

Convolutional Neural Networks

Christian Wilms

Computer Vision Group
Universität Hamburg

Wintersemester 17/18

28. November 2017

Übersicht

1 Klassifikation - Machine Learning Perspective

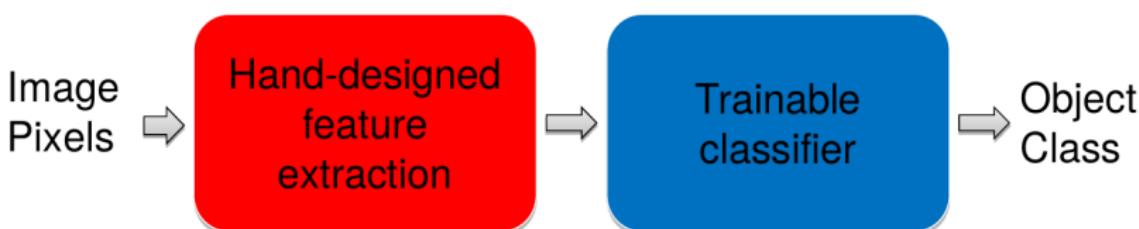
2 Neuronale Netze

3 Convolutional Neural Networks

4 Deep Learning mit GPUs

Pipeline der Klassifikation

Klassifikation wie wir sie bisher gemacht haben...



- Mittelwert
 - Standardabweichung
 - Histogramme
 - Seitenverhältnis
 - HOG
 - ...
-
- Nearest Neighbour Klassifikator
 - *k*-Nearest Neighbour Klassifikator
 - Multi Layer Perceptron

Merkmale

lokal

- Statistiken v. Obj.
- Seitenverhältnis
- Fläche

global

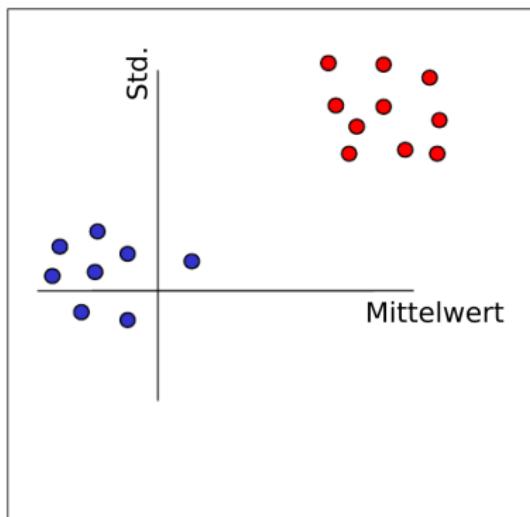
- Bildstatistiken
- Histogramme
- HOG

Was wir bisher nicht aktiv betrachtet haben

Hierarchien von Merkmalen:

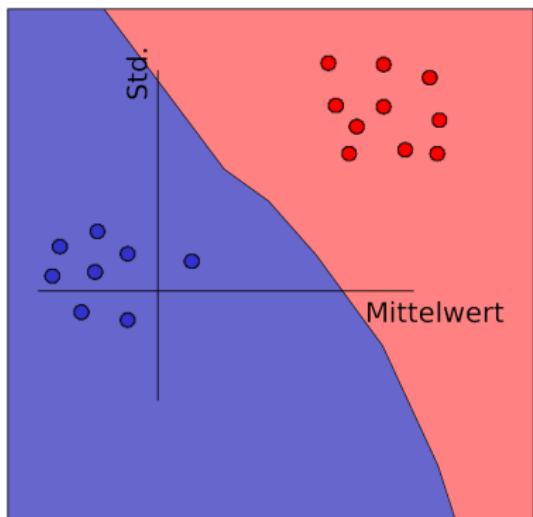
- Auto: Karosserie, Räder
- Räder: Kreise
- Kreise: Kantenzüge
- Kantenzüge: Kanten

Klassifikator



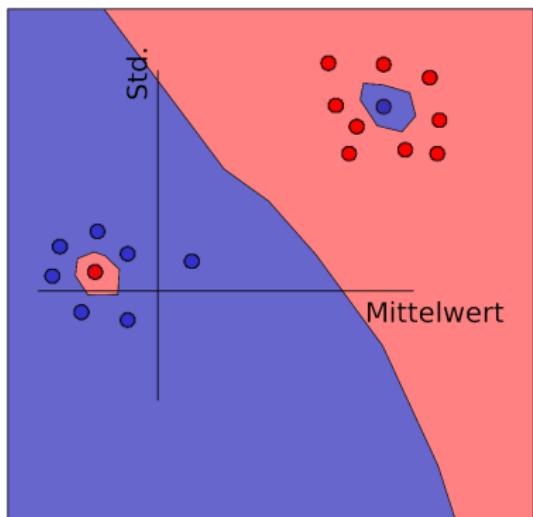
- es werden n Merkmale berechnet
- die Bilder liegen in einem n -dimensionalen Raum (hier: 2D)
- Wie werden die beiden Klassen gut voneinander getrennt?
- wir brauchen eine Grenze zwischen den Klassen

Klassifikator - Nearest Neighbour



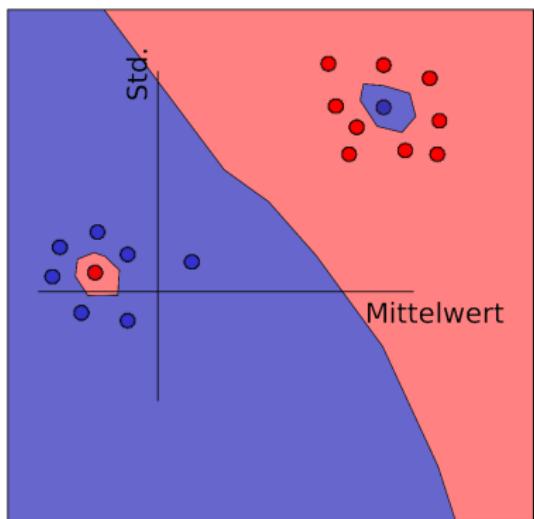
- wir kennen schon den 1NN-Klassifikator
- hier wird implizit eine Entscheidungsgrenze (Decision Boundary) erstellt
- Nachteile: Speicher, Geschwindigkeit, Probleme mit Ausreißern

Klassifikator - Nearest Neighbour



- wir kennen schon den 1NN-Klassifikator
- hier wird implizit eine Entscheidungsgrenze (Decision Boundary) erstellt
- Nachteile: Speicher, Geschwindigkeit, Probleme mit Ausreißern

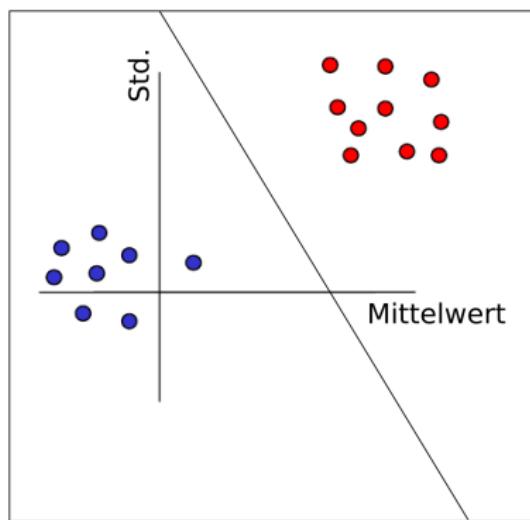
Klassifikator - Nearest Neighbour



- wir kennen schon den 1NN-Klassifikator
- hier wird implizit eine Entscheidungsgrenze (Decision Boundary) erstellt
- Nachteile: Speicher, Geschwindigkeit, Probleme mit Ausreißern

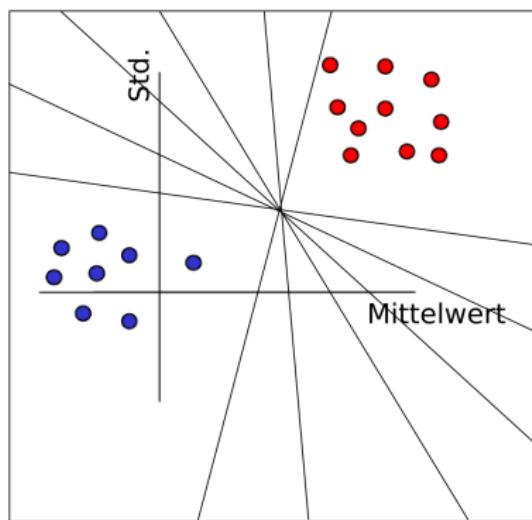
Wir brauchen etwas Robusteres!

Klassifikator - Parametrisches Modell



- Funktion für die Entscheidungsgrenze finden
- hier linear: $y = f(\vec{x}) = \vec{w}^T \vec{x} + b$
- \vec{x} ist der Deskriptor oder das Bild
- y gibt mit dem Vorzeichen die Klassenzugehörigkeit
- \vec{w} ist ein Vektor von Gewichten
- b ist ein Gewicht als Skalar
- Hinweis: für mehr als zwei Klassen lässt sich das Modell generalisieren

Klassifikator - Parametrisches Modell



- Funktion für die Entscheidungsgrenze finden
- hier linear: $y = f(\vec{x}) = \vec{w}^T \vec{x} + b$
- \vec{x} ist der Deskriptor oder das Bild
- y gibt mit dem Vorzeichen die Klassenzugehörigkeit
- \vec{w} ist ein Vektor von Gewichten
- b ist ein Gewicht als Skalar
- Hinweis: für mehr als zwei Klassen lässt sich das Modell generalisieren

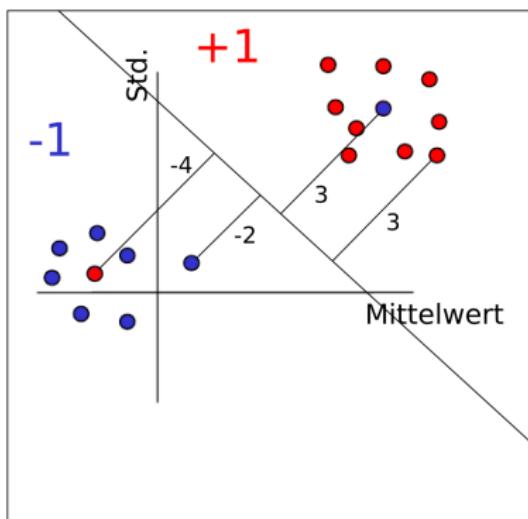
Wie können wir \vec{w} und b optimal bestimmen?

Klassifikator - Entscheidungsgrenze bewerten

Wir brauchen ein mathematisches Kriterium, das die Güte der Entscheidungsgrenze ermittelt → Loss-Funktion.

Klassifikator - Entscheidungsgrenze bewerten

Wir brauchen ein mathematisches Kriterium, das die Güte der Entscheidungsgrenze ermittelt → Loss-Funktion.



Loss-Funktion

- bewertet die Entscheidungsgrenze
- berechnet gegeben eine Entscheidungsgrenze wie viele falsche Klassifikationen in der Trainingsmenge gemacht werden und wie falsch sie sind
- es gibt viele verschiedene Loss-Funktionen

Entscheidungsgrenze optimieren

Wie kann nun, gegeben eine Loss-Funktion, die beste Entscheidungsgrenze gefunden werden? Woran können wir überhaupt drehen?

Woran wir drehen können:

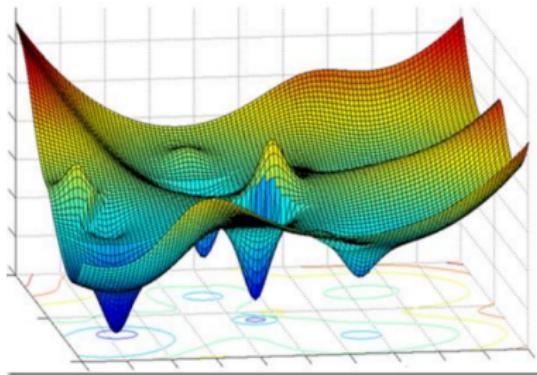
Gewichte \vec{w}, b

Die beste Entscheidungsgrenze ist die, welche die Loss-Funktion L minimiert.

Optimierungsproblem

- ① mit beliebigen/zufälligen Werten für \vec{w} und b starten
- ② Loss / berechnen über Loss-Funktion L
- ③ Änderung von \vec{w} und b berechnen → Gradient
- ④ zurück zu 1

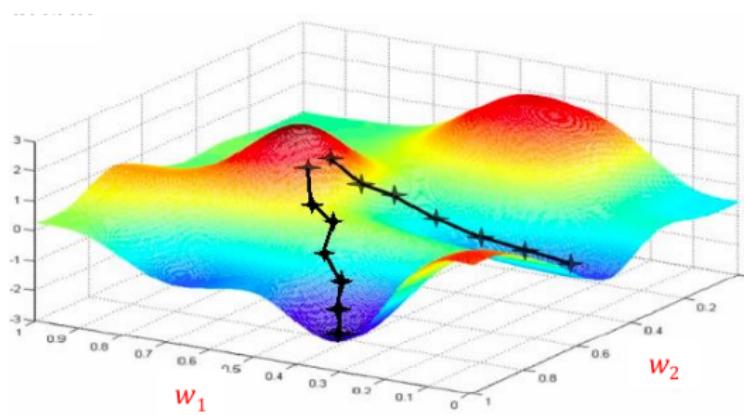
Gradientenabstieg - I



- Parameterraum im Bild
2-dimensional
- die Fläche visualisiert den jeweiligen Loss für verschiedene Parameterkombinationen
- man kennt nur einzelne Punkte
- die Täler sind die interessanten Stellen

Gradientenabstieg - II

- gegeben eine Startkombination lässt sich der Gradient der Loss-Funktion an dieser Stelle berechnen (part. Ableitungen)
- entlang des Gradienten können wir nun absteigen
- nach einigen Iterationen gelangen wir zu einem lokalen Minimum
- die Lösung ist nicht unbedingt eindeutig oder optimal



Übersicht

1 Klassifikation - Machine Learning Perspective

2 Neuronale Netze

3 Convolutional Neural Networks

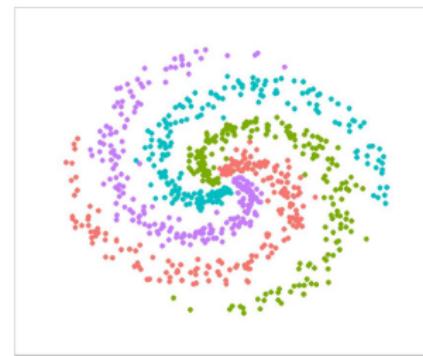
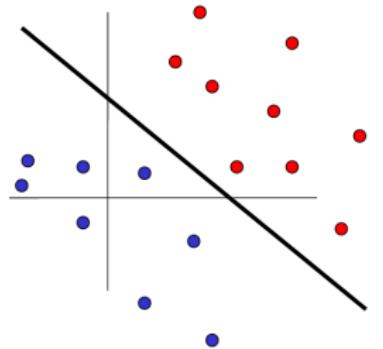
4 Deep Learning mit GPUs

Was nun?

Wie kommen wir nun zu einer Entscheidungsgrenze?

Lösungen

- lineares Modell nutzen und selber Parameter raten
- ein Machine Learning-Verfahren nutzen, um auch nicht lineare Probleme zu lösen → es ist schwierig alle möglichen Funktionen für eine Entscheidungsgrenze auszuprobieren



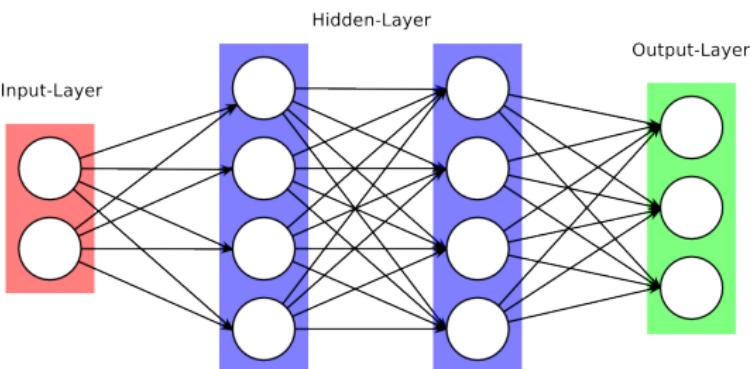
Neuronale Netze

Basis-Prinzip

- abgeleitet von der Funktionsweise eines Gehirn
- Neuronen (Perzeptron) bekommen viele Inputs, die gewichtet und addiert werden
- ist eine Schwelle dabei überschritten feuert das Neuron
- die Neuronen sind in Netzwerken mit verschiedenen Schichten angeordnet
- Entscheidungsgrenze wird durch die Gewichte an allen Neuronen bestimmt → VIELE Gewichte
- am Ende wird ein neues Bild/Deskriptor eingegeben und das Netz gibt ein Label als Antwort

Ein Neuronales Netz kann für die Klassifikation von 2-Klassen-Probleme oder auch n -Klassen-Problem genutzt werden.

Struktur eines Neuronalen Netzes



fully connected layer - jeder mit jedem verbunden

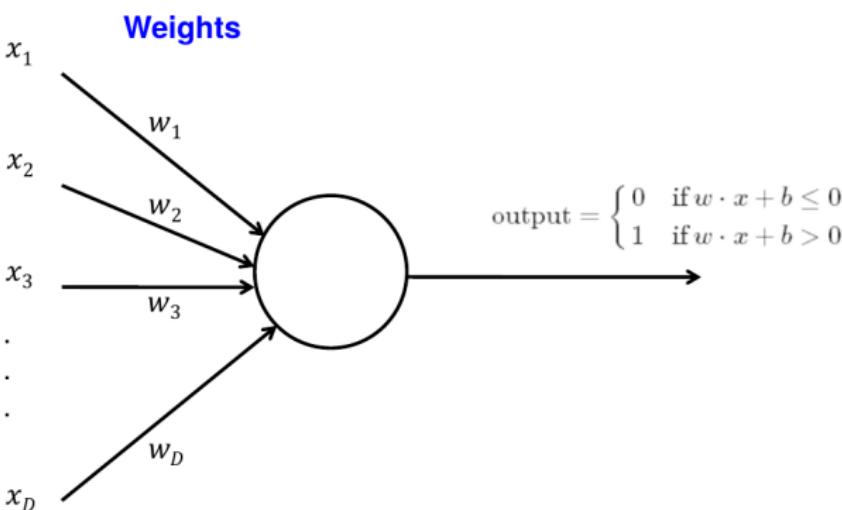
Input repräsentieren die Merkmale als Deskriptor oder die Pixel

Hidden kombinieren die Eingaben, hier werden aus Merkmalen neue Merkmale kreiert

Output fassen die obersten Merkmale zu einem Ergebnis pro Klasse zusammen

Ein Neuron - linear

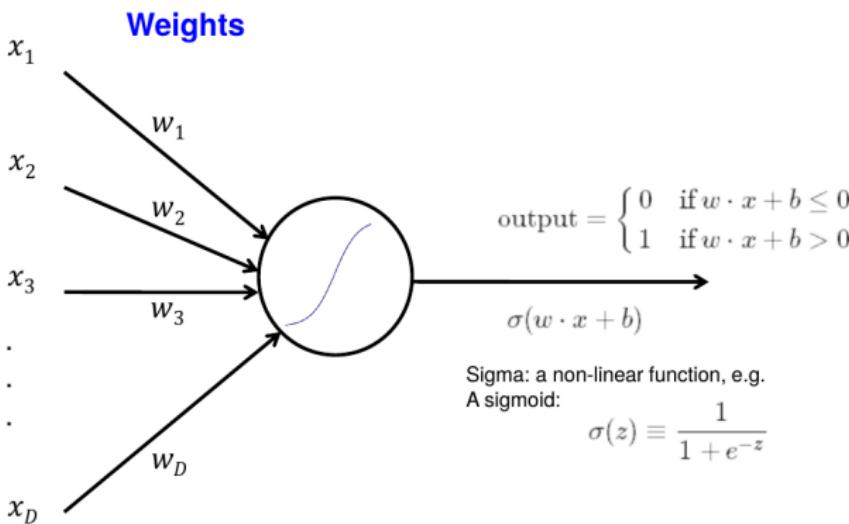
Input



Noch immer ist die Ausgabe eine lineare Kombination der Eingabe
→ Entscheidungsgrenze linear!

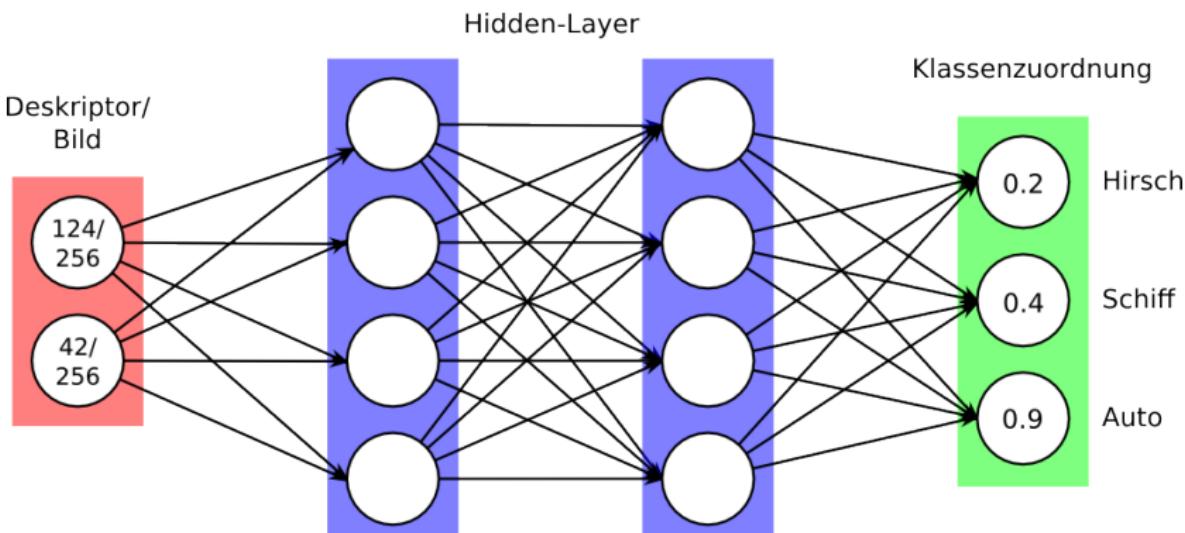
Ein Neuron - nicht-linear

Input



Durch die Anwendung einer nicht linearen **Aktivierungsfunktion** auf die Summe, entsteht eine nicht lineare Entscheidungsgrenze!

Beispiel Fully Connected Neural Network



Wie kann das Netz jetzt trainiert werden?

Training von Neuronalen Netzen

Üblicherweise werden die Trainingsdaten mehrfach durch das Netz geschickt, ein Durchgang wird dabei Epoche genannt:

- ① Teil der Trainingsmenge auswählen (Batch)
- ② Batch durch das Netzwerk schicken
- ③ Loss / über das gesamte Batch berechnen
- ④ Gradienten $\frac{\partial L}{\partial w_i}$ der Loss-Funktion L bestimmen
- ⑤ alle Gewichte entspr. des Gradienten updaten $w_i \leftarrow w_i - \alpha \frac{\partial L}{\partial w_i}$

Um alle Gewichte zu updaten, müssen die partiellen Ableitungen zu allen Gewichten gebildet werden.

$$L(f_i(f_j(f_k(\dots, \vec{w}_k, b_k), \vec{w}_j, b_j), \vec{w}_i, b_i)))$$

Der Loss oder Error wird dadurch durch das Netz zurück propagiert → Backpropagation.

Übersicht

1 Klassifikation - Machine Learning Perspective

2 Neuronale Netze

3 Convolutional Neural Networks

4 Deep Learning mit GPUs



Fully Connected Neural Network und Bilder

Probleme von FCNN auf Bildern

- recht viele Parameter → viele Parameter heißt **viele Trainingsbeispiele**
- alle partiellen Ableitungen zu berechnen dauert sehr lange → Training ist **sehr aufwendig**
- nicht ganz intuitiv zu designen → für Anzahl Layer und Neuroens gibt es **nur Faustregeln**
- erstmal nur ein Klassifikator → wir müssen die Merkmale selber bauen oder das ganz Bild als Eingabe nehmen

Lösungen

Flickr+Community (Imagenet), GPUs, Erfahrung

Deep Learning

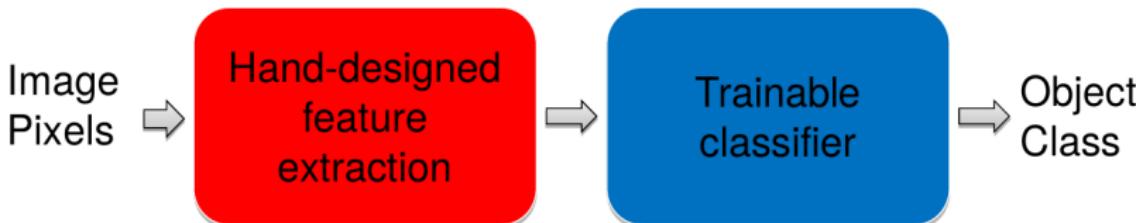
Wir wollen die Merkmale nicht mehr selber bauen!

- Merkmale lassen sich hierarchisch aufbauen wie die Netze
- jeder Layer könnte bestimmte Merkmale aus den vorherigen Layern kombinieren
- erster Layer lernt Kanten, zweiter Kantenzüge, dritter Kreise,..., n -ter lernt Autos

Deep Learning

Wir wollen die Merkmale nicht mehr selber bauen!

- Merkmale lassen sich hierarchisch aufbauen wie die Netze
- jeder Layer könnte bestimmte Merkmale aus den vorherigen Layern kombinieren
- erster Layer lernt Kanten, zweiter Kantenzüge, dritter Kreise,..., n -ter lernt Autos



Deep Learning

Wir wollen die Merkmale nicht mehr selber bauen!

- Merkmale lassen sich hierarchisch aufbauen wie die Netze
- jeder Layer könnte bestimmte Merkmale aus den vorherigen Layern kombinieren
- erster Layer lernt Kanten, zweiter Kantenzüge, dritter Kreise,..., n -ter lernt Autos



Deep Learning

Wir wollen die Merkmale nicht mehr selber bauen!

- Merkmale lassen sich hierarchisch aufbauen wie die Netze
- jeder Layer könnte bestimmte Merkmale aus den vorherigen Layern kombinieren
- erster Layer lernt Kanten, zweiter Kantenzüge, dritter Kreise,..., n -ter lernt Autos

Das Bild als Eingabe

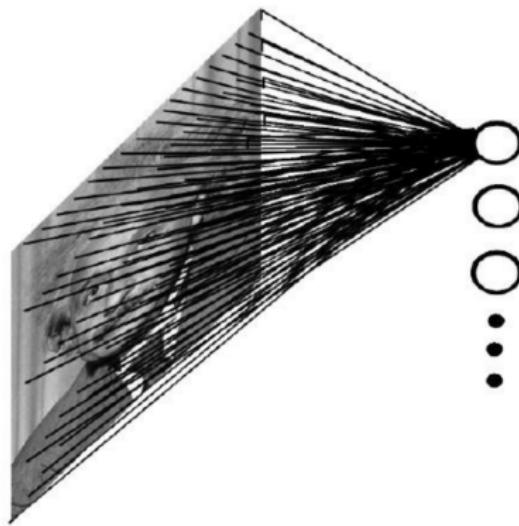
- Bild-Größe 224×224
- 3 Hidden-Layer mit 1024 Neuronen
- 10 Output-Neuronen für 10 Klassen

⇒ über 53 Millionen Parameter ohne große Tiefe oder Breite

Convolutional Neural Networks (CNN)

Beobachtungen

- die meisten Gewichte sind zwischen dem ersten Layer und dem Bild zu finden (ca. 51 Millionen)
- für einfache Merkmale braucht man nur einen sehr begrenzten lokalen Bereich

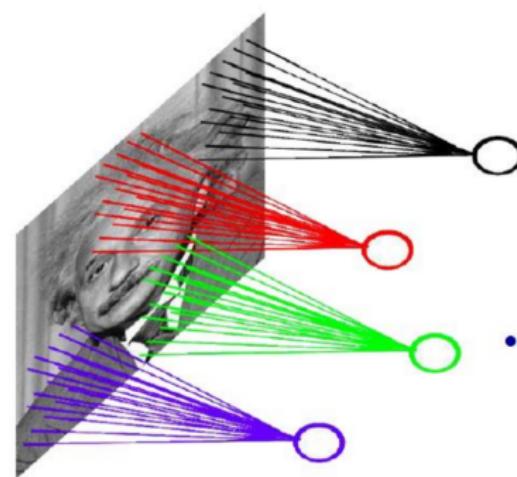


Convolutional Neural Networks (CNN)

Beobachtungen

- die meisten Gewichte sind zwischen dem ersten Layer und dem Bild zu finden (ca. 51 Millionen)
- für einfache Merkmale braucht man nur einen sehr begrenzten lokalen Bereich

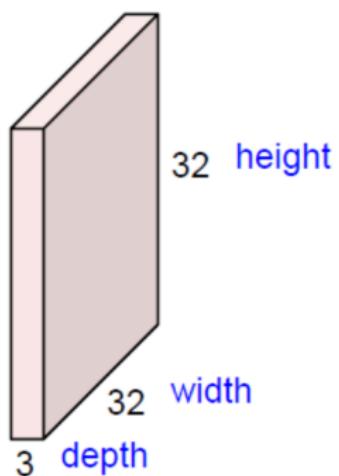
Wir benutzen Faltungen und lernen die Gewichte in die Faltungskerne!



$$w = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix}$$

Convolutional Layer

32x32x3 image



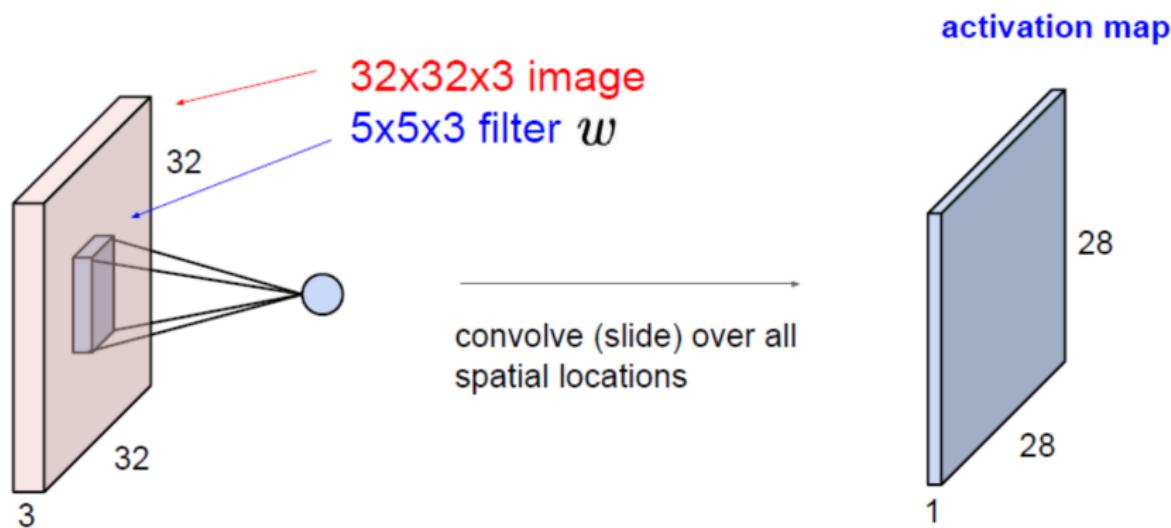
5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

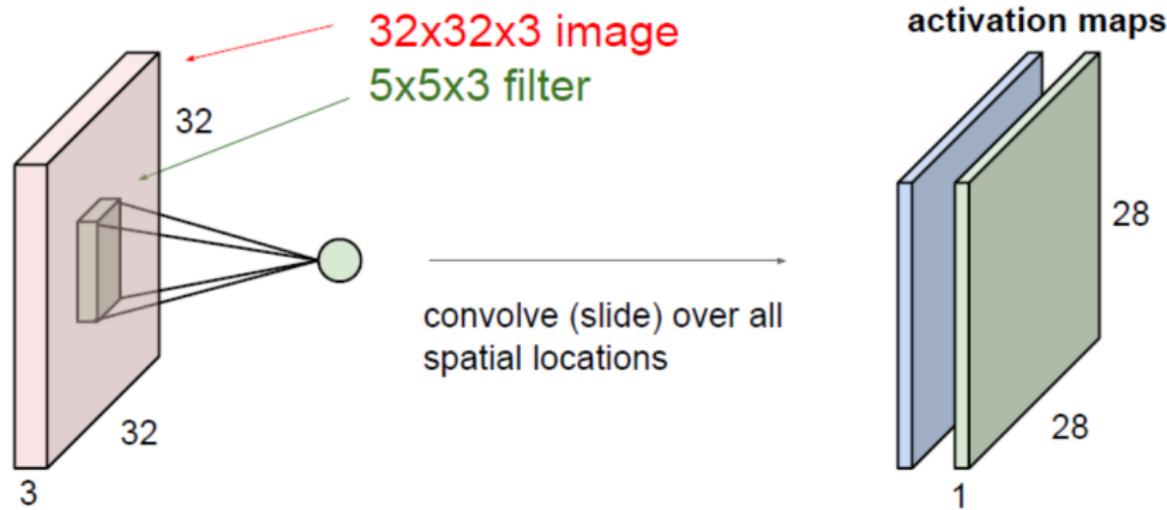
Wir falten das RGB-Bild mit einem 3D-Faltungskern (hier 5×5 ,
aber oft 3×3)

Convolutional Layer



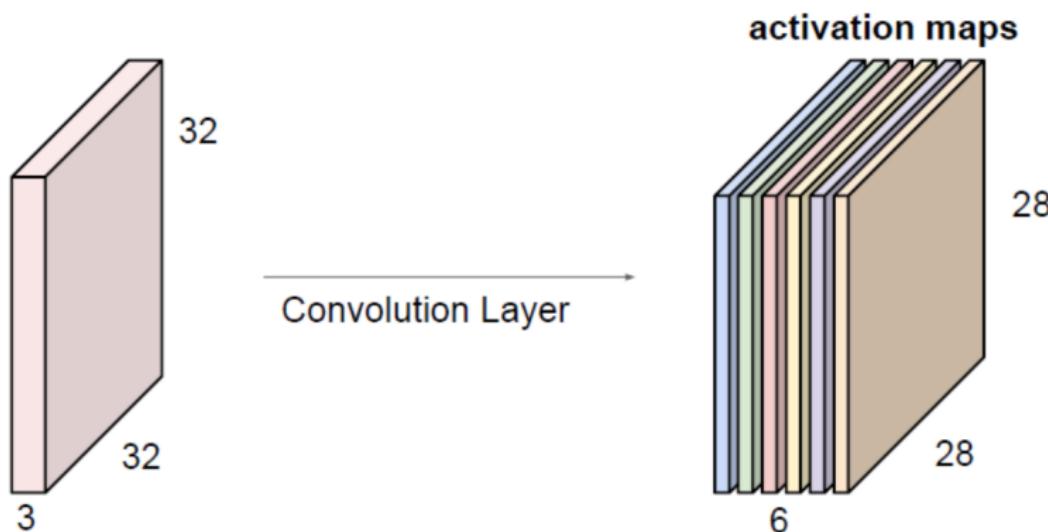
Das Ergebnis ist ein "Graustufenbild", das als Merkmalskarte interpretiert werden kann

Convolutional Layer



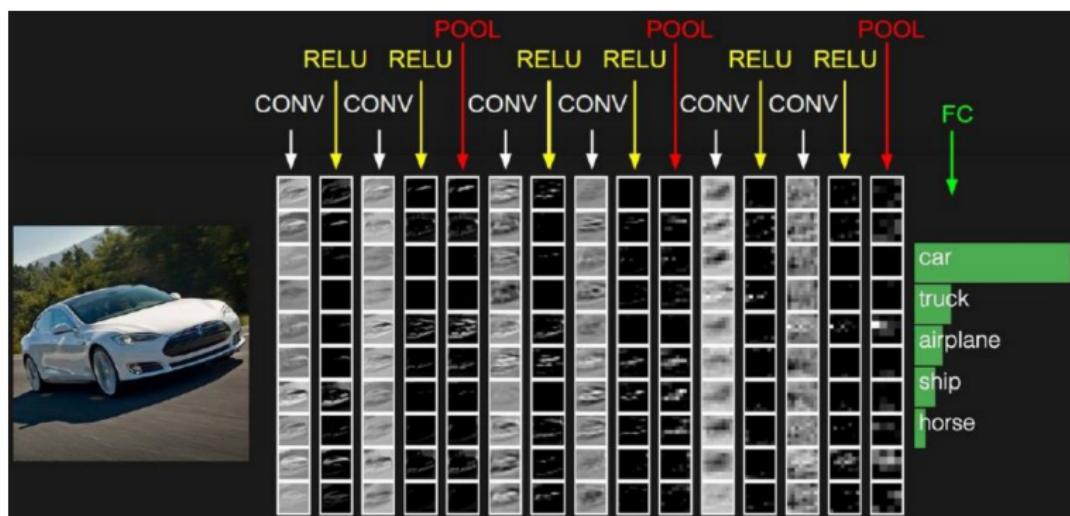
Dies wir nicht nur einmal gemacht...

Convolutional Layer



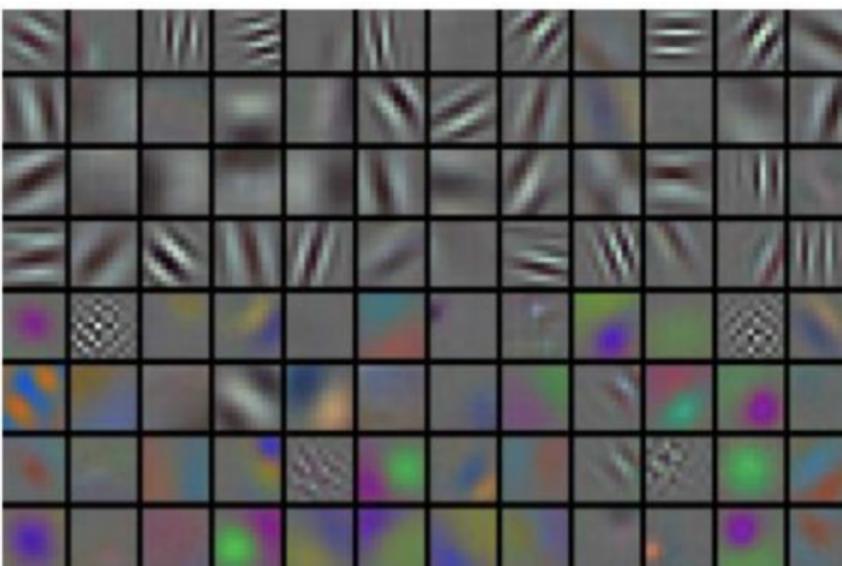
sondern oft. Es entsteht somit ein neues "Bild" mit vielen Kanälen für verschiedene Merkmale.

CNN-Struktur



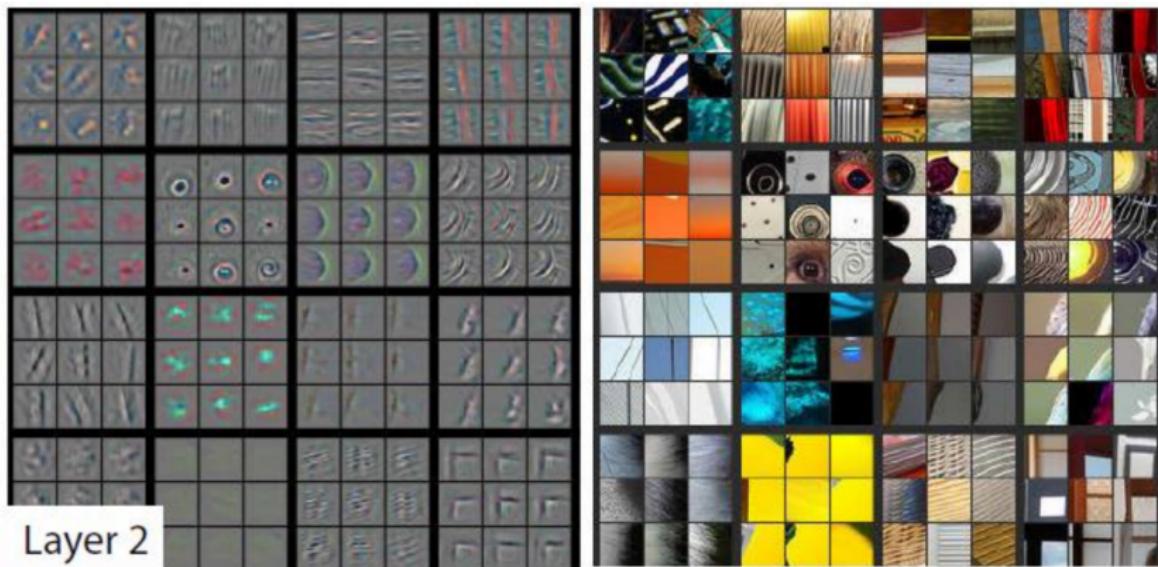
Prinzip CNN: Convolutional Layer werden aufeinander gestapelt/angewendet → es entsteht eine Hierarchie von Merkmalen

Welche Merkmale werden gelernt?



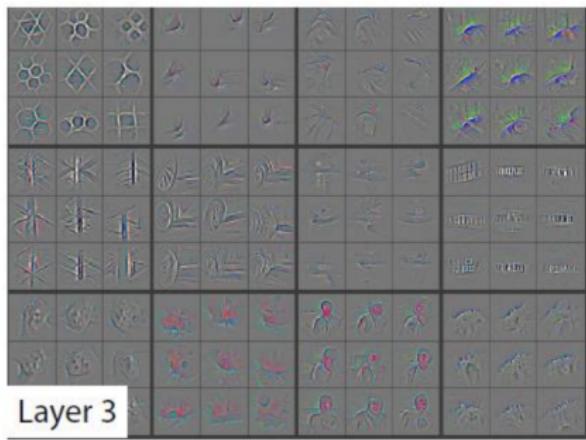
Erkannt werden Kontraste, Kanten und Wellen.

Welche Merkmale werden gelernt?



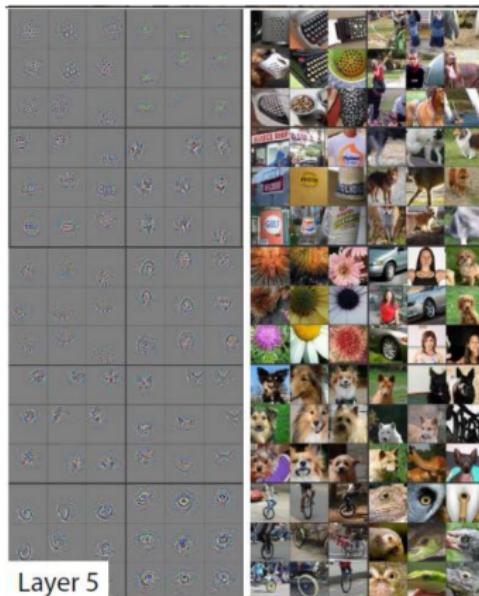
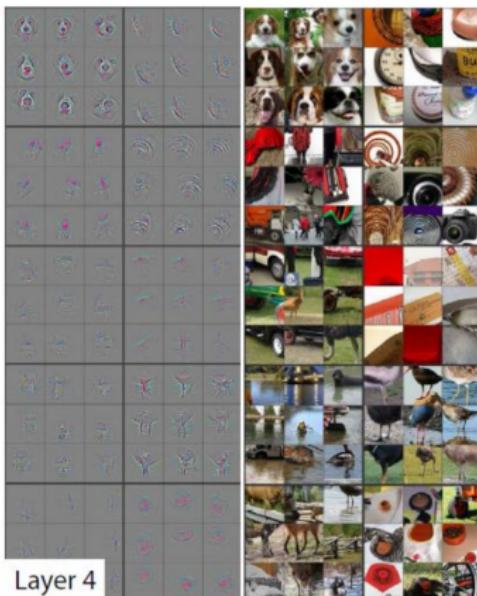
Erkannt werden Formen und homogene Flächen.

Welche Merkmale werden gelernt?



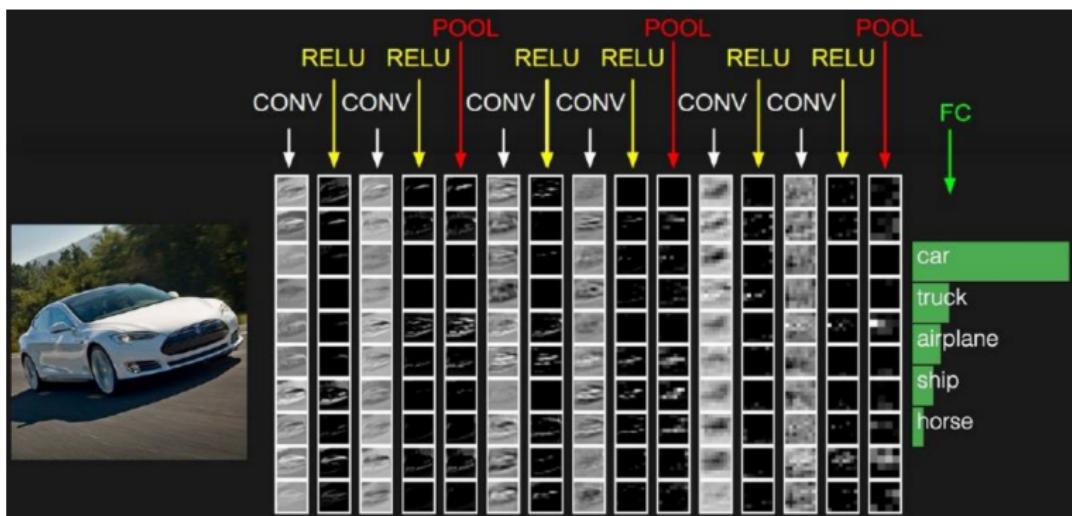
Erkannt werden Texturen und Objektteile.

Welche Merkmale werden gelernt?



Erkannt werden Objekte und Objektteile.

CNN-Struktur



Was ist mit den anderen Layern: RELU, POOL und FC?

Weitere Layer

ReLU

Pooling

Fully-Connected (FC)

Weitere Layer

ReLU

- ReLU (Rectified Linear Unit) dient als Aktivierungsfunktion
- erzeugt also die Nichtlinearität
- ähnlich der Sigmoid-Funktion vorher

Pooling

Fully-Connected (FC)

Weitere Layer

ReLU

Pooling

- beim Pooling wird aus jeweils $n \times n$, meist 2×2 "Pixeln" das Maximum (Max-Pooling) genommen
- diese $n \times n$ Fenster überlappen sich nicht
- wird auf jedem Kanal individuell angewendet
- sorgt für eine Verkleinerung bei zunehmender Tiefe
- reduziert die Anzahl an Berechnungen
- es ist für die Klassifikation auch nicht entscheidend wo das Auto im Bild ist

Fully-Connected (FC)

Weitere Layer

ReLU

Pooling

Fully-Connected (FC)

- nach dem letzten Conv-Layer mit ReLU und Pooling wird das Ergebnisbild abgerollt
- die "Pixel" dienen nun als Eingabe in ein normales Neuronales Netz
- hier ein Neuronales Netz zu benutzen hat den Vorteil, dass es weiterhin alles zusammen trainiert werden kann

Training

Ein CNN trainiert sich vom Prinzip exakt so wie ein normale Neuronales Netz.

From Scratch

- Training wie vorher beschrieben
- nur sinnvoll, wenn der Datensatz groß ist oder das Netz klein

Fine-tuning

- das Netz wurde bereits auf einem (großen) Datensatz trainiert
- viele Merkmale werden vermutlich ähnlich sein
- daher werden nur die FC-Layer oder die obersten Conv-Layer trainiert

CNN als Merkmalsextraktor: gar kein Training nötig.

Wie soll mein CNN aussehen?

- für den Anfang solltet ihr mit kleinen Netzen arbeiten (z.B. 2 bis 5 Layer mit je 32 Faltungskernen)
- es gibt auch bereits eine ganze Reihe Standard-Netze, die deutlich größer sind

AlexNet 5 Conv-Layer

VGG 12 oder 14 Conv-Layer

GoogLeNet viele

Res-Net noch mehr

- für diese Netze lassen sich auch bereits die Imagenet-Gewichte herunterladen → nur noch fine-tuning nötig

Übersicht

- 1 Klassifikation - Machine Learning Perspective
- 2 Neuronale Netze
- 3 Convolutional Neural Networks
- 4 Deep Learning mit GPUs

Wie kann ich das alles nutzen?

Technische Voraussetzungen

- je tiefer das Netz, umso länger dauert alles
- kleine Netze kann man auf der CPU rechnen
- große Netze muss man auf der GPU rechnen
- geht aktuell quasi nur mit NVIDIA-GPUs → CUDA

Bibliotheken

Tensorflow Bibliothek für Deep Learning mit Python, die recht viel Spielraum für eigene Veränderungen lässt

Keras High-Level Bibliothek, die u.a. auf Tensorflow aufsetzt und die Benutzung tlw. stark vereinfacht

Basisprinzip von Keras

- Daten laden (Array mit Shape (Anz. Bilder, Höhe, Breite, Anz. Kanäle) dazu ein 1D-Array mit den Labels)
- es wird ein Model-Objekt definiert
- diesem werden alle Layer durch Methodenaufrufe hinzugefügt (wie einer Liste)
- Layer sind wiederum selbst Objekte
- dem Model wird ein Solver-Objekt übergeben, das die Informationen zur Backpropagation enthält
- das Model-Objekt hat eigene Methoden zum Kompilieren, Trainieren und Evaluieren

```
model = Sequential()
```

Layer in Keras

Flatten-Layer (Abrollen)

FC-Layer (Dense)

Conv-Layer

Pooling

Dem ersten Layer muss stets die `input_shape` der Daten gegeben werden!

Layer in Keras

Flatten-Layer (Abrollen)

`Flatten()`

FC-Layer (Dense)

Conv-Layer

Pooling

Dem ersten Layer muss stets die `input_shape` der Daten gegeben werden!

Layer in Keras

Flatten-Layer (Abrollen)

FC-Layer (Dense)

```
Dense(128, activation='relu', name='fc1')
```

Einen eigenen Layer für die Aktivierungsfunktion gibt es nicht!

Conv-Layer

Pooling

Dem ersten Layer muss stets die `input_shape` der Daten gegeben werden!

Layer in Keras

Flatten-Layer (Abrollen)

FC-Layer (Dense)

Conv-Layer

```
Convolution2D(32, 3, 3, activation='relu',  
name='conv1')
```

32 Faltungsmasken, je 3×3

Einen eigenen Layer für die Aktivierungsfunktion gibt es nicht!

Pooling

Dem ersten Layer muss stets die `input_shape` der Daten gegeben werden!

Layer in Keras

Flatten-Layer (Abrollen)

FC-Layer (Dense)

Conv-Layer

Pooling

`MaxPooling2D(pool_size=(2, 2))`

Dem ersten Layer muss stets die `input_shape` der Daten gegeben werden!

Model-Objekt in Keras

```
model = Sequential()
model.add(Flatten(input_shape=(28,28,1)))
model.add(Dense(128, activation='relu', name='fc1'))
model.add(Dense(128, activation='relu', name='fc2'))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer=SGD(lr=0.01, momentum=0.9),
              metrics=['accuracy'])
```

`categorical_crossentropy` eine Loss-Funktion für Klassifikation

`SGD` Stochastic Gradient Descent, Annäherungsverfahren
zur Gradientenbestimmung

`lr` Learning Rate, Schrittgröße beim Gradientenabstieg

`metrics` berechne die Accuracy

Wie man lernt

Aufteilung der Daten

Trainingsdaten werden genutzt, um die Gewichte (Parameter) des Modells zu tunen

Validierungsdaten nutzen um die Hyperparameter und das Modell zu bestimmen und das Training zu beenden (Anzahl der Trainingsiterationen)

Testdaten nur benutzen, um am Ende die Ergebnisse für den Report zu erzeugen

Wenn man das Training weiter laufen lässt, bis der Fehler auf den Trainingsdaten minimiert wurde, hat das Netz vermutlich die Trainingsdaten auswendig gelernt!

Daher, die Genauigkeit der Validierungsdaten beobachten und bei bestem Ergebnis/Plateau stoppen!

Training mit dem Model-Objekt

```
model.fit(X_train, Y_train, batch_size=32,  
          nb_epoch=10, validation_split = 0.2,  
          verbose=1)
```

`X_train` Bilder

`Y_train` Label hier als Array mit Shape (Anz. Bilder, Anz. Label) und einer 1 je Zeile

`batch_size` Größe eines Batches (Trainingsbilder für die zusammen der Loss berechnet wird)

`nb_epoch` Anzahl der Epochen (Durchläufe durchs Trainingsset)

`validation_split` Anteil der zur Validierung genutzten Trainingsdaten

`verbose` Konsolenausgabe einschalten

Auswertung mit dem Model-Objekt

```
score = model.evaluate(X_test, Y_test, verbose=1)
```

X_test Bilder

Y_test Label hier als Array mit Shape (Anz. Bilder, Anz. Label) und einer 1 je Zeile

verbose Konsolenausgabe einschalten

score Tupel aus Loss und Accuracy

Speichern und Laden von Gewichten

Speichern der Gewichte eines Modells:

```
model.save_weights('cifar10weights.h5')
```

Laden der Gewichte in eine anderes Modell:

```
model.load_weights('cifar10weights.h5', by_name=True)
```

Wichtig: Die Gewichte werden an Hand der Namen der Layer zugeordnet! D.h., Layer mit gleichem Namen müssen die gleiche Größe haben!

Für Standardnetze hat Keras eigene Funktionen zum Laden der Struktur plus vortrainierten Gewichten (Imagenet).

Standard-Datensätze

MNIST

- `keras.datasets.mnist`
- Datensatz mit handgeschriebenen Ziffern
- Label ist die geschriebene Zahl
- Bilder der Größe 28×28 in Graustufen

CIFAR-10

- `keras.datasets.cifar10`
- Datensatz zur Objektklassifizierung
- 10 verschiedene Objektklassen
- RGB-Bilder der Größe 32×32

Referenzen zu Keras und Deep Learning

Keras keras.io

Dokumentation zu Keras

Stanford Lecture on Deep Learning cs231n.github.io/
sehr gute Vorlesung zum Thema mit viel Material
und YouTube-Videos

Deep Learning Book DAS Deep Learning Buch mit sehr viel Inhalt:
Deep learning: Ian Goodfellow, Yoshua Bengio and
Aaron Courville, MIT Press, 2016

Präsentation Projektidee

- Stellt eure Domäne vor!
 - Was sind eure Klassen?
 - Wie sehen sie aus?
 - Worin unterscheiden sich die Klassen?
 - Was sind Gemeinsamkeiten?
- Was ist euer Plan?
 - Woher kommen die Daten?
 - Was für Merkmale wollt ihr nutzen?
 - Habt ihr euch eigene Merkmale überlegt?
 - Wollt ihr das Bild binarisieren?
 - Wird das Problem auch wissenschaftlich betrachtet?
 - Was soll euer Klassifikator sein?
 - Wollt ihr auch Deep Learning (CNNs) ausprobieren?

Das, was ihr nächste Woche erzählt, ist nicht in Stein gemeißelt!

DEMO

MNIST mit Neural Network (ohne Convolution) in Keras
klassifizieren.

Aufgabenblatt 6