Mehrschichtige Netze

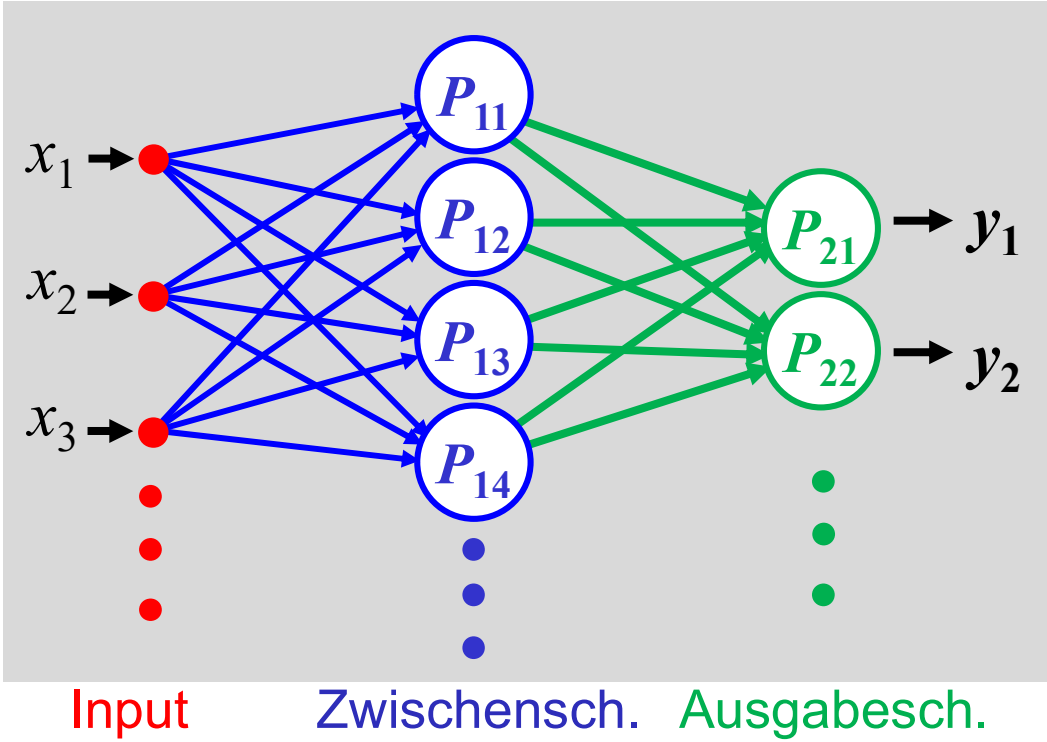
TECHNISCHE HOCHSCHULE MITTELHESSEN

Weitere Lösung des XOR Problems: Mehrere Schichten Neuronen

⇒ **Neuronales Netz!**

Genauer: **Feed forward Neural Network**

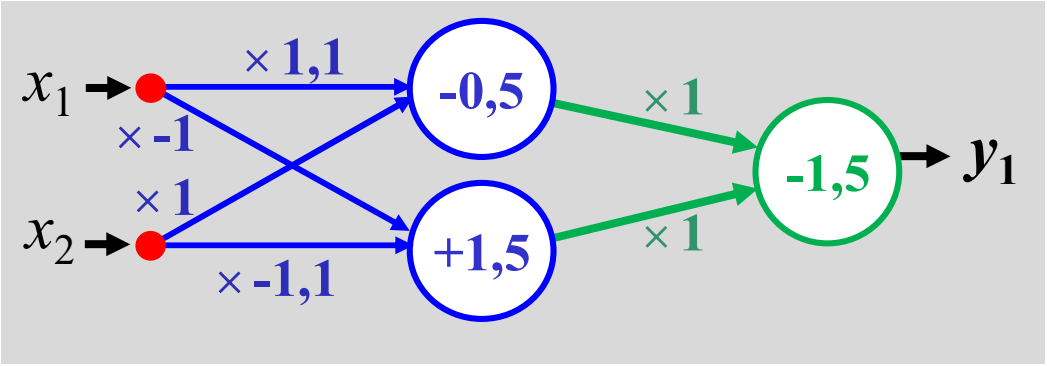
Neuronen nur mit Neuronen der vorhergehenden Schicht verbunden.
In „altmodischen“ Netzen: mit jedem Neutron dort

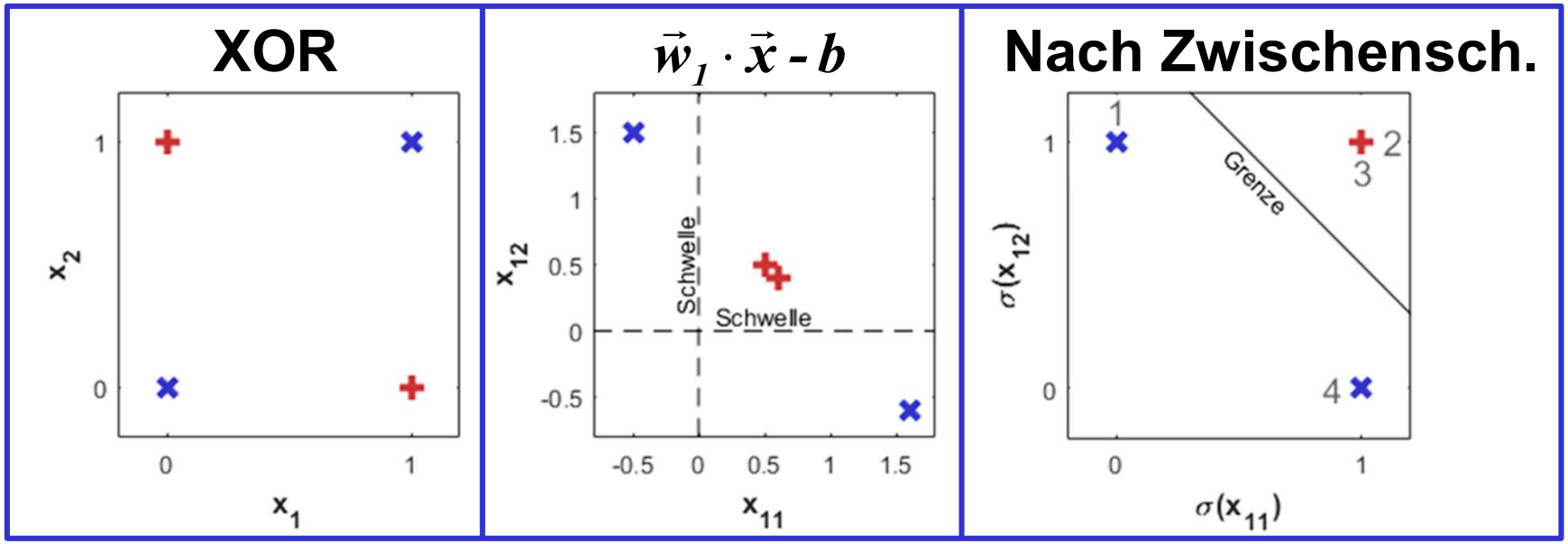


Mehrschichtige Netze: XOR

THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Eine von vielen möglichen Lösungen:





Lernen bei vielen Schichten

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

Probleme :

- Lernen: Bestimmen Parameter $\vec{\theta}$ (Gewichte $\vec{w}_k$, Schwellen s_k)
  Nur Ausgangsschicht erlaubt Soll-Ist Vergleich,
  Kriterium für Zwischenschichten?
- Hyperparameter: Grösse / Anzahl Schichten, ...
- Allgemeine Formulierung für möglichst viele Netzwerktypen

Optimales Netz liefert: $P(\omega_i)\, p(\omega_i \mid \vec{x})$

$\Rightarrow$ Statistische fundierte optimale Hergehensweise!

Bausteine neuronales Netz für Klassifikation

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

Lerndaten, Menge $\mathbb{P}$ ettiketierter Stichproben $\left(\vec{x}_i, \vec{Y}_i\right) \in \mathbb{P}$

Modell $\vec{y} = \vec{n}\left(\vec{x}, \theta\right)$ für Wahrscheinlichkeiten, hier neuronales Netz mit Parametern $\vec{\theta}$ (Schwellen s_k, Gewichte $\vec{w}_k$)

Zielfunktion (Loss-Function): $\ell\left(\mathbb{P}, \vec{\theta}\right)$ wird $\approx$ minimal für optimales Modell

Optimierungsalgorithmus: Optimiert Modell-Parameter $\vec{\theta}$, so dass $\ell\left(\mathbb{P}, \vec{\theta}\right)$ klein (Minimum??)

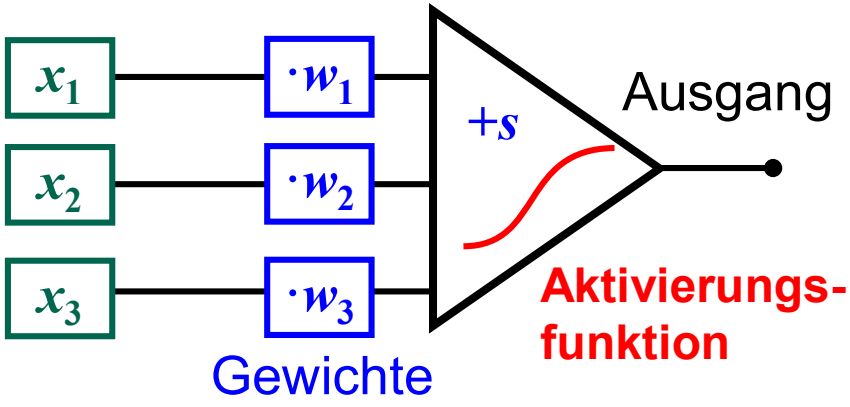
Aktivierungsfunktionen

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

Gutes Netzwerk ≈ Bedingte Wahrscheinlichk. x Apriori:

Resultat sollte kontinuierlich von Merkmalsvektoren und Parametern abhängen. Schwelle ⇔ Stufenfunktion: unstetig!

Auch: Optimierungsalgo. benötigt Ableitung ≠ 0 („Gefälle“)



$$y_k(\vec{x}) = \sigma(\vec{w}_k \cdot \vec{x} + s_k)$$

$\vec{x}$: Eingangssignale

$\vec{w}_k$: Gewichte Neuron k

s_k: negative Schwelle Neuron k

Konvention : Schwelle Vorzeichenwechsel!

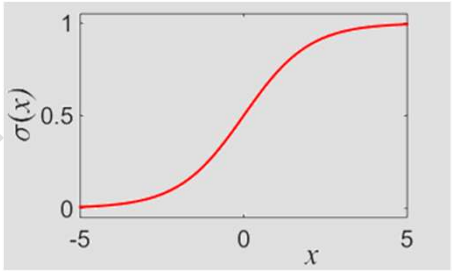
$\sigma(x)$: Aktivierungsfunktion, z.B.: "kontinuierliche Stufe"

Aktivierungsfunktionen

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

- **Sigmoid:** Schaltet Neuron an/aus

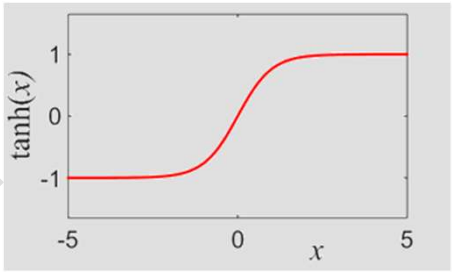
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- **Tangens Hyperbolicus:**

  Neuron hemmt/verstärkt

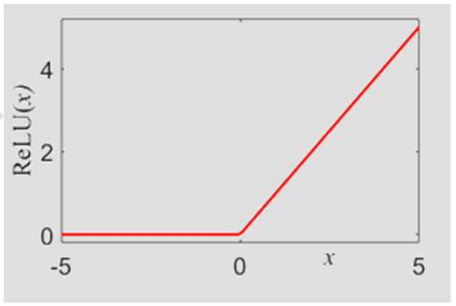
$$\theta(x) = \tanh(x)$$



- **ReLU:** Rectified-Linear-Units:

$$\mathrm{ReLU}(x) = \max(0, x)$$

  Wichtig für tiefe Netzwerke,
  keine Sättigung für grosse x!

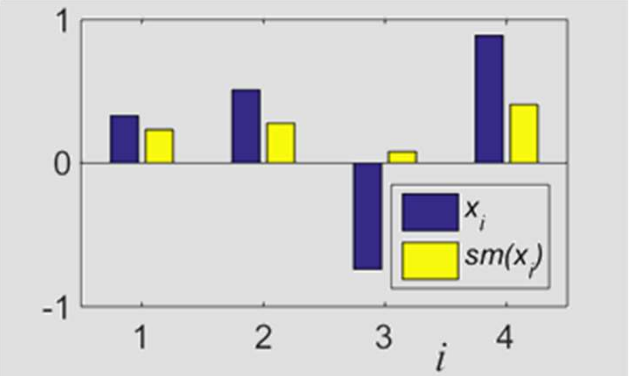


Aktivierungsfunktionen

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

- **Softmax**: normierte E-Fkt Outputs für Outputschicht Mustererkennung,
  ≈ bedingte Wahrscheinlichkeit

$$S(x) = \frac{e^{x_n}}{\left\| e^{\vec{x}} \right\|}$$



Output:
- Summe über alle Neuronen = 1
- $S(x) \in [0,1]$
- E-Funktion: damit oft ein Output dominiert

Reproduziert Klassifikator!

Rechnen: ein Layer

TECHNISCHE HOCHSCHULE MITTELHESSEN

Einzelnes Neuron: $y_k\left(\vec{x}_i\right)=\sigma\left(\vec{w}_k\cdot\vec{x}_i+s_k\right)$

Alle Neuronen einer Schicht : $\vec{y}\left(\vec{x}_i\right)=\sigma\left(\hat{W}\cdot\vec{x}_i+\vec{s}\right)$

Viele Stichproben → Matrixm. : $\hat{Y}\left(\hat{X}\right)=\sigma\left(\hat{W}\cdot\hat{X}\oplus\vec{s}\right)$

Effizient auf Grafikkarte

$\vec{x}_i, \hat{X}=\left(\vec{x}_1 \quad \vec{x}_2 \quad \cdots\right)$: Merkmalsvektor Stichprobe i, viele Stichproben

$\vec{w}_k,\ \hat{W}=\begin{pmatrix}\vec{w}_1^T\\ \vec{w}_2^T\\ \vdots\end{pmatrix}$ Gewichte Neuron k, aller Neuronen einer Schicht

$s_k, \vec{s}$: neg. Schwelle Neuron k, aller Neuronen einer Schicht

$\sigma(x), \sigma(\vec{x})$: Aktivierungsfunktion, elementweise

$y_k, \vec{y}, \hat{Y}=\left(\vec{y}\left(\vec{x}_1\right), \vec{y}\left(\vec{x}_1\right), \cdots\right)$: Ausg. Neuron k, eine Schicht, viele Merkmalsvektoren

$\oplus$, oft einfach "+": Gleicher Schwellvektor für jede Spalte von $\hat{W}\cdot\hat{X}$

Ziel (Loss-) Funktion

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

Loss-Function: Maß (Skalar!) für Abweichung Soll - Ist

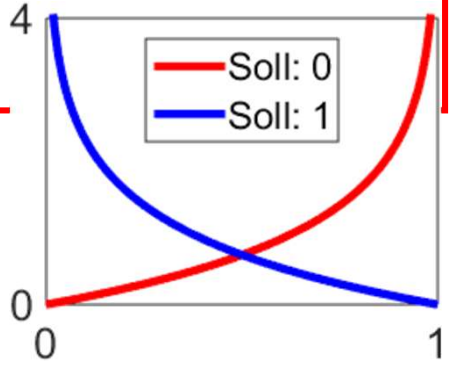
Gegeben ettiketierte Stichproben $\mathbb{P}: (\vec{x}_i, k_i)$ und Modell $\vec{Y}_i = \vec{n}(\vec{x}_i, \vec{\theta})$

- Einfach: **Abstand**-Quadrat Soll zu Ist-Vektor: $\ell(\vec{\theta}) = \sum_{i=1}^{N} \left\| \vec{Y}_i - \vec{y}_i \right\|_2$

- **Kreuzentropie**: $\ell(\vec{\theta}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{m=1}^{M} \left[y_{im} \cdot \ln(Y_{im}) + (1 - y_{im}) \cdot \ln(1 - Y_{im}) \right]$

$\vec{x}_i$	Merkmalsvektoren	$\vec{y}_i$	Soll-Output $\left(= \delta_{mk_i},\ k_i \text{ ist Klasse von } \vec{x}_i\right)$
N	Etikettierte Stichproben	M	Klassen/Output Neuronen
$\vec{\theta}$	Parametervektor	$\left\|\vec{v}\right\|_2$	Länge Vektor $\vec{v}$

Kreuzentropie:



Regularisierung Teil 1

Netzwerk hat viele Parameter: z.B.: 100 Inputs. 2 Schichten á 100 Neuronen, 10 Klassen: 21 000 Parameter

$\Rightarrow$ **Overfitting garantiert!**

$\Rightarrow$ **Weight-Decay** Gewichte klein halten:

$\left\|\vec{\theta}\right\|_2^2 = \sum \theta_i^2$ (L2-Norm, normaler Abstand^2)

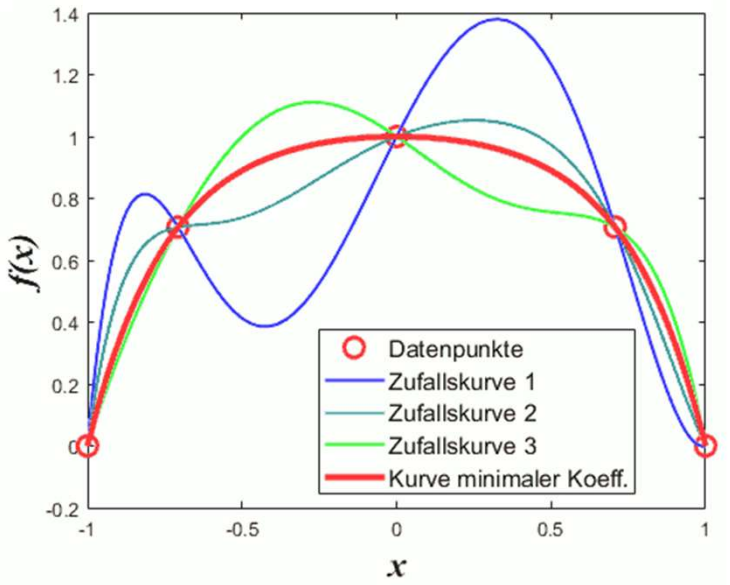
auch: $\left\|\vec{\theta}\right\|_1 = \sum \left|\theta_i\right|$

Loss-Function + Weight-Decay:

$\ell_w\left(\vec{\theta}\right) = \ell\left(\vec{\theta}\right) + \lambda\vec{\theta}\cdot\vec{\theta}$ L2-Norm

$0 < \lambda \ll 1$: Gewicht für Weigt-Decay

(λ lässt Gewichte kleiner werden)



Kleine Gewichte generalisieren besser!

Lernalgorithmen

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

Mathe: Finde Minimum nichtlinearer Funktion vieler Variablen

Probleme:
- Kein Algorithmus mit garantiert absolutes Minimum
- Iterationen, Anfangswertabhängiges Resultat

Algo Gradientenabstieg:

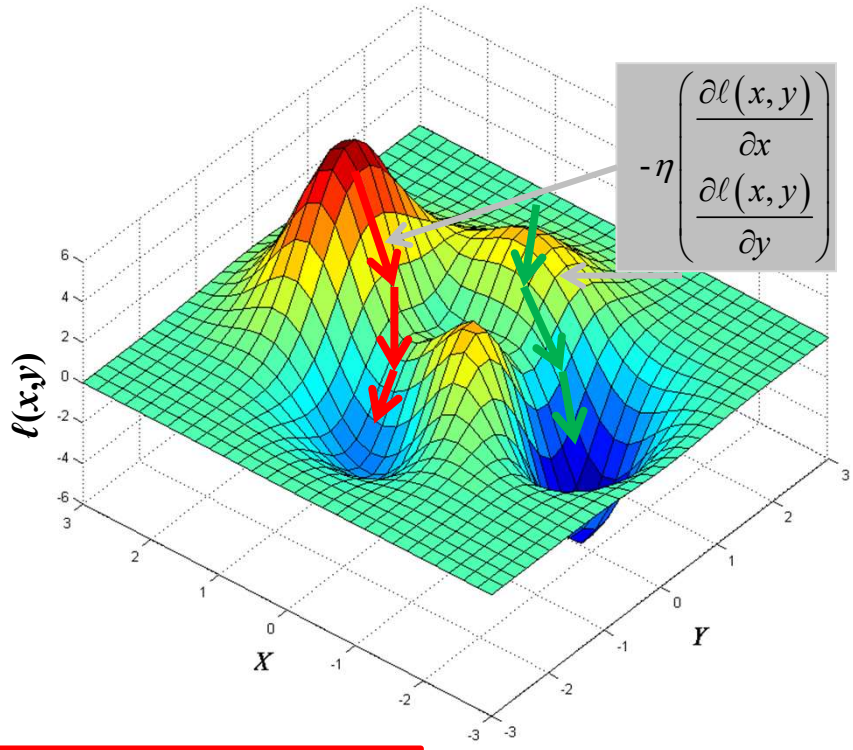
- Wähle Anfangswerte für $\vec{\theta}$
- Schleifen, mehrfach über alle k:

$$\theta_k' = \theta_k - \eta \frac{\partial \ell}{\partial \theta_k} \Rightarrow \ell\left(\vec{\theta}'\right) < \ell\left(\vec{\theta}\right)$$

"bergab": Ableitung $-\frac{\partial \ell}{\partial \theta_k}$ (neg. Gradient)

$0 < \eta \ll 1$: Gehe nur kleines Stück

- Abbruch: $\ell\left(\vec{\theta}\right)$ wird nicht kleiner??



Anfangswerte zufällig: Resultat nicht wiederholbar!
Drastisch für Netzwerk, schwach für Fehlerrate

Lernalgorithmen

Anfangswerte (Pseudo-) Zufallszahlen:

- Je nach Startpunkt des Gradientenabstiegs anderes lokales Minimum!
- Gelernte Netze nicht wiederholbar, drastisch unterschiedliche Parameter $\vec{\theta}$
- Aber: Weniger starke Abweichungen individueller Fehler
- … und (oft noch weniger) Fehlerrate für Testdaten

Meta-Algorithmen: Kombiniere mehrere Netzwerke

Z.B. einfachste Möglichkeit: Summiere Outputs mehrerer NN.

Stochastic Gradient Descent (SGD)

THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Problem: in Lossfunktion $\ell\left(\vec{\theta},\mathbb{P}\right)$ gehen alle Stichproben ein!

$\Rightarrow$ Gradient $\vec{g}\left(\vec{\theta},\mathbb{P}\right)$ davon sehr aufwendig zu berechnen!

Sehr viel effizienter:

- Unterteile Stichproben $\mathbb{P}$ in Teilmengen $\mathbb{P}_i$ (Mini-Batches)
- Gradient $\vec{g}\left(\vec{\theta},\mathbb{P}_i\right)$: ungenau aber schnell

$\Rightarrow$ Bei gleichem Aufwand kommen viele ungenaue Schritte dem Minimum näher als ein genauer Schritt.

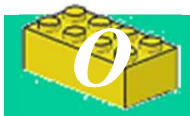
SGD mit Momentum: Gradiententiefpass $g_k \leftarrow \beta g_{k-1} + \left(1-\beta\right) g_k$

$\beta \in \left(0,1\right)$, typisch $\beta \approx 0,9 \rightarrow$ Gradient reagiert träge auf Änderung

ADAM: verbesserter SGD, Richtung zum Minimum optimiert

Epoche: Lernschritte bis alle Stichproben je 1-mal verarbeitet

Optimale Mini-Batch Größe problem- und hardwareabhängig!

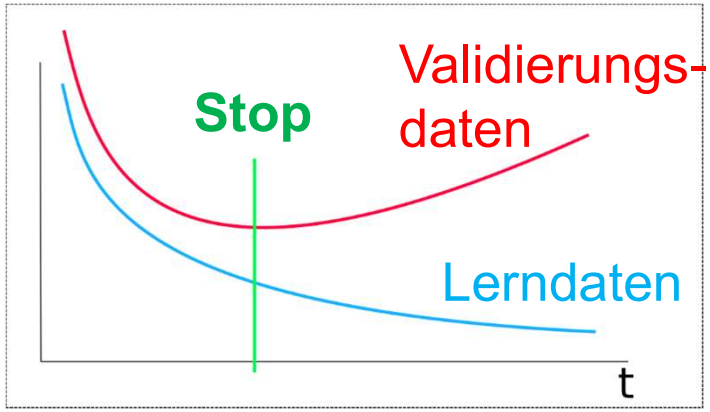


Regularisierung Teil 2

Iterationsalgorithmus: Nach wenigen Schritten grobes Modell,
Je mehr Epochen umso detaillierter Modell!

Early Stop: Stop Optimierungsalgorithmus vor Erreichen des Minimums für Lerndaten!

Kriterium: Periodisch Netzwerk mit Nicht-Lerndaten überprüfen, stoppe Optimierung falls Fehlerrate ansteigt / konstant



Oft: 3. Teilmenge der ettikettierten Sichproben: **Validierungsdaten**

Aktivierungsf.: Probleme

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

Beweisbar: Neuronales Netz mit einem versteckten Layer kann jede "sinnvolle" Funktion $\vec{y} = f(\vec{x})$ beliebig genau annähern, vorausgesetzt man hat ausreichend viele versteckten Neuronen.

Alternativen: wenige breite oder viele schmale Schichten?

Besser: viele Schichten (Deep Learning)!

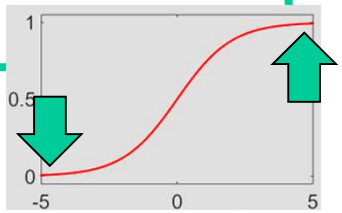
Mehr Layer:
- benötigen insgesamt weniger Neuronen
- verallgemeinern besser

Sättigung: Damit Gradientenoptimierung Lösung findet, muss $\|Gradient\| > 0$! Zielfunktion / Aktivierungfkt.

Zielfunktion: Kreuzentropie

Aktivierungfkt.: Softmax

Keine Sättigung:
- **ReLU**
- kleine Anfangswerte



Input Normalisierung

Falls unterschiedliche Grössenordnungen der Merkmale (Inputs ins NN) dann:

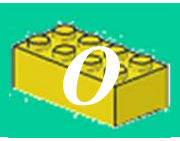
Merkmal i Stichprobe j: Mittelwert $\overline{x}_i = \frac{1}{N}\sum_{j=0}^{N-1} x_{ij}$ Varianz $\sigma_i^2 = \frac{1}{N-1}\sum_{j=0}^{N-1}\left(x_{ij} - \overline{x}_i\right)^2$

$$x'_{ij} = \frac{x_{ij} - \overline{x}_i}{\sigma_i}$$

Gleiche Normalisierung Lernen und Klassifizieren!

Oft bei Bildern einfache feste Werte: z. B. $\overline{x}_i = 128,\ \sigma_i = 1$

Lernrate beschleunigt!

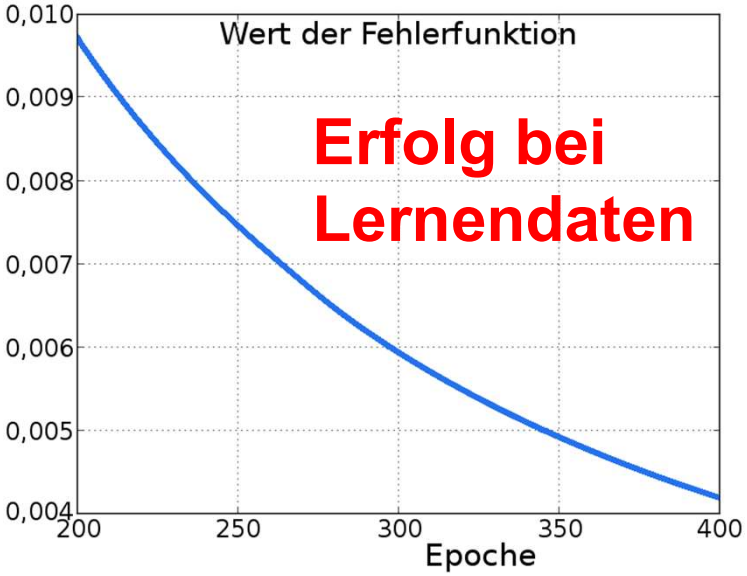


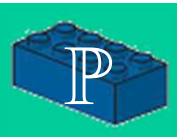
Testen / Overfitting

Testen - Teile etikettierte Stichproben in 3 Untermengen:

- Trainingsdaten: Zum Lernen
- Validierungsdaten: Zum Testen verschiedener NN-Parameter
- *Testdaten*: Zum Abschätzen der Leistung des finalen NN.

Großes Problem: Overfitting Netz lernt Trainingsdaten zu detailliert, verbessert sich mit Testdaten nicht!





Augmentierung

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

In der Regel: zu wenig etikettierte Stichproben

Um so eher ein Problem, je mehr Parameter das Netzwerk hat!

Abmildern durch synthetischen Input: geringfügig veränderte bekannte etikettierte Stichproben.

- Bilder: geringe Verschiebung, - Vergrößerung, - Drehung, Spiegelung horizontal/vertikal, Crop, …
  Allgemein: Rauschen, …
- Nur sinnvolle Augmentierung. (z.B. steht reales Objekt Kopf?)
- Große Datensätze (Speicher!): Augmentierung on-the-fly, d.h. für jeden Batch neuen augmentierten Input berechnen.
  Keras: ImageDataGenerator (dg) und model.fit_generator(dg.fit(…), …)

Wahl Hyperparameter

Hyperparameter:
Parameter des Netzwerkes, die nicht gelernt werden.

- Schichten
- Größe Lernschritte
- Mini-Batch Größe
- …
- Zahl Neuronen pro Schicht
- Art Aktivierungsfunktion
- Merkmalsvektoren: verarbeitet - unverarbeitet, wie viele, …

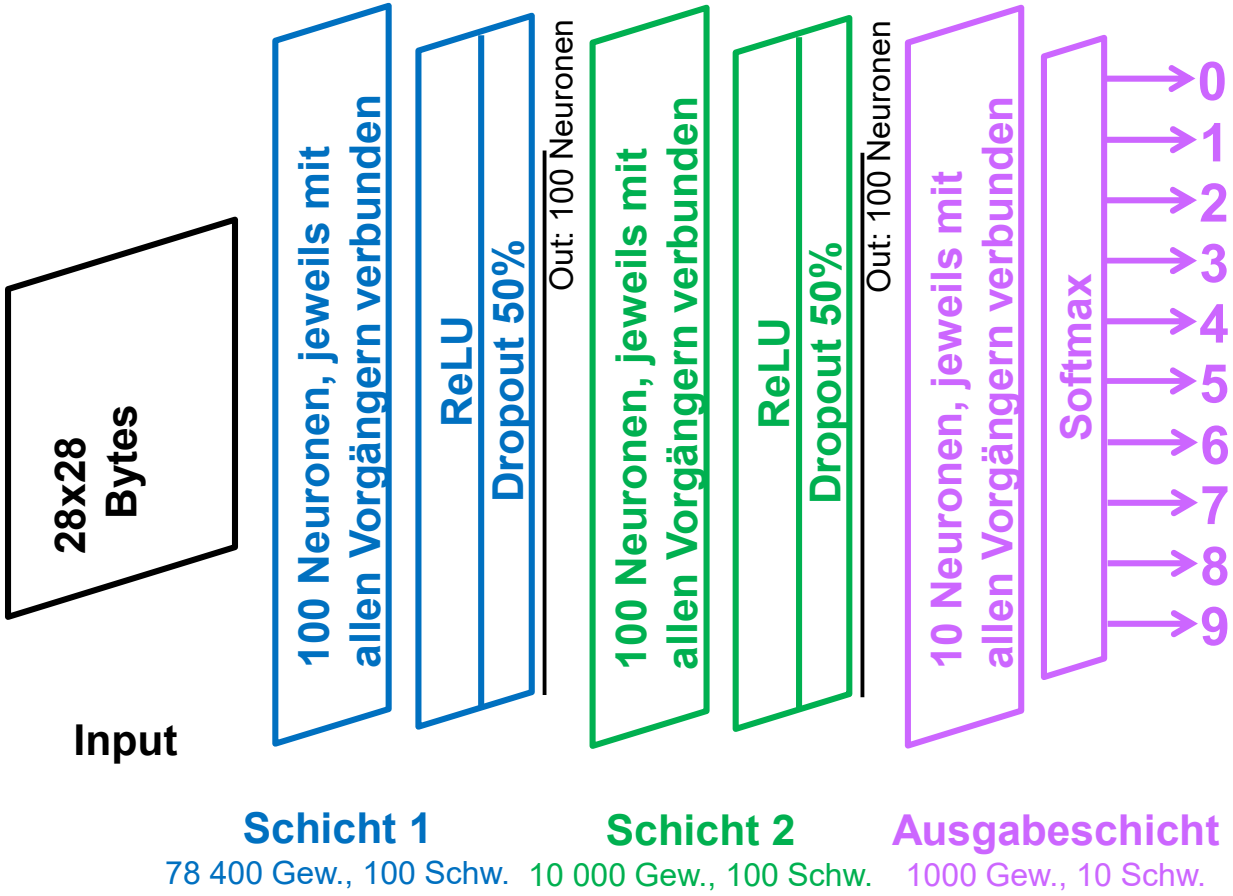
Wichtig / Schwierig: Zunächst einmal Konfiguration finden, die überhaupt lernt (bessere als zufällige Resultate) ⇒
Starte mit einfachem Netz, reduziere Rechenzeit: viele Tests

Optimierung Hyperparameter damit weniger anspruchsvoll

Beispiel 3 Schicht Netzwerk

TECHNISCHE HOCHSCHULE MITTELHESSEN

MNIST-Datensatz: handschriftliche Ziffern, 28 x 28 Grauwertbilder
Lerndaten: 60 000 Stichproben Testdaten: 10 000 Stichproben



Mit Testdaten: 2,4% Fehler $\Leftrightarrow$ 240 falsch klassifizierte Ziffern
$\approx$ 90 000 Parameter

Deep Learning

TECHNISCHE HOCHSCHULE MITTELHESSEN

<u>Fortschritt letzte ≈ 10 Jahre</u>:

- Sehr tiefe Netzwerke möglich (lernbar)
- Neue Schichttypen: Weniger Parameter pro Schicht
- Bessere Berücksichtigung der Merkmalsstruktur
  z.B. 2D Merkmale
  Nahe und entfernte Pixel haben unterschiedliche Bedeutung
- Interpretation: Netzwerk lernt optimale Deskriptoren
  - Rohdaten als Input
  - Schichten bilden optimale Merkmalen
  - Gute Merkmale auch für andere Probleme gut!
    Nimm aufwendig vortrainiertes Netz, tausche Ausgabeschicht aus, trainiere mit wenig Aufwand (im Wesentlichen Ausgabesch.)

Schöner Online Kurs in Deep Learning (You Tube) Stanford CS231n:
Li, Johnson, Yeung: Convolutional Neural Networks for Visual Recognition

Konvolution (Merkmalsbildung)

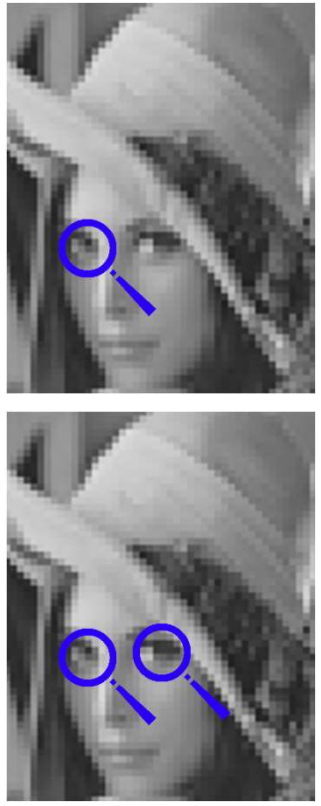
THM TECHNISCHE HOCHSCHULE MITTELHESSEN

Grundprinzip Konvolution:

- Wesentliche Information in Nachbarschaft
  2D Anordnung der Daten wird berücksichtigt
- Sieht verschobene Nachbarschaft gleich aus, dann gleiche Bedeutung
  Gleiche Gewichte/Schwellen für verschobene Neuronen, Schichten möglich mit wenig Gewichte/Schwellen

Pooling: fasse benachbarte Neuronen zusammen
- Maximum / Mittelwert 2x2, 3x3 Umgebung
- Nachfolgende Schicht ist entsprechend kleiner

Genaue Position Objektbaustein nicht so wichtig,
wichtiger: Grobe Anordnung, Objekt existiert!

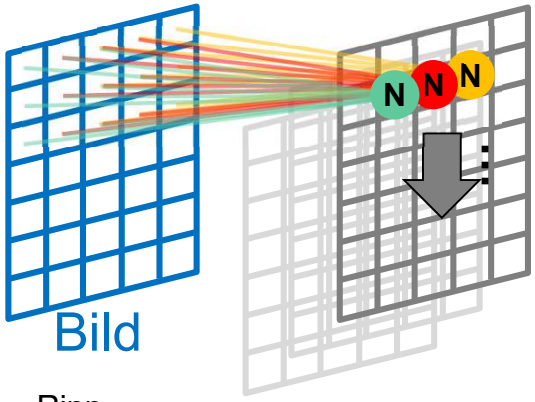


Konvolutionsschicht (CNN)

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

- Erhält 2D-Bildstruktur (Sinnvoller Input: Matrix!)
- Reduziert drastisch Anzahl Parameter $\vec{\theta}$

- Alle Neuronen einer Schicht:
  - gleiche Gewichte/Schwellen
  - Verarbeiten kleinen Bildausschnitt (3x3, 5x5, … Gewichte)
  - … der von Neuron zu Neuron verschoben ist
- Viele parallele Schichten liefern viele Higher-Level Merkmale
  Input 2. CNN-Schicht: gleicher Ausschnitt aller Vorgänger
  Z.B. 64 vorhergehende Schichten, 3x3 Umgebung: 576 Gewichte!



CNN-Schichten

Aktuelle Strategie bevorzugt viele 3x3 Schichten (tiefes Netzwerk)

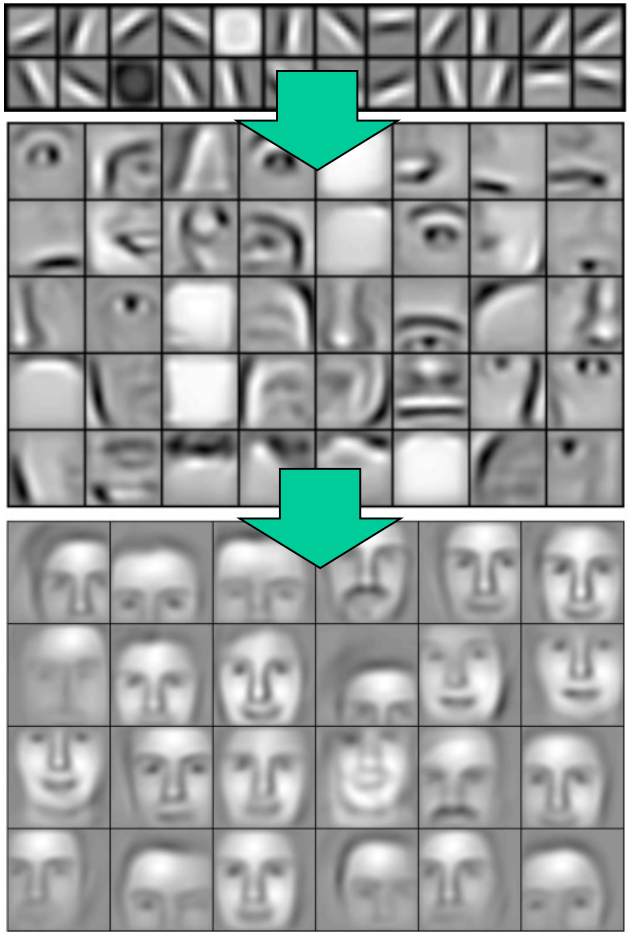
Konvolution plus Pooling

THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Biologisch inspiriert (Verarbeitung Sehnerven Säugetier)

Wirkungsweise:

- 1. Konvolutionsschicht „erkennt“ lokale einfache Merkmale
- Pooling: wenig verschobene Merkmale $\Rightarrow$ gleiche Bedeutung
- Höhere Konvolutionsschichten machen aus einfachen Merkmalen komplexe.



Regularisierung Teil 3

TECHNISCHE HOCHSCHULE MITTELHESSEN

Noch mehr Schichten: noch mehr Parameter, noch leichter Overfitting!!

- **Mehr Trainingsdaten** Auch: synthetische Daten
- **Weight Decay** `keras.regularizers.l2(`*`z.B.: 1e-4`*`)`
- **Dropout** (Neu! Zusätzlich benutzen):
  Beim jedem Lernschritt Anteil der Neuronen abschalten.
  (z.B. für 50% zufällig ausgewählt Neuronen: Output = 0)
  Anwenden des Netzes mit allen Neuronen, Gewichte skalieren mit Dropout Anteil ≅ Mittelwert vieler Teilnetze

- *Early Stop:* sollte nicht notwendig sein!

Oft sinnvoll: Viele Epochen lernen, starke Regularisierung (gegen Overfitting) ⇒ bessere Resultate

Pooling / Subsampling

TECHNISCHE HOCHSCHULE MITTELHESSEN

Ausgabeschicht: Neuronen = Anzahl Klassen!
⇒ also Reduktion der Neuronen pro Schicht!

Idee: Bildinhalt um 1 Pixel verschoben ⇒ gleiches Objekt

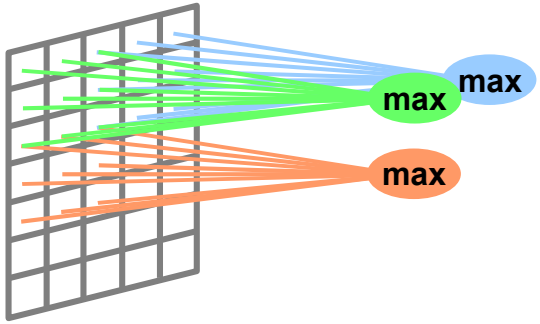
Pooling: Schicht berechnet aus Nachbarschaft einen Wert.

Hyperparameter:

- Größe der Nachbarschaft
- Strides: Verschiebung für nächstes Neuron
- Art der Berechnung:
  - Maximum
  - Mittelwert

} keine Parameter!

Subsampling: Konvolution in Schritten (stride) > 1



Z. B.: Max-Pooling mit 3 x 3 Nachbarschaft, Stride 2
7 x 5 Convolutional Layer ⇒ 3 x 2 Layer

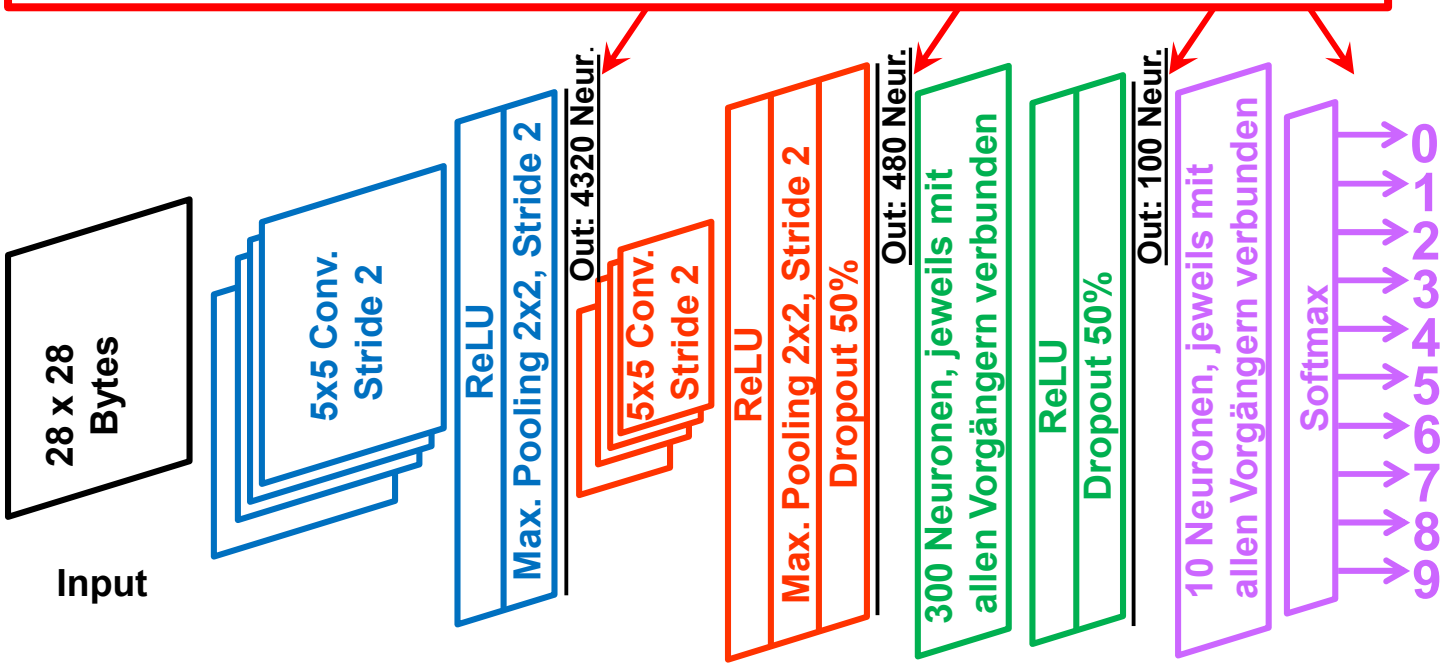
Beispiel Deep Learning

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

MNIST-Datensatz: handschriftliche Ziffern, 28 x 28 Grauwertbilder
Lerndaten: 50 000 Stichpr. Validierungsdaten: 10 000 Stichpr. Testdaten: 10 000 Stichpr.

- Sequenzen (Convolution x n, Auflösungsreduzierung) wiederholen
- Neuronen in Fully-Connected: zwischen Outputs vorher und Klassenanzahl

Trichterförmiges Design: Viele Inputs → wenige Klassen



30 Konvolutionssch.
750 Gewichte, 30 Schwellen

30 Konv.-Sch.
22 500 Gew., 30 Schw

Fully-Conn. Schicht
144 000 Gew., 300 Schw.

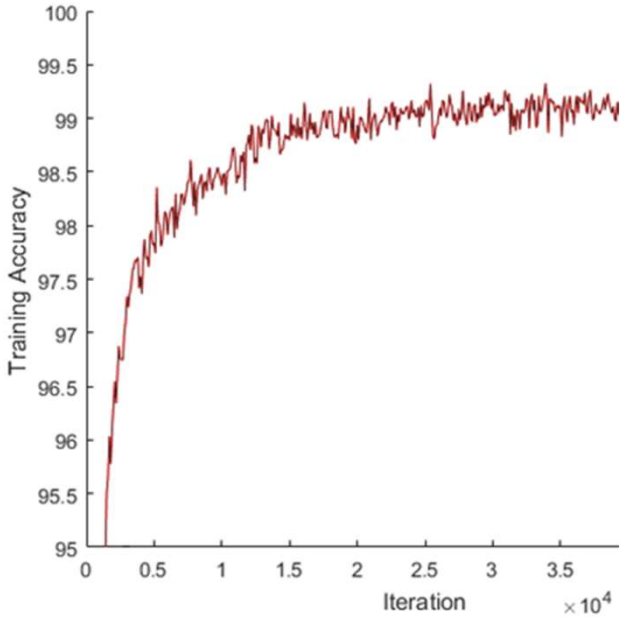
Ausgabeschicht
3000 Gew., 10 Schw.

Insgesamt:
170 250 Gewichte,
370 Schwellen!

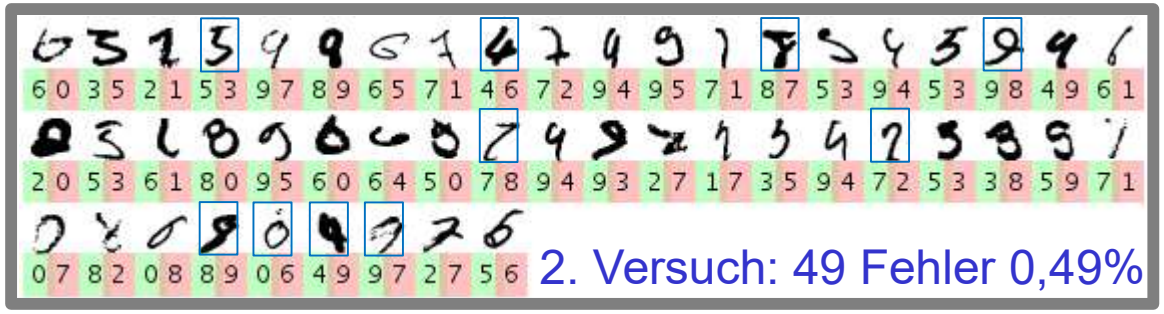
Beispiel Deep Learning

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

100 Epochen gelernt, ≈ 170 000 Parameter, Fehlerrate mit Testdaten



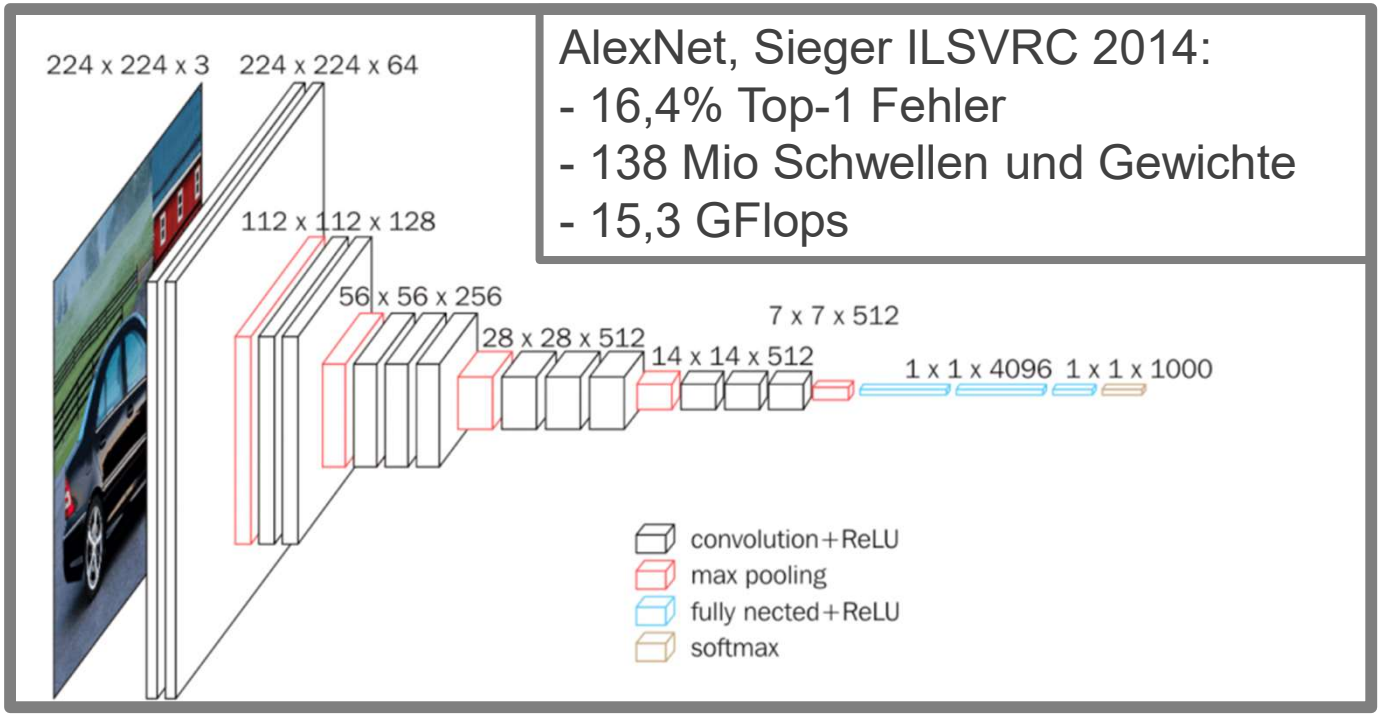
Sehr viele Lern (Optimierungs-) Schritte generalisiert noch gut!



Fehler im 2. Netzwerk, die im 1. nicht auftreten

Lernen hat deutliche Zufallskomponente
- Fehlerrate
- Unterschiedliche Klassifikation

Tiefe Netzwerke sind besser!



ILSVCR:
ImageNet Large Scale Visual Recognition Challenge

Publiziert:
- 200 Klassen
- Je 1000 Bilder

Wertung:
unbekannte Bilder

Imagenet:
- 14 Mio Bilder
- aus Netz viele Formate
- Menschl. Labelling in 20k Klassen
  Crowdsourcing, MTurk

⇒ Wird oft vortrainiert benutzt
⇒ Neue Ideen gefragt für tiefere und einfachere Netze

Berechnung tiefer Netze

TECHNISCHE HOCHSCHULE MITTELHESSEN

Sehr tiefe Netze

- Sehr viel Rechenzeit fürs Lernen
- Unhandlich, selbst bei Vorhersage (Rechenzeit, Speicherplatz)
- Tiefer bedeutet nicht Overfitting wird problematischer!

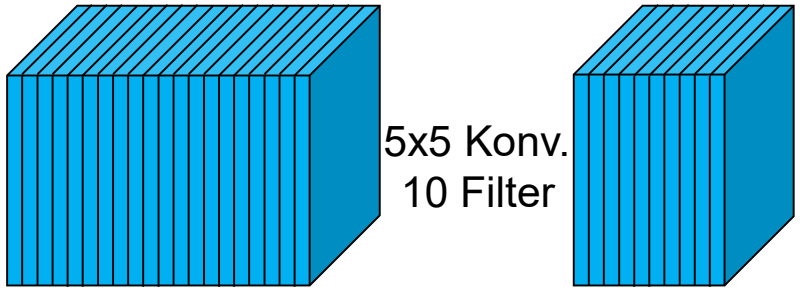
Sorry, Schleichwerbung für Nvidia Cuda: **GPGPU**
Unterstützt zur Zeit mit Abstand am besten massiv parallele Berechnungen (RTX 2080 FP16 ≈ 446·Resnet152/s),
im Vergleich zu CPU ohne weiteres ≫ 20-fach schneller!

Aktuelle performante Netze sind mit CPU alleine nicht mehr effektiv zu trainieren. (z.B. mehrere Tesla V100 á 14 Terraflops)
Training leicht 100 Epochen á 1 Mio Stichproben

Aufwand für Konvolution

Einfache und tiefe CNN:
Wie mehr Schichten und weniger Gewichte/Operationen?

Was „kostet“ Konvolution? Zum Beispiel:



	Gewichte	Ops
1 Neuron	5·5·20=500	5·5·20=500
1 Schicht	…=500	…·14·14=98 000
10 Sch.	…·10=5000	…·10=980 000

Allgemein: $Gewichte = k^2 \cdot L_0 \cdot L_1$

$Operationen = k^2 \cdot r \cdot c \cdot L_0 \cdot L_1$

k : Größe Konvolution
r, c : Bildzeilen/-spalten
L_0 : Schichten vorher
L_1 : Schichten nachher

Also Ideen:
- 3x3 oder 1x1 (?) Konv.
- Geringer Tiefe (Schichten)

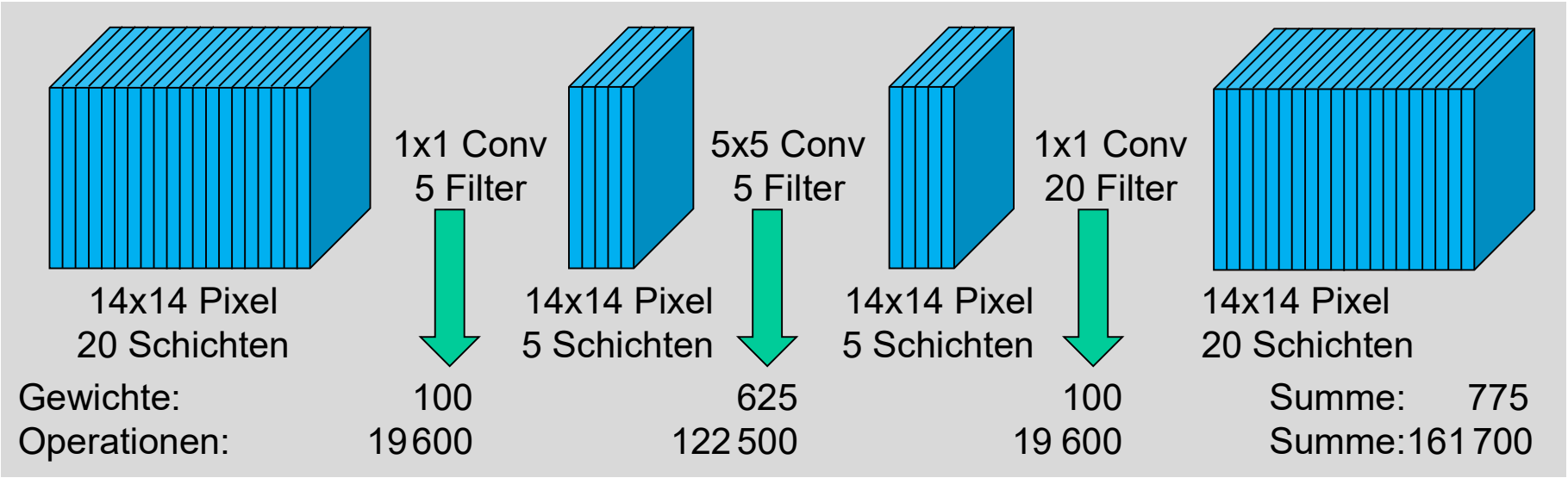
1x1 Konvolution

GoogLeNet/Inception v1 2014

TECHNISCHE HOCHSCHULE MITTELHESSEN

Einsicht: Konvolution ist mit weniger Gewichten darstellbar!

Idee: Lass Netzwerk lernen wie Features der Schicht vorher nützlich sind



Direkte 5x5 Konvolution mit 20 Filtern: 10 000 Gewichte und 1 960 000 Operationen!

Fazit: Starke Reduktion Gewichte und auch Operationen! *Hier auf 8%!!*

Besser: 1x1 Konvolution mit wenigen Filtern (Schichten),
… plus größere Konvolution
… plus 1x1 Konvolution mit mehr Filtern (Schichten)

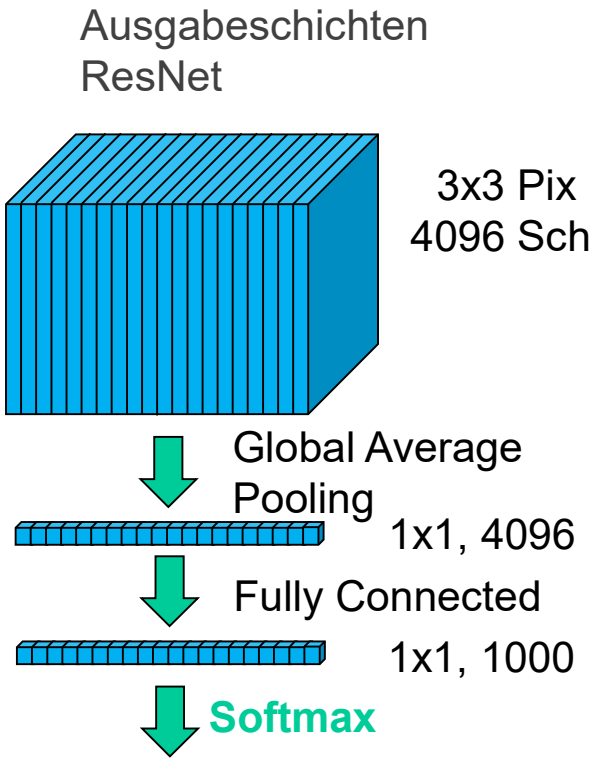
Fully Connected → Average Pooling

THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Problemzone: Fully-Connected Schichten!

Beispiel (VGG16): 3x3, 512 Konvolution →
4096 Fully Connected
$3 \cdot 3 \cdot 512 \cdot 4096 \approx 19$ Mio Flops

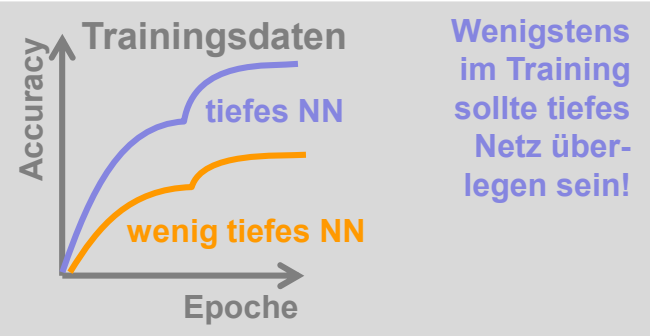
Lösung: Mittelwertbildung über Konvolutionsschichten → Parameterfrei!

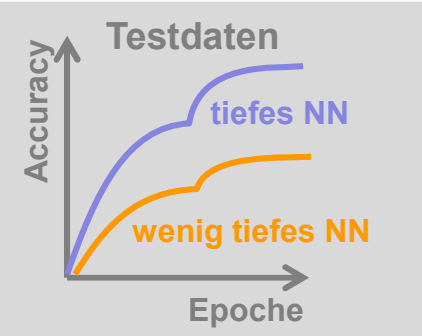


Residual Netz (ResNet)

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

Problem: Tiefere Netzwerke sollten besser sein (?) Aber: lernen schlecht (kein Problem: Overfitting!)





Diagnose: Probleme mit Optimierungsalgorithmus, denn tiefes Netz kann alles was flacheres Netz kann.
(Tiefes Netz = flaches Netz plus x1 Schichten)

Idee: Abkürzung (Shortcut) umgeht Schichten, Schicht lernt nur Änderung (Residuals) zum Vorgänger.

Vorteil: Direkter Pfad für Gradient auf tiefe Schichten bei SGD, Weight Decay „möchte“ x1 Schicht.

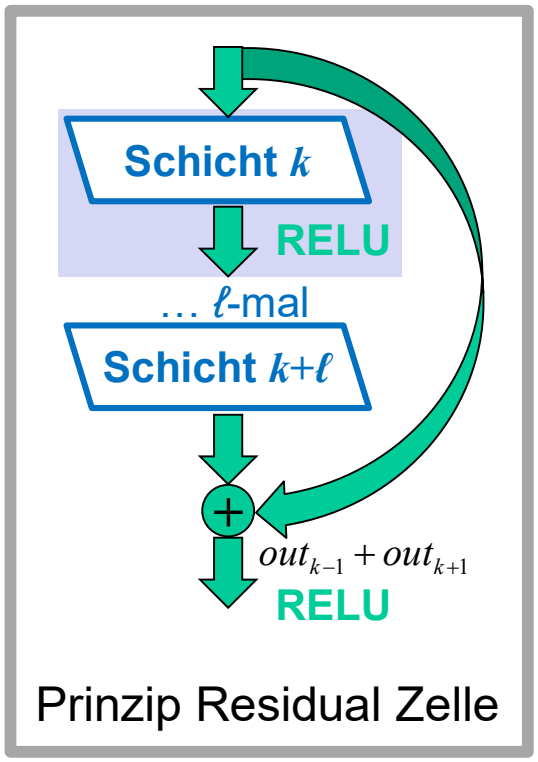
Residual Netz (ResNet)

TECHNISCHE HOCHSCHULE MITTELHESSEN

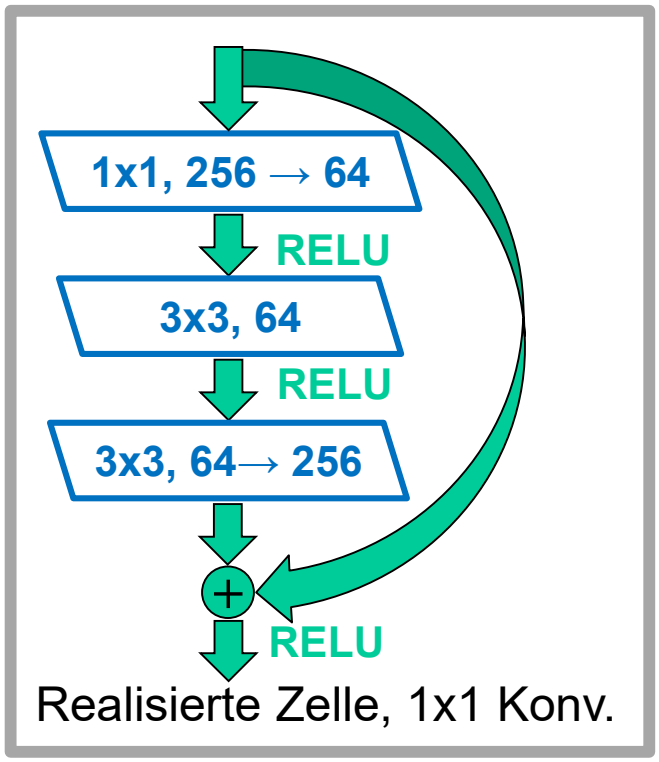
Idee: Schicht liefert nur Unterschied zum Vorgänger!

Realisation: Abkürzung (Shortcut) umgeht Schichten

Vorteil: Einfacher Zugriff auf tiefe Schichten bei SGD



Prinzip Residual Zelle



Realisierte Zelle, 1x1 Konv.

ResNet mit 152 Schichten gewinnt ILSVCR 2015 mit nur 11,3 Gflops:
5,71% Top-5 Fehler

Batch Normalisierung (BN)

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

Ioffe, Szegedy 2014: Normalisierung nicht nur Input, auch Zwischenschicht k

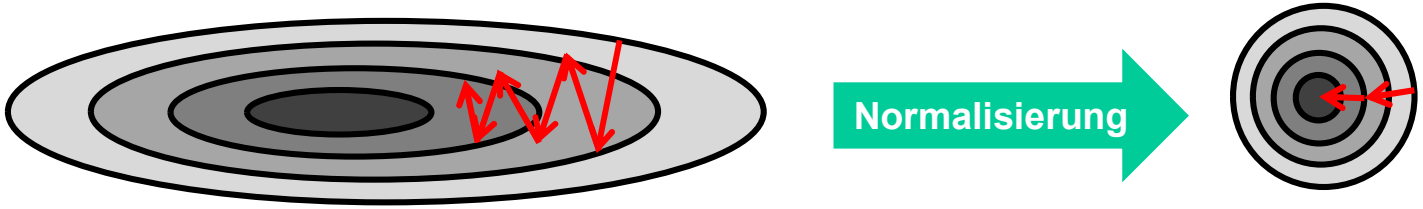
Normalisierung $x'_{ijk} = \frac{x_{ijk} - \overline{x}_{ik}}{\sigma_{ik}}$ und $x''_{ijk} = \gamma_{ik} \cdot x'_{ijk} + \beta_{ik}$

Mittelwerte $\overline{x}_{ik}$ und Standardabweichungen σ_{ik} nur über Stichproben des aktuellen Mini-Batches!

γ_{ik}, β_{ik}: 2 weitere lernbare Parameter pro Neuron

Damit könnte man z.B. BN rückgängig machen!

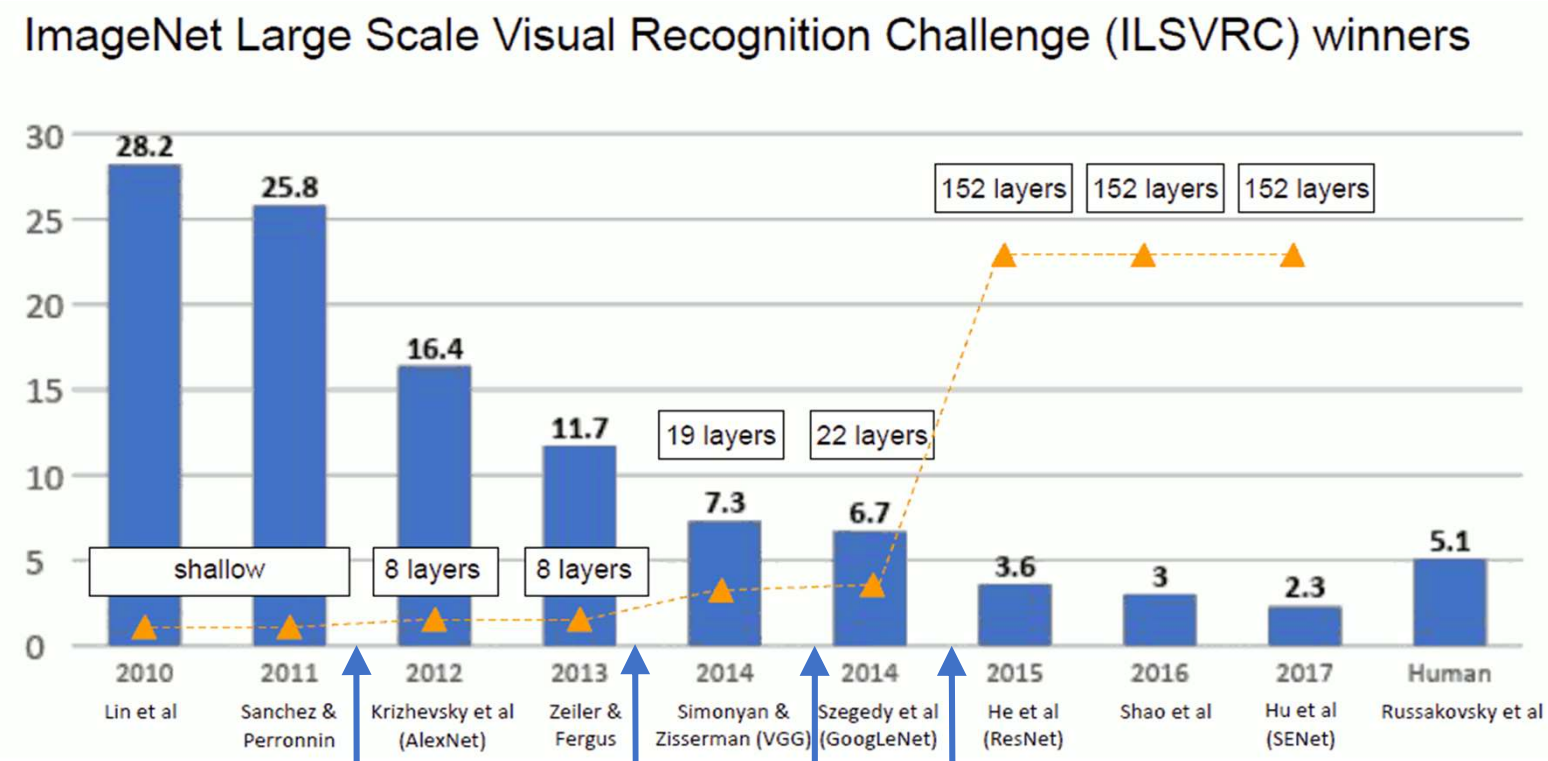
$\Rightarrow$ **Schnelleres Lernen** (auch durch grössere SGD-Schritte)



Unterschiedlich große Merkmale: trog-artiges Minimum Loss-Fkt., SGD hat Probleme

Ähnlich große Merkmale: rundes Minimum Loss-Fkt., SGD schnell!

ILSVRC: Fortschritte



Nach 2015:
Aufgabe erledigt!

Quelle Grafik: Stanford Kurs cs231n ;
Fei-Fei Li & Justin Johnson & Serena Yeung;
Convolutional Neural Networks for Visual Recognition

Status Mustererkennung ILSVRC

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

Aktuelle weitere Konzepte:

- Ausgefeilte Lernstrategien (Augmentierung)
- ReLu mit lernbarer Steigung < 0 4,94% Top-4, PReLU-Net 2015
- Batch Normalization 4,7% Top-4, Inception v.2 2015
- ResNet Zelle mit parallenen Konvolutionen 3,03% Top-5, ResNext 2016
- 1x1 Konv. weitergedacht (Dünnbesetzte Konv.) IGCNet, 2017
- Kombination unterschiedliche Netze 2,99% Top-5, Trimps-Soushen 2016
- Squeeze-And-Excitation 2,25% Top-5, SENet 2017
- Automatisierte Suche nach bestem Zellendesign NASNet 2018
- …

Trends:

-Benutze Netze oben als vortrainierte Merksmalgeneratoren

-Geschwindigkeitsoptimierung, z.B. für mobile Applikationen

Detect/Localize Objekts

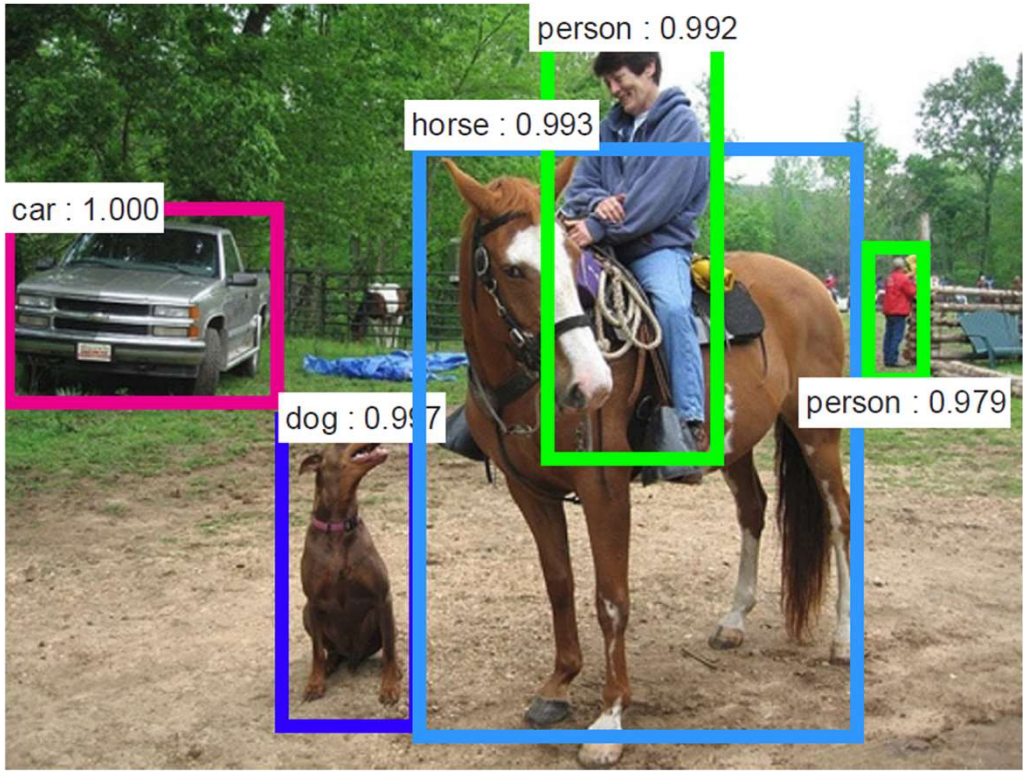
(ILSVCR seit 2012)

THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Menschliche Leistungsfähigkeit bei Mustererkennung erreicht, größere Herausforderung werden wichtiger!
Wettbewerbe/Datensammlungen: COCO, ImageNet

Aufgabe: Finde wo ist welches Objekt im (Bild)

- Wo?
  Umrandendes Rechteck
- Welche?
  5 wahrscheinlichste Klassen



Oft Strategie:

- Nehme gutes Netz für Mustererkennung
  (gut abgeschnitten bei ILSVCR)
- Anwenden auf Region-vorschläge

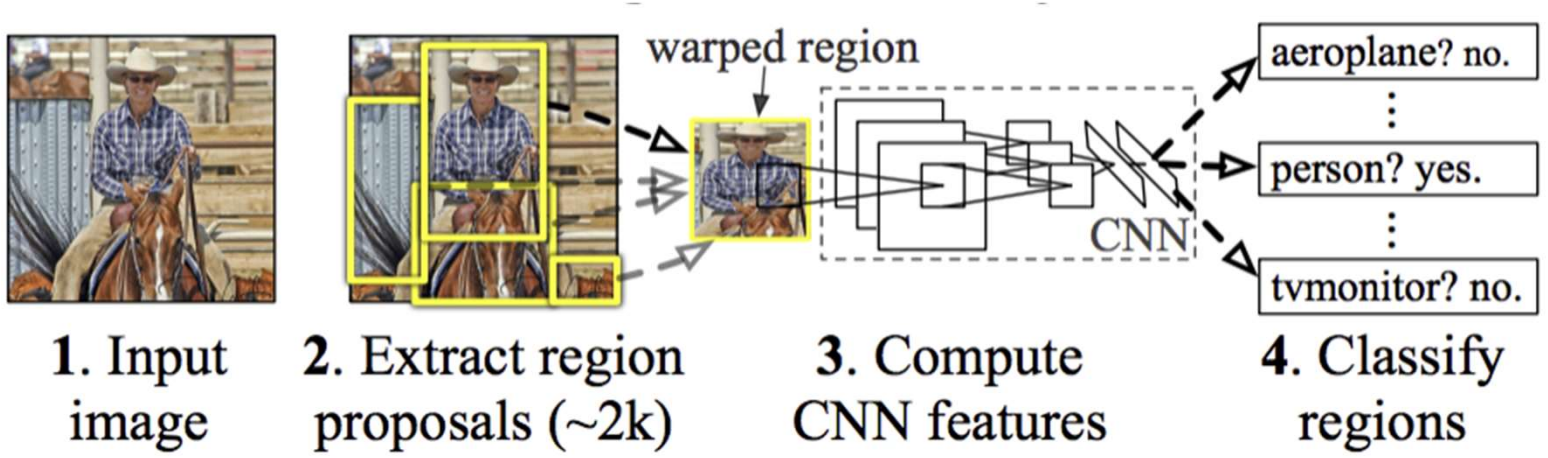
Z.B.: R-CNN Algo (und Optimierungen)

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

Methode: Regions with CNN features (R-CNN); Girshick et al. 2014
schneller: Fast R-CNN (2015), Faster R-CNN (2016)
Ziel: Lokalisiere/Detektiere Objekte

Algo Teil 1: Regionvorschläge

- Segmentiere Bild (ähnliche Farbe, Textur, …)
- Fasse Segmente zusammen (umrandendes Rechteck) zu einigen 1000 Regionen

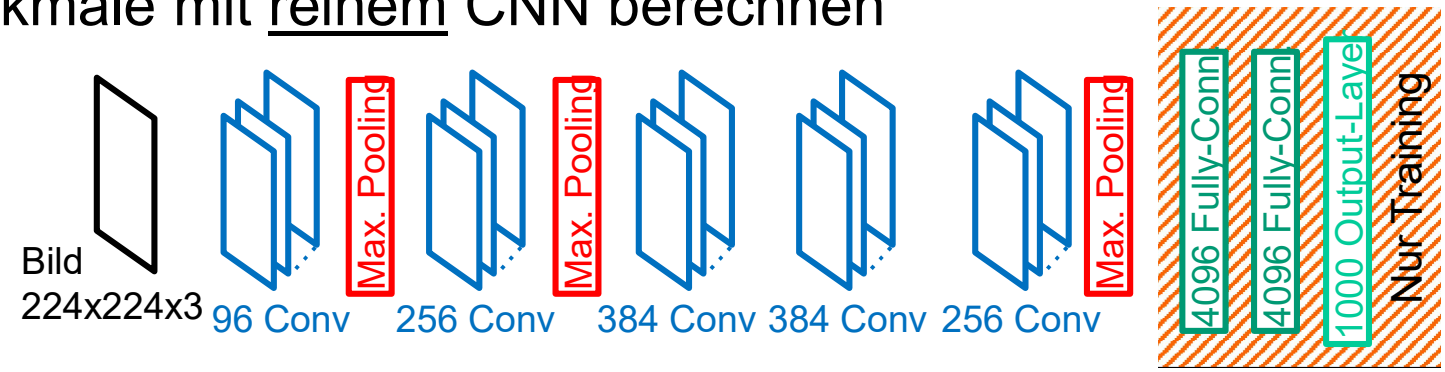


R-CNN Algorithmus

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

Algo Teil 2: Mustererkennung mit Regionvorschläge

- 4096 Merkmale mit reinem CNN berechnen

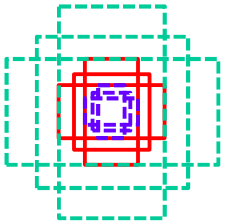


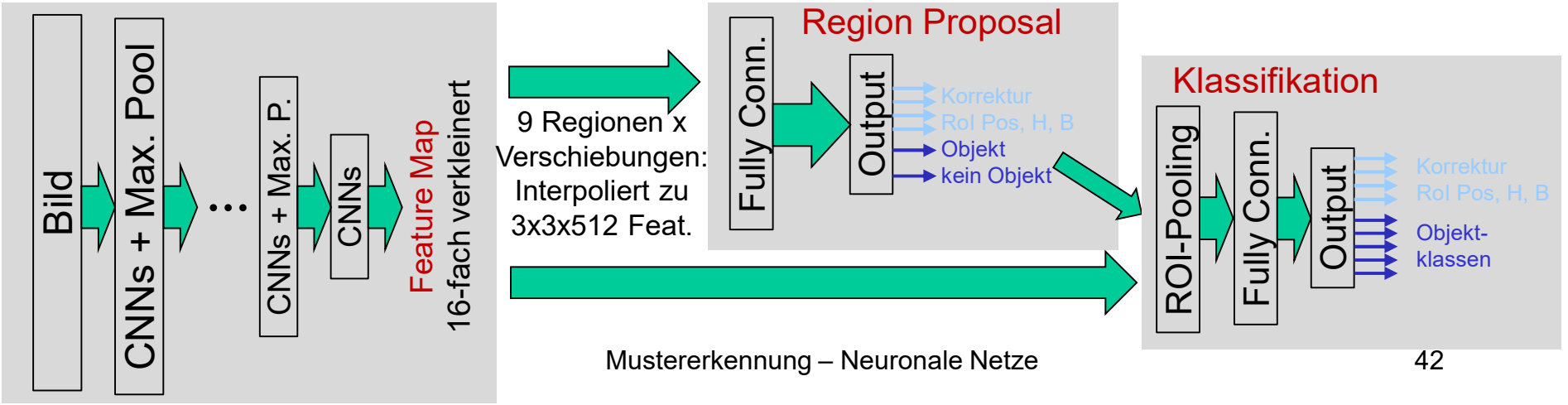
  Nur für Training: Ergänzt mit Fully Connected + Ausgabeschicht
- Erkennung: mit Support-Vektor-Machine, pro Klasse boolsche Zugehörigkeit (d.h. incl. „Keine Klasse“)
- Non-Maxima-Unterdrückung: Für grob gleiche Regionen* nur die mit höchstem Score berücksichtigen *Schnittmenge > 30% Vereinigung
- Effizienz?? Algo iteriert über Regionsvorschläge

Optimierung: Faster R-CNN

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

- Komplettes Bild als CNN-Input, Netzwerk zunächst NUR Konvolution
  $\Rightarrow$ für beliebig große Bilder anwendbar!
  Ergibt Feature Map
- Region Proposal 9 Regionen pro Position (Stride 16)
  Output pro Stride: Objekt/kein Obj., Eckenoffsets
- Klassifikation nur für Objekt-Regionen:
  - Region unterteilen in festes Gitter durch Max-Pooling
  - Ein fixes Netz für alle Regionen
- Lernen: L2 Loss für RoI, Cross-Entropie für Klassif., gewichtet Addiert
  Hyperparameter: Anteil der beiden!



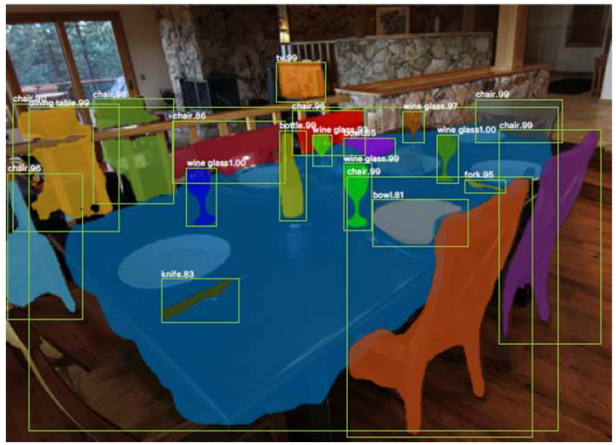


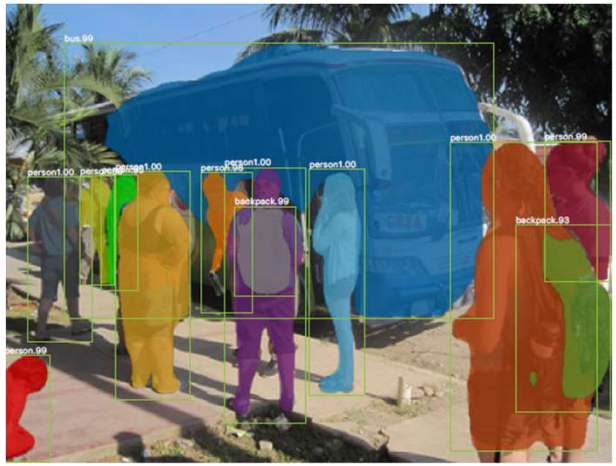
Mask R-CNN

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

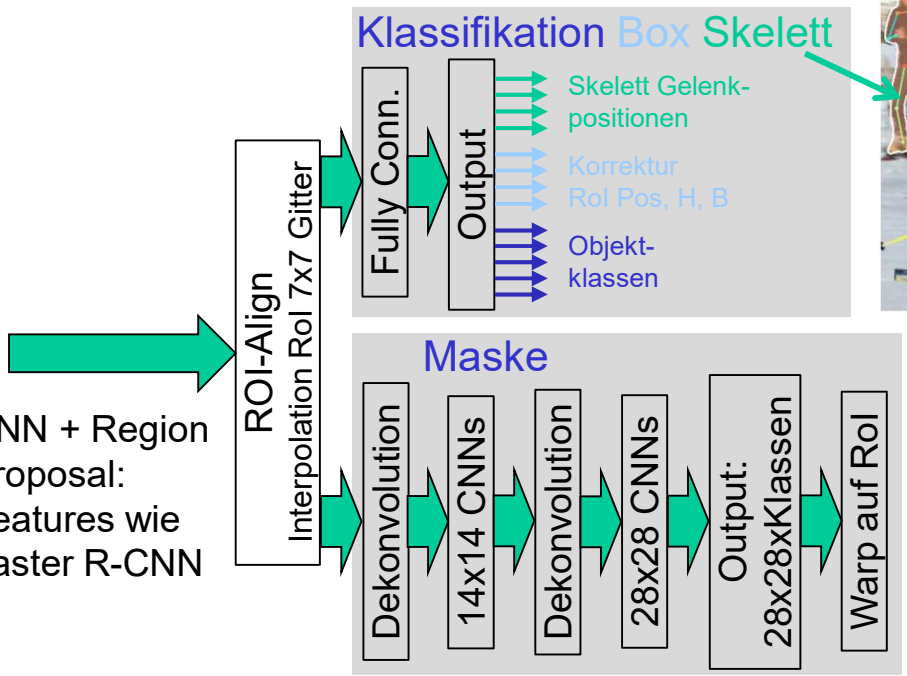
Nächst komplexere Aufgabe: semantisches Bildsegmentieren

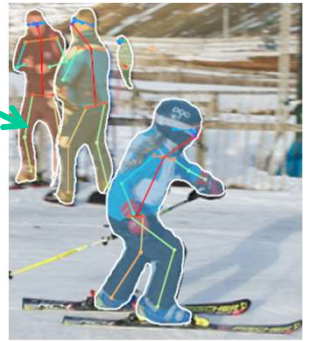
$\Rightarrow$ Label für jedes Pixel: Klasse + 1 (für Hintergrund)





CNN + Region Proposal: Features wie Faster R-CNN





z.B.: 14 Ge-lenkpos. pro Person

Dekonvolution: Vergrösserung z.B. 2x2 lernbare Gewichte w_{ij}

$$p'_{2i,2j} = w_{00} p_{ij}, \quad p'_{2i+1,2j} = w_{10} p_{ij}, \quad p'_{2i,2j+1} = w_{01} p_{ij}, \quad p'_{2i+1,2j+1} = w_{11} p_{ij}$$