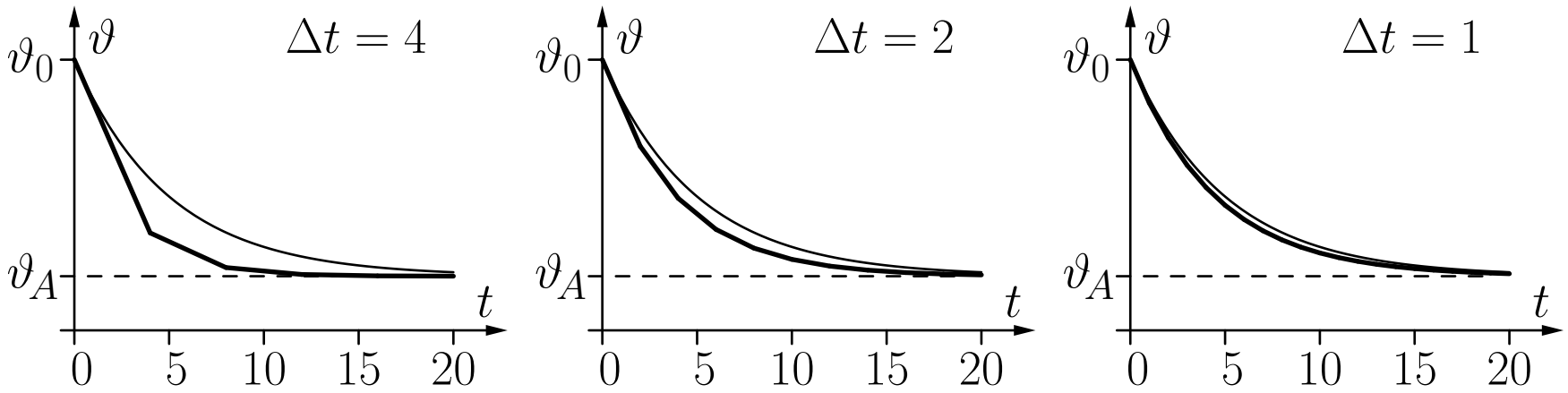
Faltende Neuronale Netze

Anwendungen

Wiederholung: Lernen Rekurrenter Netze

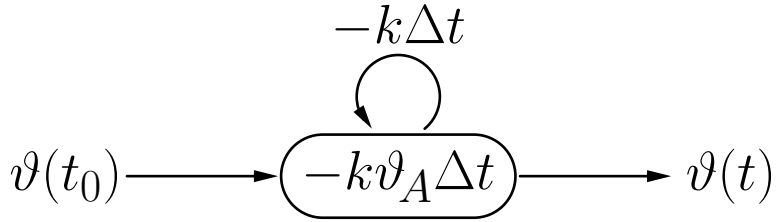
Beispiel: **Newton'sches Abkühlungsgesetz**

Euler–Cauchy-Polygonzüge für verschiedene Schrittweiten:



Die dünne Kurve ist die genaue analytische Lösung.

Rekurrentes neuronales Netz:



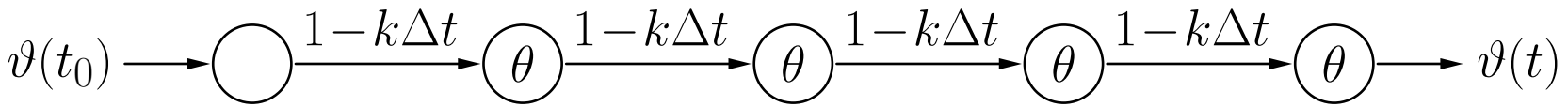
Wiederholung: Fehler-Rückpropagation über die Zeit

Annahme: Wir haben Messwerte der Abkühlung (oder Erwärmung) eines Körpers zu verschiedenen Zeitpunkten. Außerdem sei die Umgebungstemperatur ϑ_A bekannt.

Ziel: Bestimmung des Werts der Abkühlungskonstanten k des Körpers.

Initialisierung wie bei einem MLP: Biaswert und Gewicht der Rückkopplung zufällig wählen.

Die Zeit zwischen zwei aufeinanderfolgenden Messwerten wird in Intervalle unterteilt. Damit wird die Rückkopplung des Netzes *ausgefaltet*. Liegen z.B. zwischen einem Messwert und dem folgenden vier Intervalle ($t_{j+1} = t_j + 4\Delta t$), dann erhalten wir

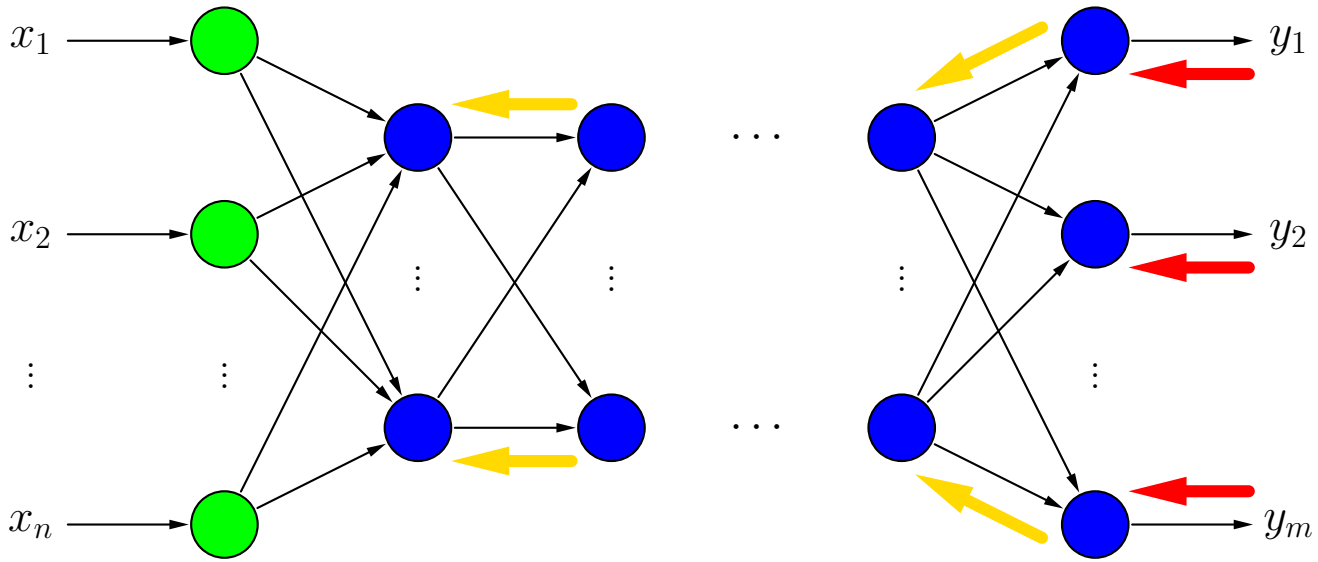


Wiederholung: Fehlerrückpropagation

$\forall u \in U_{\text{in}}:$
$\text{out}_u^{(l)} = \text{ex}_u^{(l)}$

Vorwärts-
propagation:

$\forall u \in U_{\text{hidden}} \cup U_{\text{out}}:$
$\text{out}_u^{(l)} = \left(1 + \exp\left(-\sum_{p \in \text{pred}(u)} w_{up}\, \text{out}_p^{(l)}\right)\right)^{-1}$



logistische
Aktivierungs-
funktion
impliziter
Biaswert

Rückwärts-
propagation:

$\forall u \in U_{\text{hidden}}:$
$\delta_u^{(l)} = \left(\sum_{s \in \text{succ}(u)} \delta_s^{(l)} w_{su}\right) \lambda_u^{(l)}$

Fehlerfaktor:

$\forall u \in U_{\text{out}}:$
$\delta_u^{(l)} = \left(o_u^{(l)} - \text{out}_u^{(l)}\right) \lambda_u^{(l)}$

Aktivierungs-
ableitung:

$\lambda_u^{(l)} = \text{out}_u^{(l)} \left(1 - \text{out}_u^{(l)}\right)$

Gewichts-
änderung:

$\Delta w_{up}^{(l)} = \eta\, \delta_u^{(l)}\, \text{out}_p^{(l)}$

Bisherige Probleme

Probleme:	**Lösungsansätze:**
Gewichtsänderung nimmt in vorderen Schichten exponentiell ab	Initialisiere Gewichte nicht zufällig, sondern abhängig vom Datensatz
Lernen dauert zu lang	Verwende schnellere Computer Lastverteilung auf GPUs
Zuwenige (gelabelte) Lernbeispiele vorhanden	Sammle mehr gelabelte Lernbeispiele
Konvergenz zu lokalen Minima	Kann nicht vollständig verhindert werden

Rectified Linear Unit (ReLU)

Wähle statt Neuron
Rectified Linear Unit

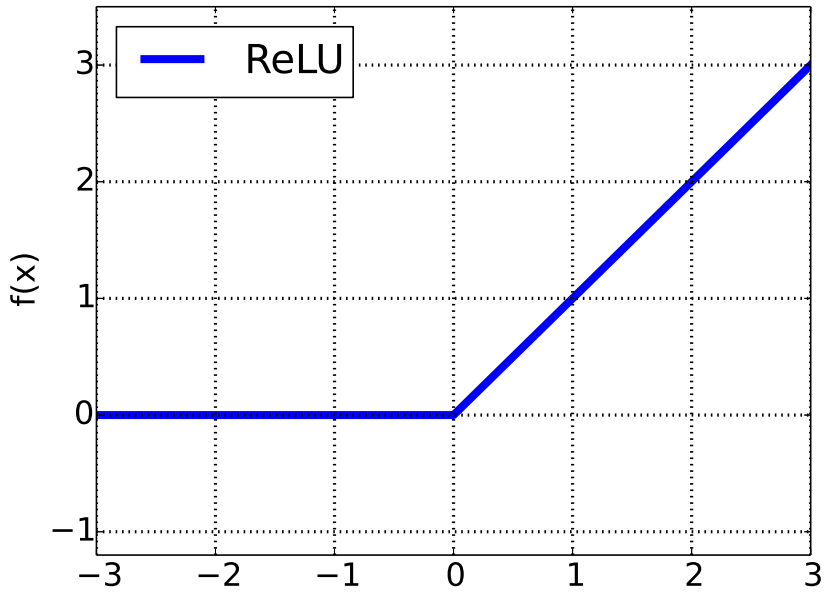
ReLU: $f(x) = max(0, x)$

Vorteile:

- sehr einfache Berechnung
- Ableitung ist leicht zu bilden
- 0-Werte vereinfachen Lernen

Nachteile:

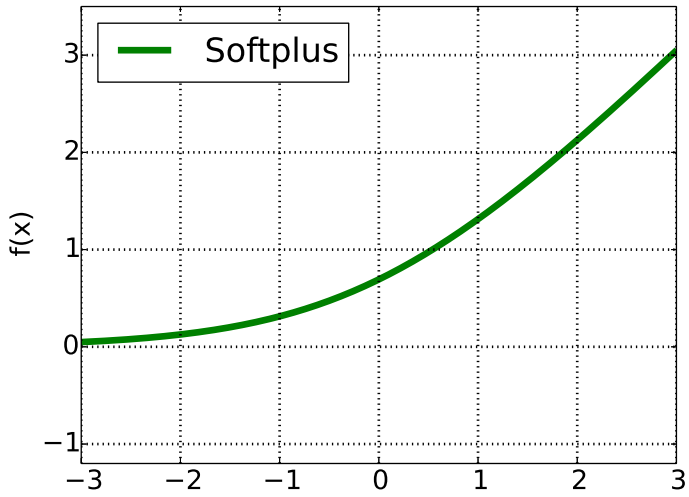
- kein Lernen links der 0
- mathematisch eher unschön
- Nicht-differenzierbarer „Knick“ bei 0



[ReLu nach Glorot et. al 2011]

ReLU: Berechnungsvarianten

Softplus:
$f(x) = ln(1 + e^x)$
- „Knick“ wurde beseitigt
- Einige Vorteile auch



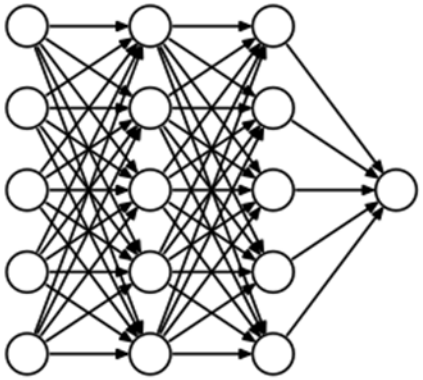
[Softplus nach Glorot et. al 2011]

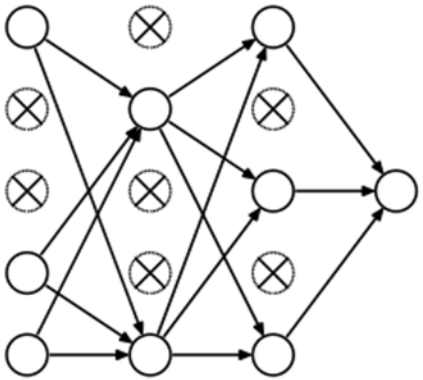
Noisy ReLU:
$f(x) = max(0, x + \mathcal{N}(0, \sigma(x)))$
- Addiert Gaussches Rauschen

Leaky ReLU
$$f(x) = \begin{cases} x, & \text{falls } x > 0, \\ 0.01x, & \text{sonst.} \end{cases}$$

Dropout

ohne Dropout





mit Dropout

Gewünschte Eigenschaft:
Robustheit bei Ausfall von Neuronen

Ansatz beim Lernen:

- Nutze nur 50% der Neuronen
- Wähle diese zufällig

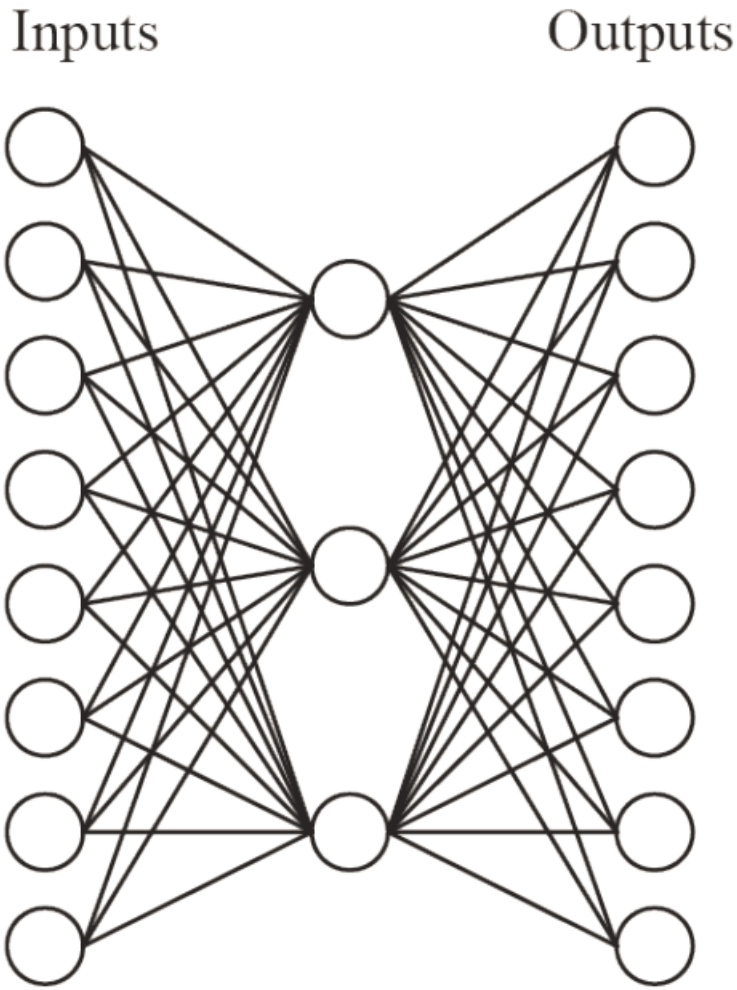
Ansatz beim Anwenden

- Nutze 100% der Neuronen
- Halbiere alle Gewichte

Ergebnis:

- Robustere Repräsentation
- Verbesserte Generalisierung

Autoencoder



Erstellt eine Kodierung der Daten

Lernt Gewichte mit Rückpropagation

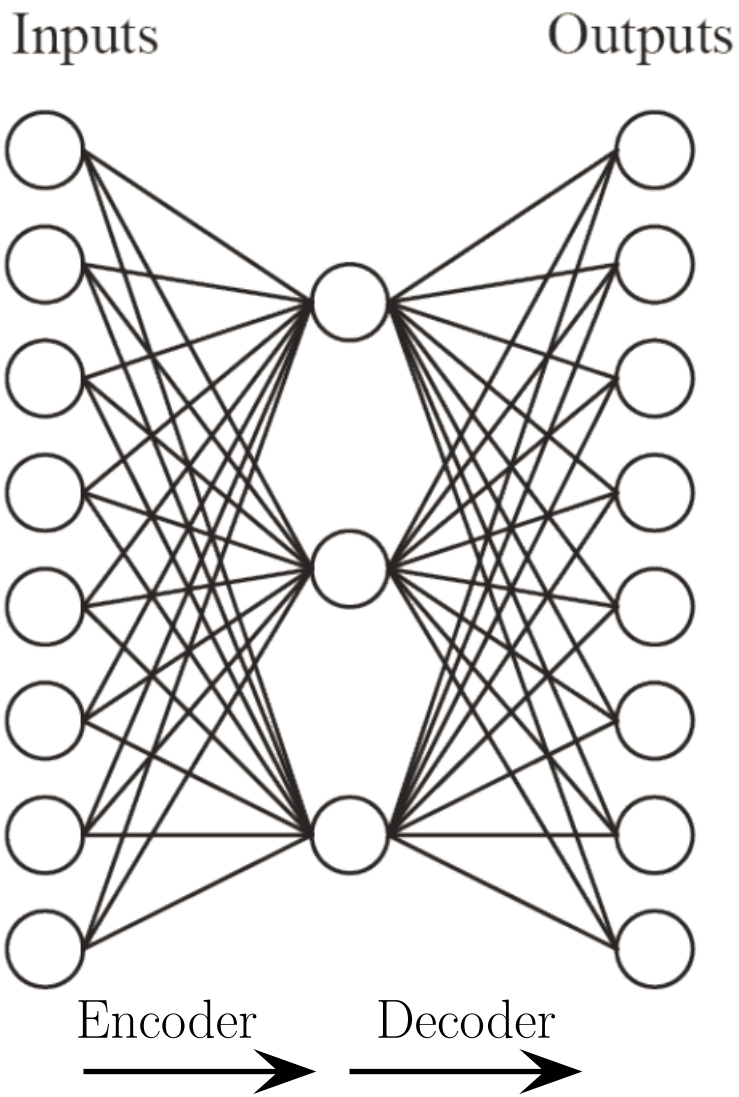
Durch **unüberwachtes** Lernen

Fehler ist $|out - in|^2$

Inputs		Versteckte Gewichte		Outputs
10000000	→	?	→	10000000
01000000	→		→	01000000
00100000	→		→	00100000
00010000	→		→	00010000
00001000	→		→	00001000
00000100	→		→	00000100
00000010	→		→	00000010
00000001	→		→	00000001

[Grafiken nach T. Mitchell, Machine Learning, McGraw Hill, 1997]

Autoencoder



Nutze für Dekodierung die transponierte Gewichtsmatrix der Encodierung

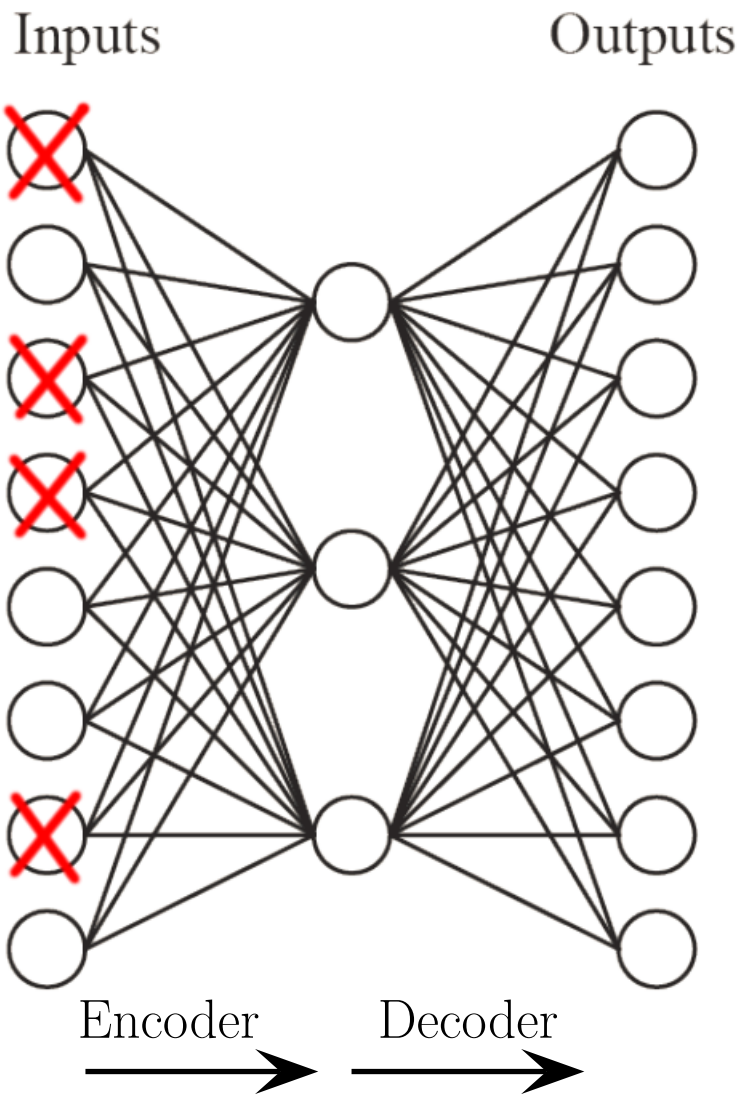
Ergebnis nach 5000 Lerniterationen:

Binäre Kodierung annähernd erreicht

Inputs		Versteckte Gewichte				Outputs
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27.	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01.	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

[Grafiken nach T. Mitchell, Machine Learning, McGraw Hill, 1997]

Rauschreduzierender (Denoising) Autoencoder



Gegeben:
eine dünne (sparse) Repräsentation

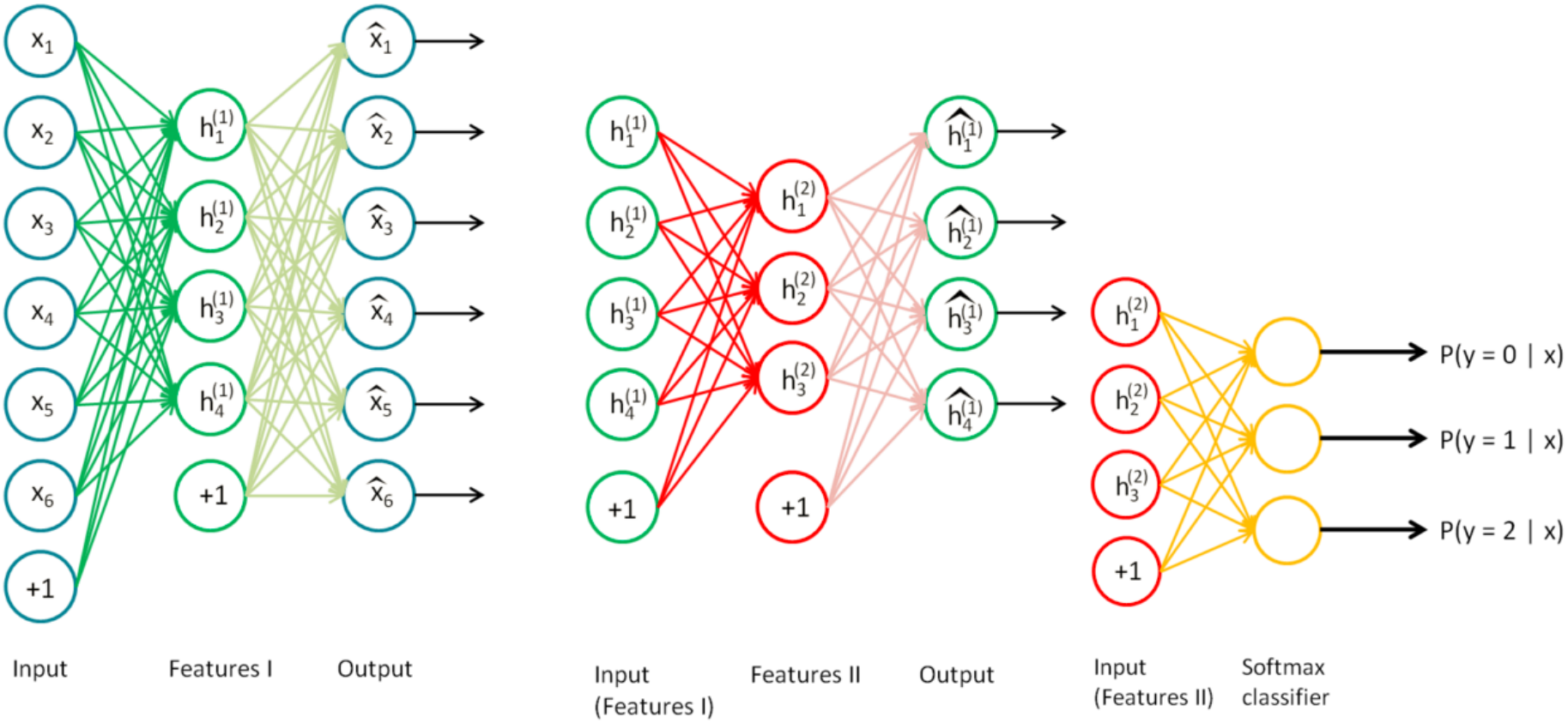
Gewünscht:
eine volle Repräsentation

Ansatz:
Kombiniere Autoencoder mit Dropout

Ergebnis:
komprimierte Darstellung
dynamisch auf Lernbeispiele zugeschnitten
Features für andere Algorithmen

Stapeln von Autoencodern

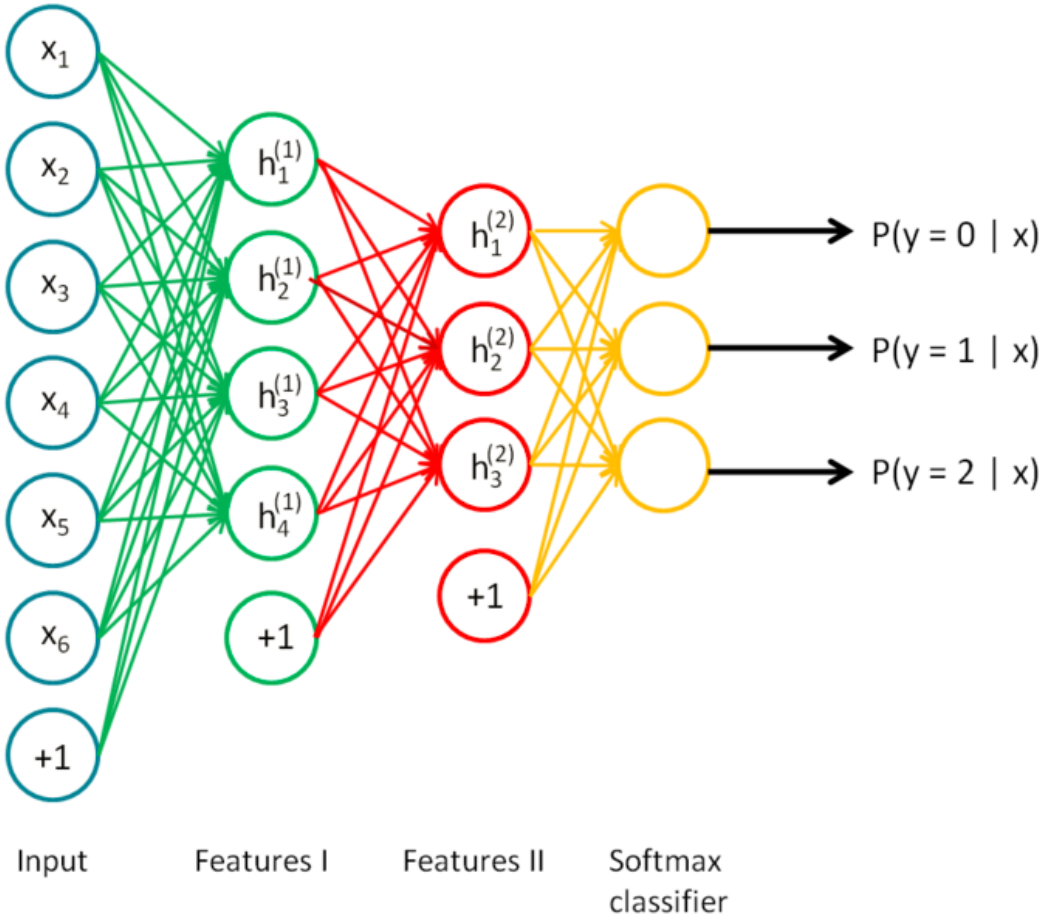
Staple Autoencoder, um die besten Features zu erhalten



[http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders]

Stapeln von Autoencodern

Nutze die (vor)gelernten Features zur Klassifikation



[http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders]

Hybrider Deep Learning Algorithmus

1. Definiere für die Lernaufgabe geeignete Netzstruktur
2. Erstelle entsprechend der Struktur Autoencoder und lasse sie mit Rückpropagation einzeln lernen
3. Verwende nur die Encoder, ihre Gewichte und eine weitere vollständig vernetzte, zufällig initialisierte Schicht zur Klassifikation
4. Lasse das so vortrainierte Netz mit Rückpropagation lernen

Problem: Objekterkennung in Bildern

Imagenet Large Scale Visual Recognition Challenge (LSVRC) seit 2010

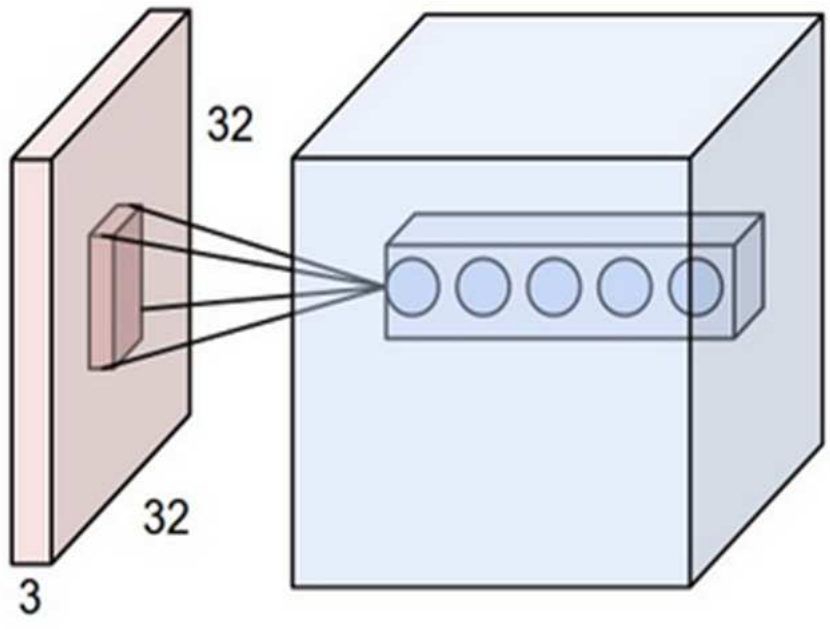
Finde 200 Objektklassen (Stuhl, Tisch, Person, Fahrrad,...)

in Bildern mit ca. 500 x 400 Pixeln, 3 Farbkanälen

Neuronales Netz mit ca. 600.000 Neuronen in der ersten Schicht

200 Neuronen in der Ausgabeschicht

Faltung (Convolution)



[Quelle]

Motivation: Egal wo auf dem Bild ein Objekt ist, soll es erkannt werden

Idee: Verwende die selben Features auf dem gesamten Bild

Umsetzung: Filter / Kernel werden auf jedem Teil des Bildes angewandt und teilen sich die Gewichte

Parameter:
Anzahl der Filter
Stärke der Überlappung

Faltung (Convolution)

Image

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter

1	0	1
0	1	0
1	0	1

Convolved Feature

4	3	4
2	4	3
2	3	4

Featuretransformation

Schiebe einen „Filter“ über die Features und betrachte die „gefilterten“ Features

Multipliziere Originalfeature mit Filter und Summiere

Originalraum: 5x5

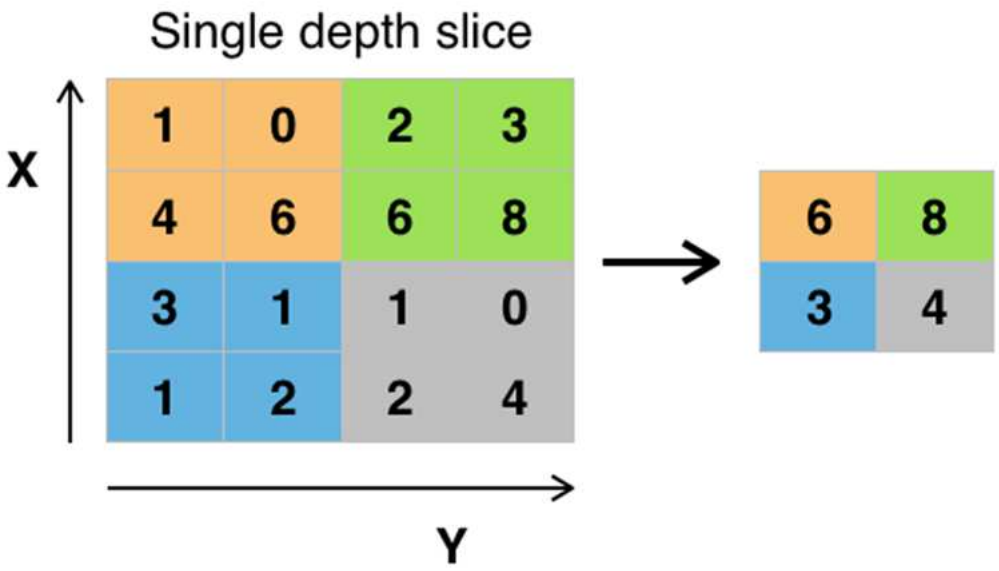
Filtergröße: 3x3

Neue Featuregröße: 3x3

Featureraum wird kleiner

[http://ufldl.stanford.edu/wiki/index.php/Feature_extraction_using_convolution]

Pooling



[Quelle]

Featuretransformation

Schiebe einen „Filter“ über die Features und betrachte die „gefilterten“ Features

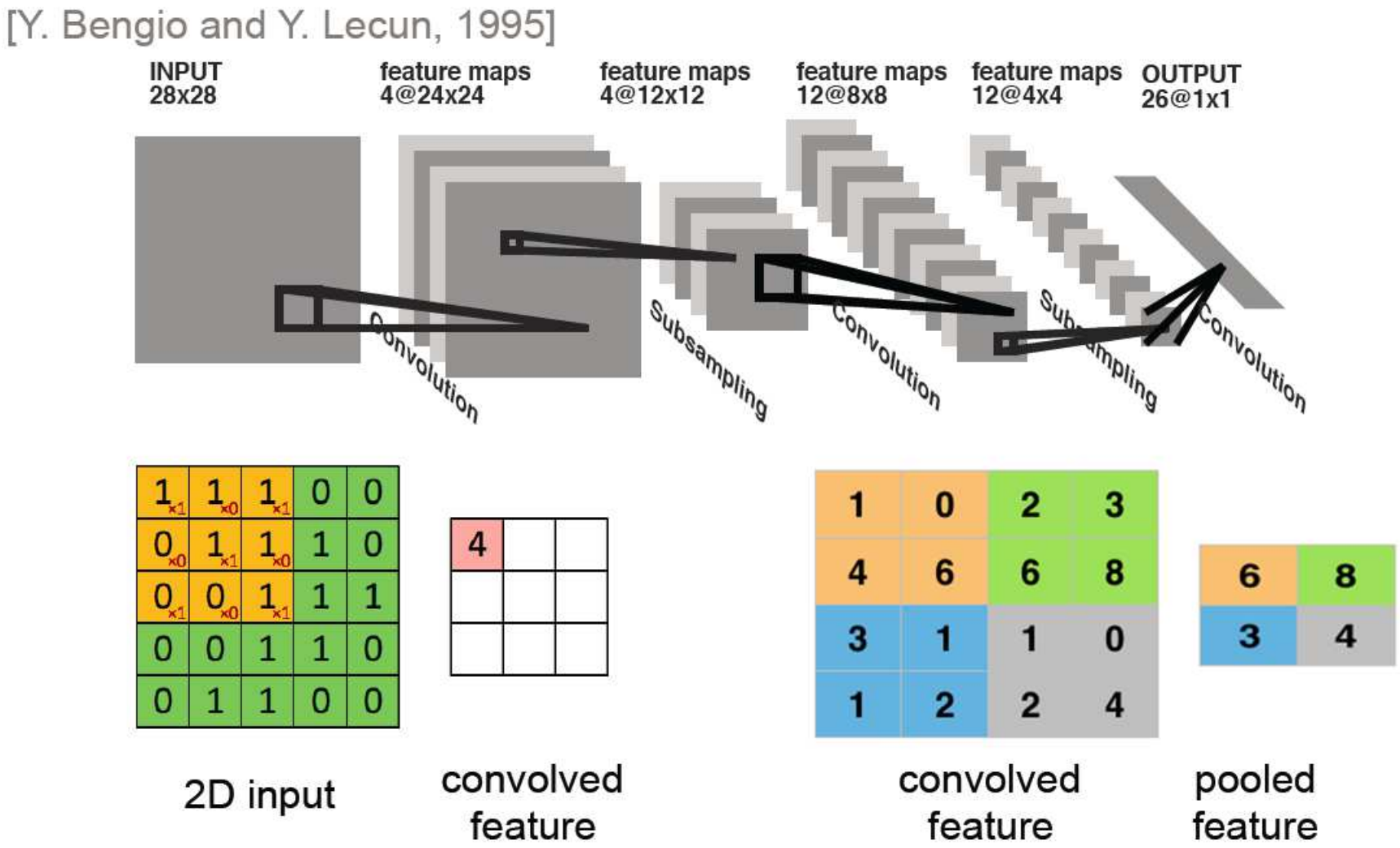
Betrachte den Bereich entsprechend der Filtergröße

Max Pooling: Nimm maximalen Wert

Mean Pooling: Nimm Mittelwert

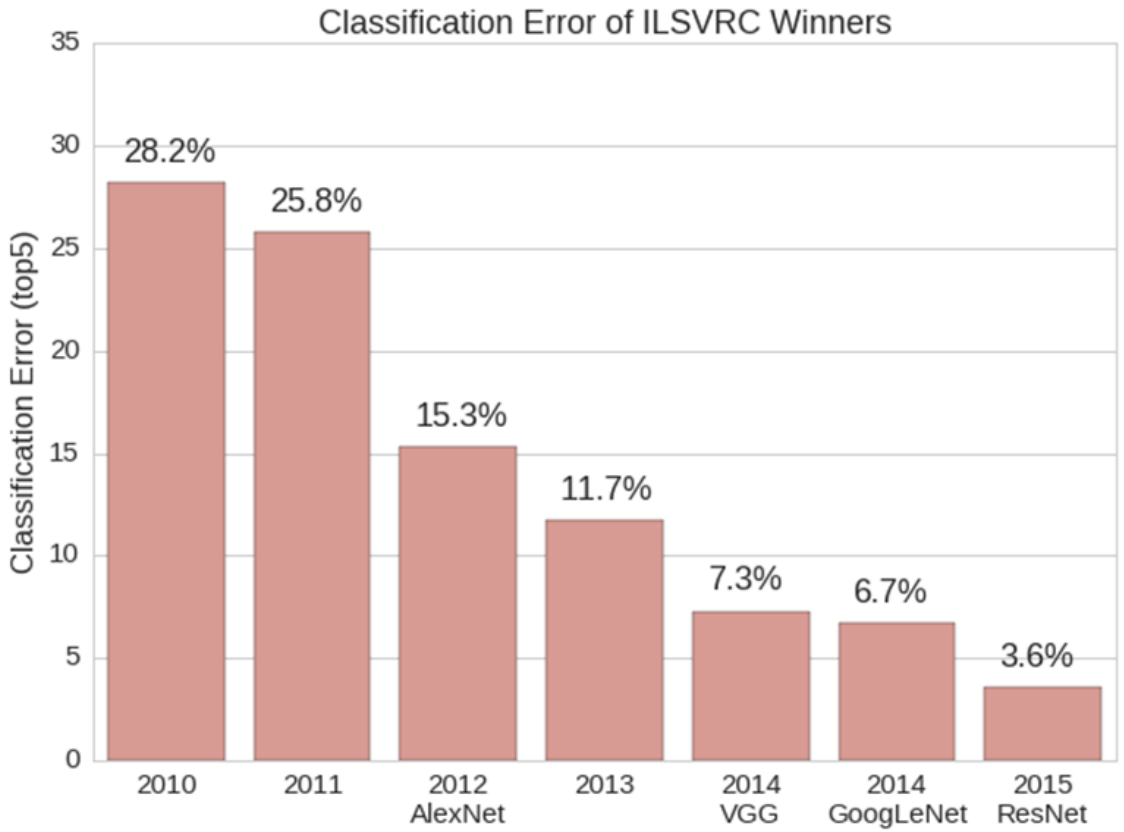
Featureraum wird kleiner

Faltendende (Convolutional) Neuronale Netze



[http://ufldl.stanford.edu/wiki/index.php/Feature_extraction_using_convolution]

Resultate im Bereich Bildklassifizierung



Grafik: William Beluch, ImageNet Classification with Deep Convolutional Neural Networks

Noch vor 10 Jahren: unmöglich

Rasante Entwicklung in den letzten Jahren

Oft verwendet: Ensembles von Netzen

Netze werden tiefer: ResNet (2015) mehr als 150 Schichten

Anwendung: IQ-Test

Lösen von verbalen Verständnisfragen in IQ-Tests [Wang et al. 2015]

Verbale IQ-Tests beinhalten hier 5 Arten von Fragen:
Analogie 1, Analogie 2, Klassifikation, Synonym, Antonym

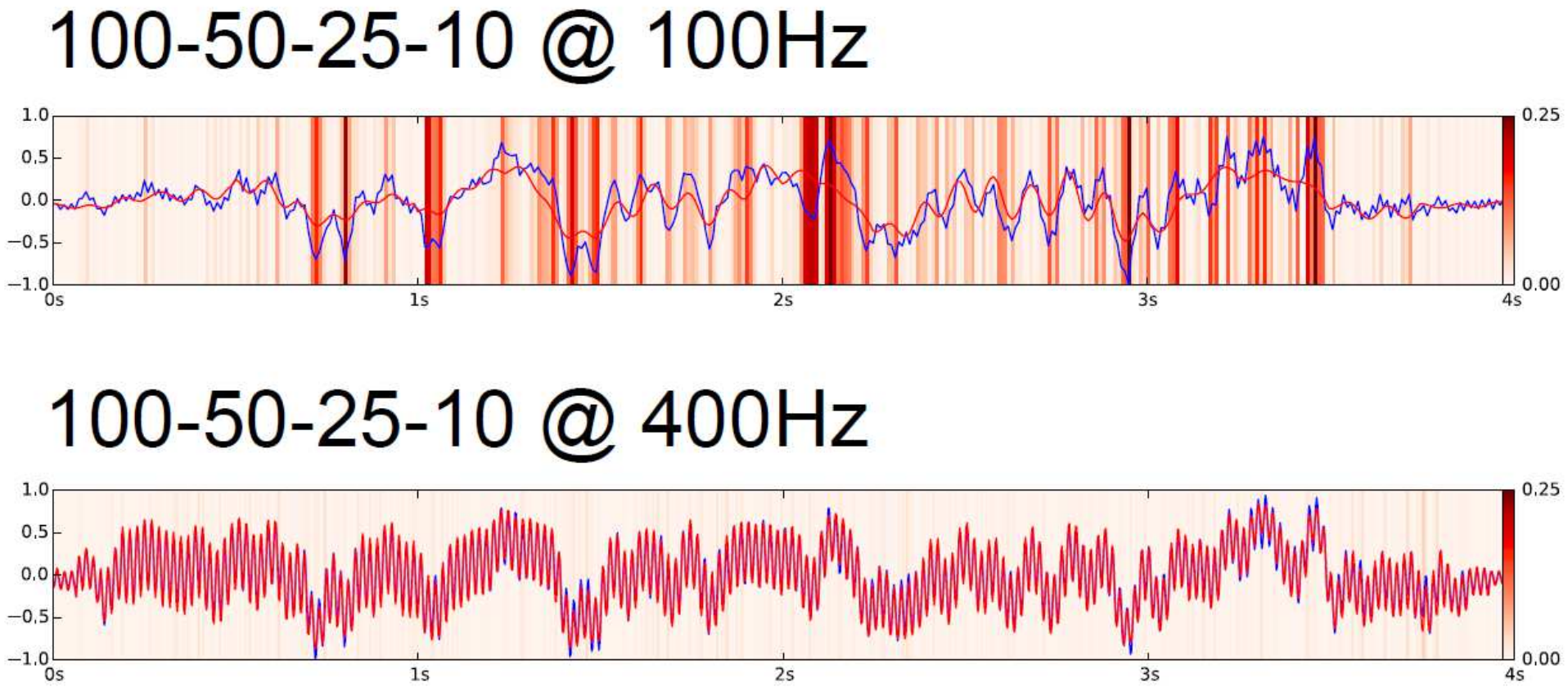
Beispielfrage(Analogie 1): Isotherm verhält sich zu Temperatur wie isobar zu?
(i) Atmosphäre, (ii) Wind, (iii) Druck, (iv) Ausdehnung, (v) Strömung

Ansatz:

- Klassifiziere den Fragentyp mit Hilfe einer SVM
- Benutze für jeden Fragentyp einen dafür erstelltes Tiefes Neuronales Netz
- Nutze zum Lernen von zusammengehörenden Wörter eine große Datenbasis (Wiki2014)

Ergebnis: DeepLearning schneidet etwas besser ab, als Bachelor-Absolventen

Rhythmus-Rekonstruktion durch EEG-Analyse



[Quelle: Sebastian Stober, DKE-Kolloquium 03.06.2014]

German Traffic Sign Recognition Benchmark (GTSRB)

Wurde analysiert bei der International Joint Conference on Neural Networks (IJCNN) 2011

Problemstellung:

Ein Bild, mehrere Klassen Klassifikationsproblem

Mehr als 40 Klassen

Mehr als 50.000 Bilder

Ergebnis:

Erste übermenschliche visuelle Mustererkennung

Fehlerraten:
Mensch: 1.16%, NN:0.56%

Stallkamp et al. 2012

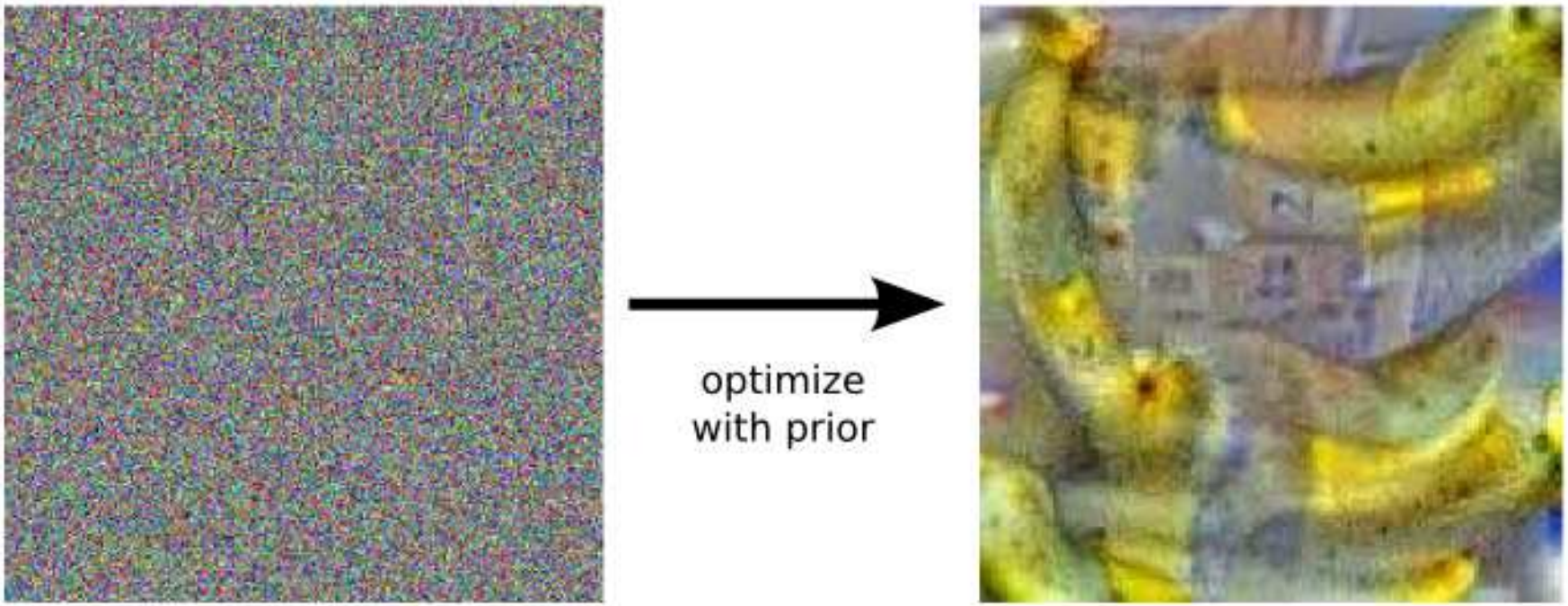
Verwendetes Netz:
Input, Conv., Max., Conv., Max., Conv., Max, Full, Full

Details zu den Gewinnern

Visualisierung von gelernten Neuronalen Netzen

Neuronale Netze zur Objekterkennung in Bildern

Was erkennt ein Neuronales Netz in Rauschen, wenn es Bananen gelernt hat?



Mehr Beispiele

Quelle: Heise: Wovon träumen neuronale Netze?

AlphaGo: Problemstellung Go

2 Spieler (Schwarz, Weiß)

Legen abwechselnd Steine

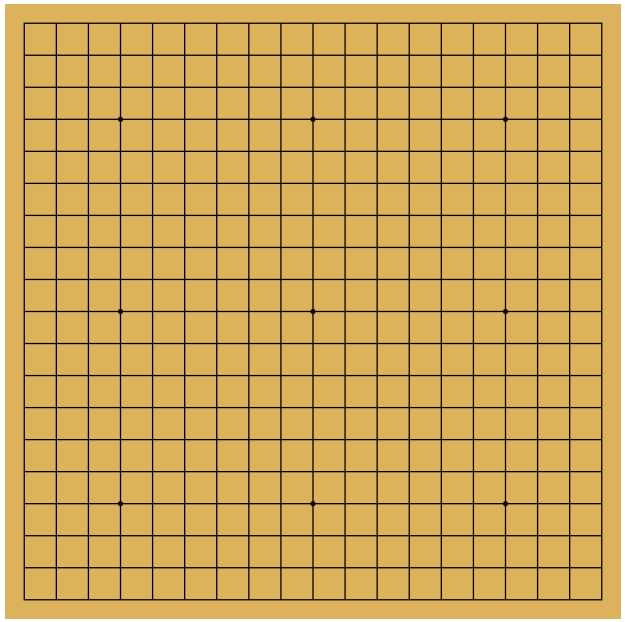
auf einem 19 x 19 Gitter

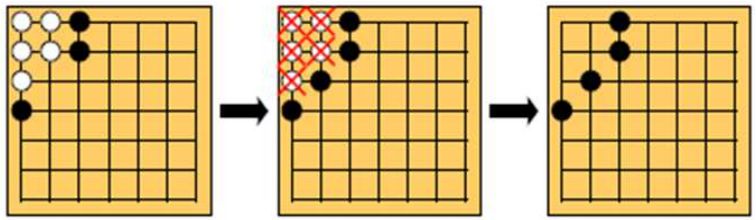
Ziel: Die Größte Fläche einkreisen

eingekreiste Steine werden weggenommen

Anzahl der Möglichkeiten: 250^{150}

Vergleich zu Schach: 35^{80}



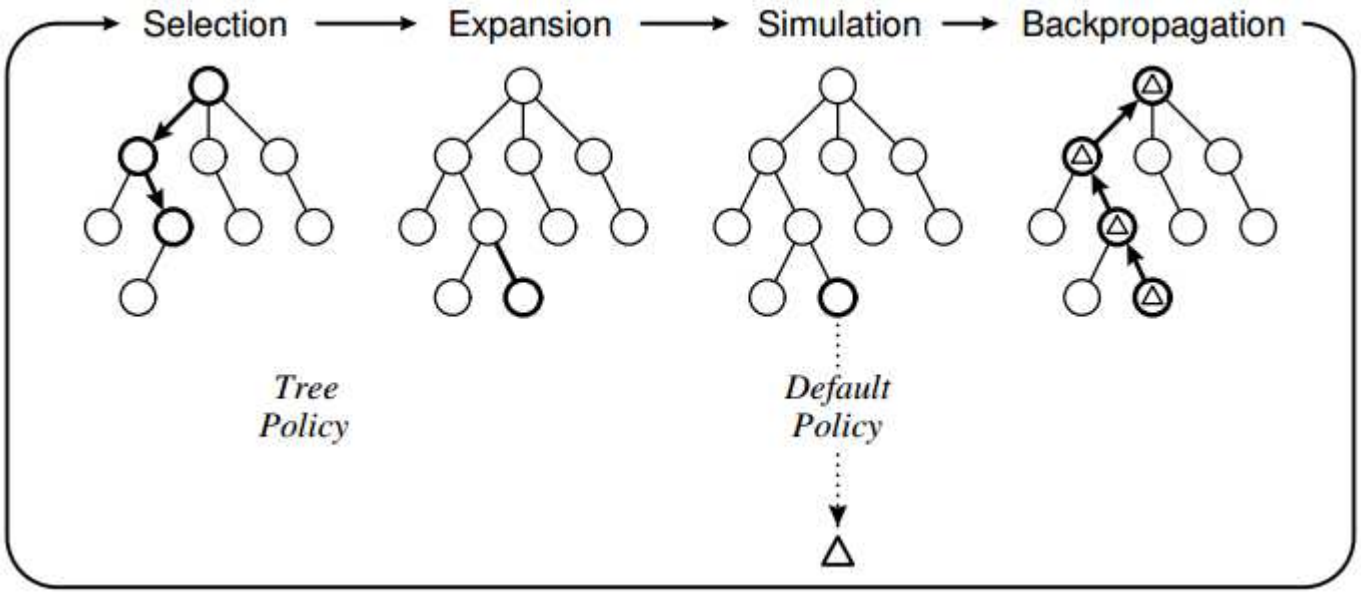


AlphaGo: Ansatz Monte Carlo Suche

Ansatz: Suche im Spielbaum

Lerne Netz 1 für menschenähnliche nächste Züge

Lerne Netz 2 zum Bewerten von Stellungen



AlphaGo: Ergebnisse

Sieg gegen Europameister, Fan Hui: 5 zu 0

Sieg gegen Top10 der Weltrangliste, Lee Sedol: 4 zu 1

AlphaGo	Search threads	CPUs	GPUs	Elo
Asynchronous	1	48	8	2203
Asynchronous	2	48	8	2393
Asynchronous	4	48	8	2564
Asynchronous	8	48	8	2665
Asynchronous	16	48	8	2778
Asynchronous	32	48	8	2867
Asynchronous	40	48	8	2890
Asynchronous	40	48	1	2181
Asynchronous	40	48	2	2738
Asynchronous	40	48	4	2850
Distributed	12	428	64	2937
Distributed	24	764	112	3079
Distributed	40	1202	176	3140
Distributed	64	1920	280	3168

Deep Learning Libraries

Theano
http://deeplearning.net/software/theano/
Python Implementierung für GPU-Verarbeitung von mathematischen Ausdrücken

Tensorflow https://www.tensorflow.org/
Verwendet von Googles DeepMind

Keras
http://keras.io
Python Implementierung, basierend auf Theano oder Tensorflow

Torch
http://torch.ch/
LuaJIT und C/CUDA Implementierung, verwendet bei Facebook, Google, Twitter

DL4J
http://deeplearning4j.org/
Plattformunabhängige Java Implementierung, kompatibel mit Spark, Hadoop

Caffe
http://caffe.berkeleyvision.org/
C++, CUDA Implementierung mit Python und MATLAB Schnittstelle
Sehr schnell, viel verwendet für Bildanalyse z.B. bei Facebook