(Künstliche) Neuronale Netze (Artificial) Neural Networks / Multi-Layer Perceptron

Christian Wilms

Computer Vision Group
Universität Hamburg

Sommersemester 2018

17. Mai 2018

Übersicht

Projektaufgabe

- im zweiten Teil des Praktikums wird es eine größere Aufgabe geben
- die Bearbeitung ist in Teams (4 Studis) vorgesehen

Aufgabenstellung

- Ihr sollt Bilder klassifizieren!
- Domäne ist völlig offen
- zwei Varianten
  - klassischer Ansatz (Merkmale selber wählen)
  - Deep Learning basiert
- es dürfen nur Methoden genutzt werden, die auch verstanden wurden
- im Praktikumsbericht wird die Lösung der Aufgabe beschrieben und evaluiert

Beispieldomänen

- Spielkarten zu Farben zuordnen
- Logos klassifizieren
- Fische den Arten zuordnen
- Landnutzung aus Google Earth-Bildern
- Haribo-Figuren den Tüten zuordnen
- Münzen den Ländern zuordnen
- Verkehrsschilder klassifizieren
- Lego-Figuren zu Themen klassifizieren
- Gemälde einem Maler zuordnen (Painter by Numbers @ kaggle.com)
- ...

Zeitplan

31. Mai

- Diskussion der Ideen mit den Kleingruppen

7. Juni

- Präsentation der Idee
- 5-10 Minuten Vortrag (eine Person)

12. Juli

- Abschlusspräsentation der Ergebnisse
- 10-15 Minuten Vortrag (eine/zwei Personen)

Daten sammeln

Datensätze aus dem Internet

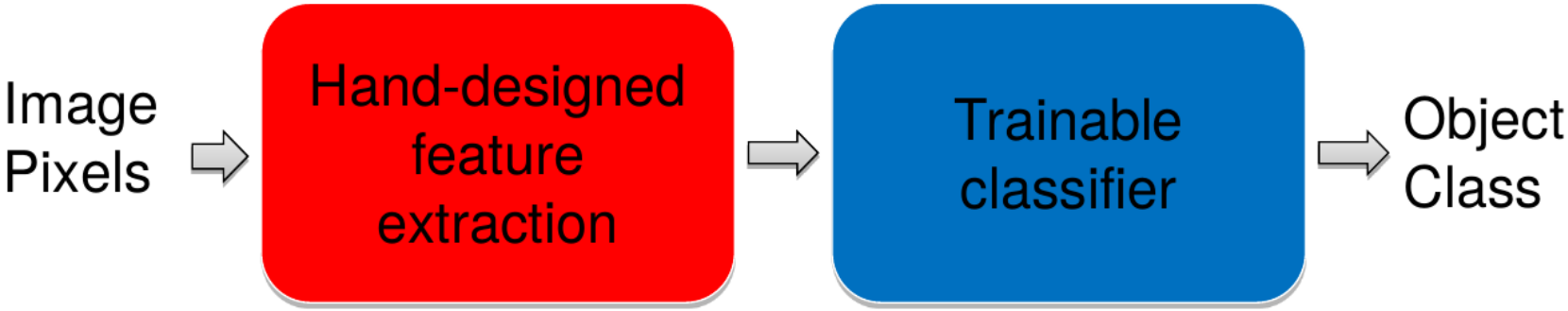
- durchs Internet geistern tausende von Datensätzen
- es gibt allgemeine Datensätze zur Klassifikation wie CIFAR-10 oder welche zu speziellen Problemen wie etwa Klassifikation von Verkehrsschildern
- eine von vielen (unvollständigen) Übersichten bietet: Yet Another Computer Vision Index To Datasets (YACVID)
- die Bilder in Datensätzen liegen teilweise in ganz unterschiedlichen Formaten vor

Datensätze selber machen

- Fotos mit einer Kamera
- Bilder aus dem Internet, etwa von Flickr oder Google Earth

Übersicht

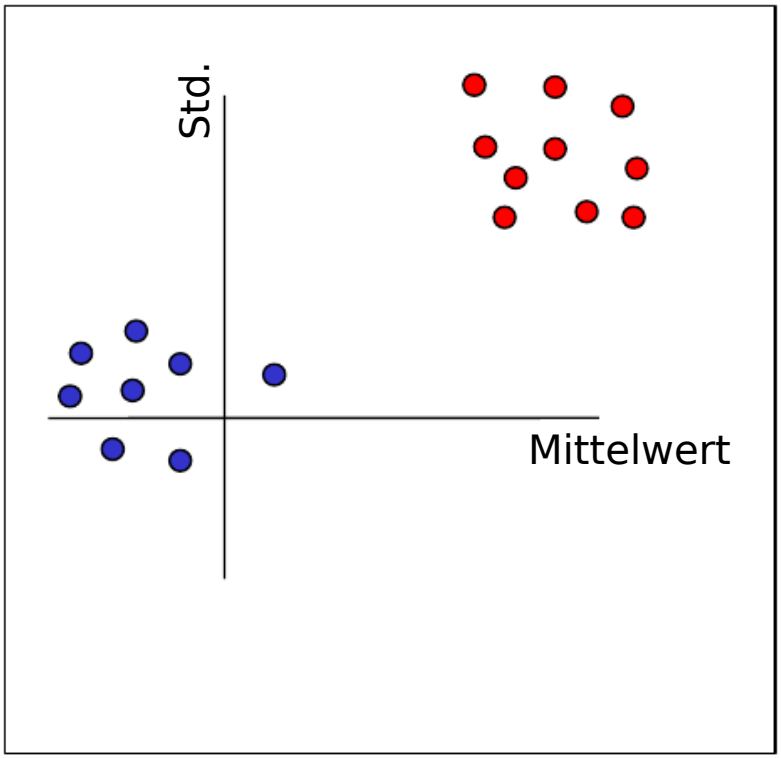
Pipeline der Klassifikation



- Mittelwert
- Standardabweichung
- Histogramme
- Seitenverhältnis
- HOG
- ...

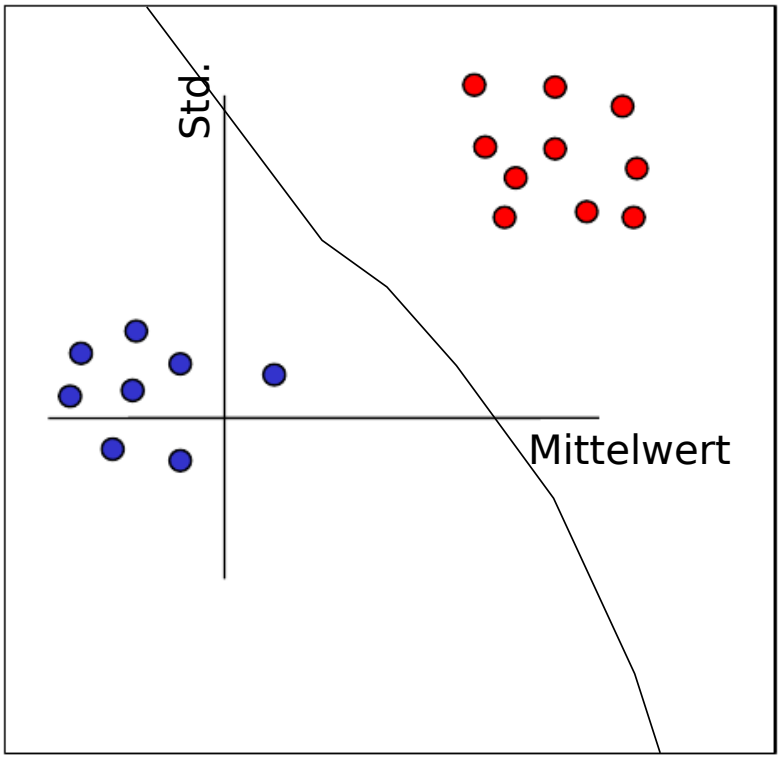
- Nearest Neighbour Klassifikator
- *k-Nearest Neighbour Klassifikator*

Klassifikator



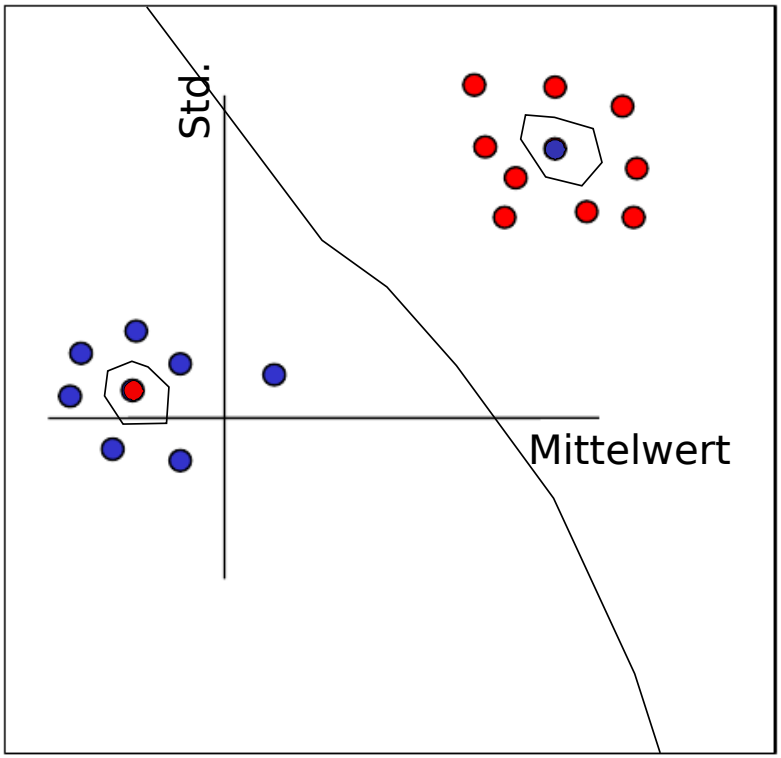
- es werden n Merkmale berechnet
- die Bilder liegen in einem n-dimensionalen Raum (hier: 2D)
- Wie werden die beiden Klassen gut voneinander getrennt?
- wir brauchen eine Grenze zwischen den Klassen

Klassifikator - Nearest Neighbour



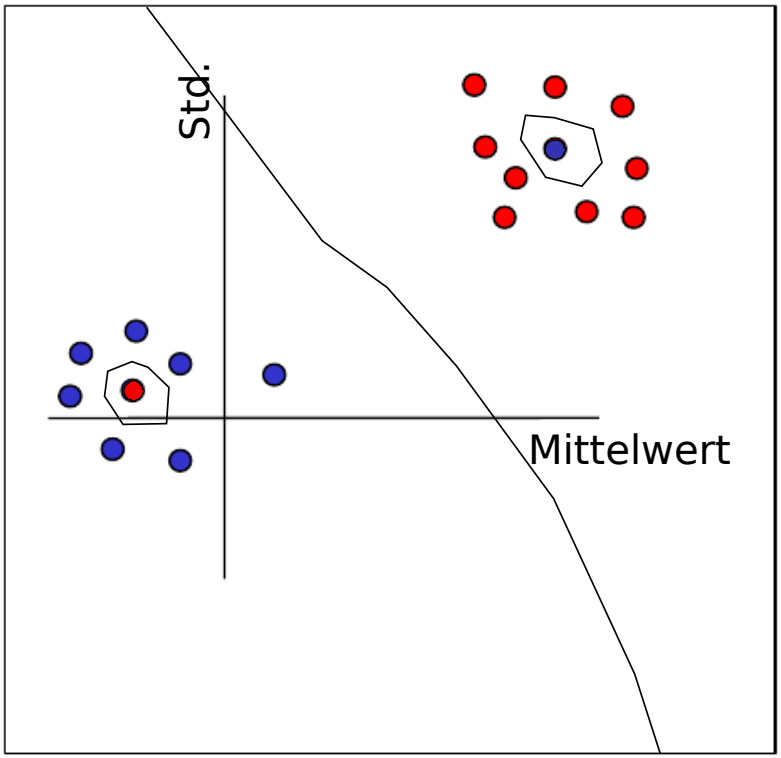
- wir kennen schon den 1NN-Klassifikator
- hier wird implizit eine Entscheidungsgrenze (Decision Boundary) erstellt
- Nachteile: Speicher, Geschwindigkeit, Probleme mit Ausreißern

Klassifikator - Nearest Neighbour



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

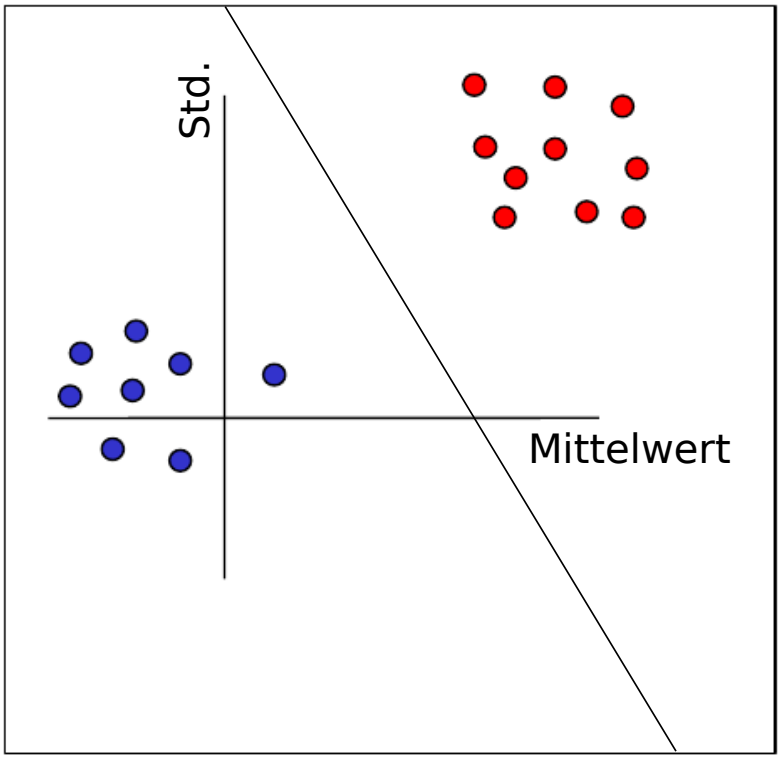
Klassifikator - Nearest Neighbour



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

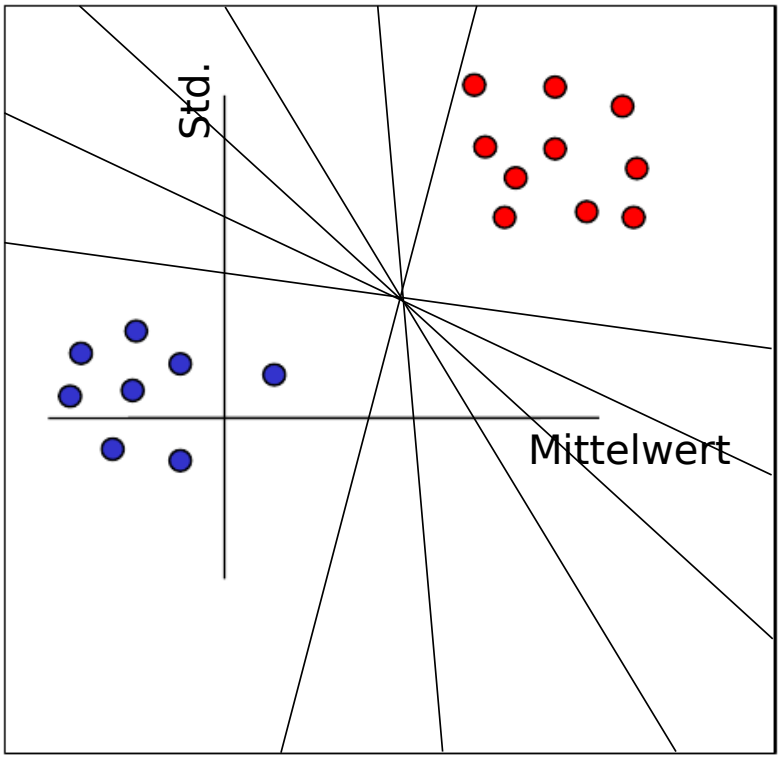
Wir brauchen etwas Robusteres!

Klassifikator - Parametrisches Modell



- Funktion für die Entscheidungsgrenze finden
- hier linear: $y = f(\vec{x}) = \vec{w}^T\vec{x} + b = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $\vec{x}$ ist der Deskriptor oder das Bild
- $\vec{w}$ ist ein Vektor von Gewichten (w_1 und w_2)
- b ist ein Gewicht als Skalar
- y gibt mit dem Vorzeichen die Klassenzugehörigkeit an
- die eigentliche Grenze liegt bei $y = 0$

Klassifikator - Parametrisches Modell



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

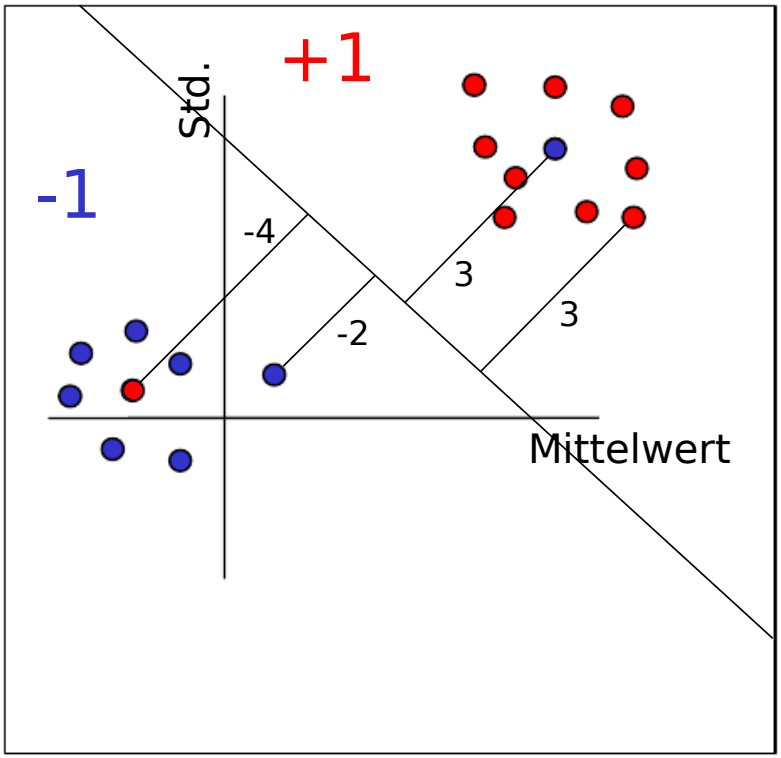
- Funktion für die Entscheidungsgrenze finden
- hier linear: $y = f(\vec{x}) = \vec{w}^T\vec{x} + b = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $\vec{x}$ ist der Deskriptor oder das Bild
- $\vec{w}$ ist ein Vektor von Gewichten (w_1 und w_2)
- b ist ein Gewicht als Skalar
- y gibt mit dem Vorzeichen die Klassenzugehörigkeit an
- die eigentliche Grenze liegt bei $y = 0$

Klassifikator - Entscheidungsgrenze bewerten

Wir brauchen ein mathematisches Kriterium, das die Güte der Entscheidungsgrenze ermittelt $\rightarrow$ 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 - I

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 minimiert die Loss-Funktion L.

Prinzip

- nach jedem Trainings-Bild/Deskriptor können wir das Ergebnis prüfen
- ist das Ergebnis der Klassifikation falsch, erhalten wir einen Loss > 0
- wir können dann $\vec{w}$ und b anpassen $\rightarrow$ Aber wie?

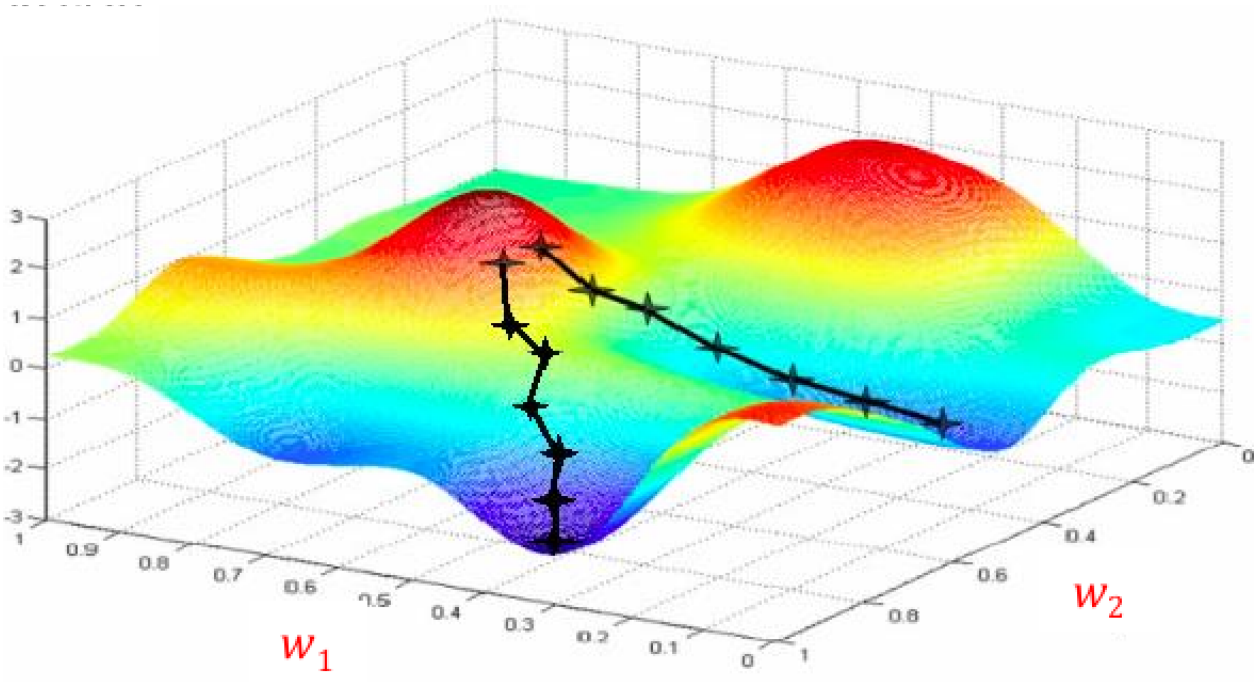
Entscheidungsgrenze optimieren - II

Ein Durchgehen aller möglichen Parameterkombinationen ist schnell nicht mehr möglich.

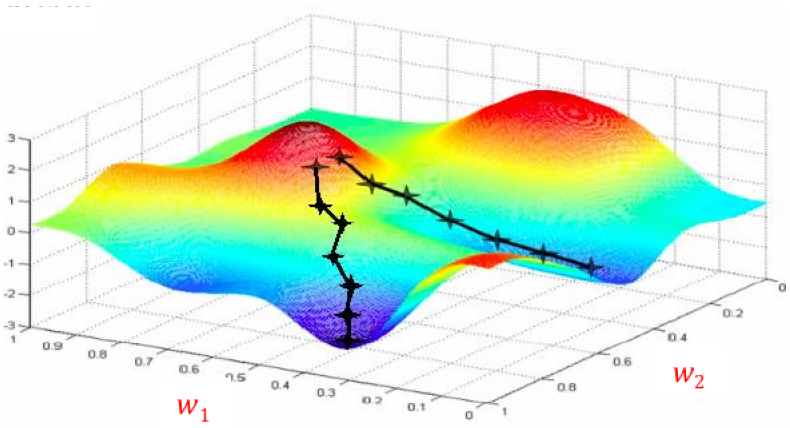
Optimierungsproblem

1. mit beliebigen/zufälligen Werten für $\vec{w}$ und b starten
2. ein Trainings-Bild/Deskriptor klassifizieren
3. war die Klassifikation richtig, zurück zu 1
4. Loss l des Bildes/Deskriptors berechnen über Loss-Funktion L
5. partielle Ableitungen der Loss-Funktion bezüglich der Parameter bilden $\rightarrow$ Gradient
6. $\vec{w}$ und b mit Hilfe des Gradienten aktualisieren
7. zurück zu 1

Gradietenabstieg - I



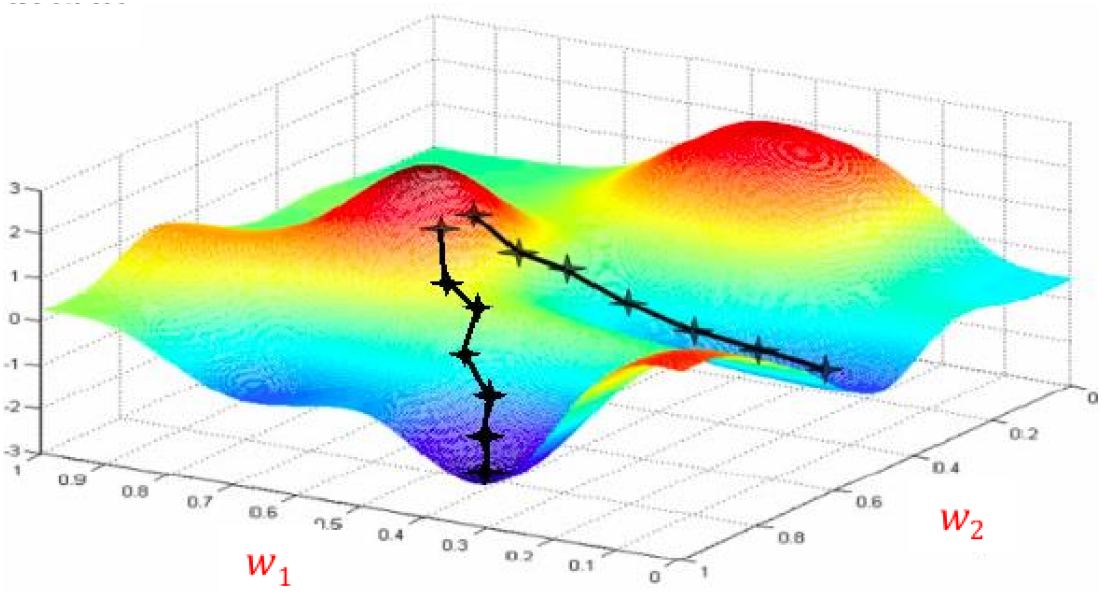
Gradietenabstieg - II



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

Gradietenabstieg - III

- 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 optimal



Gradietenabstieg - Beispiel

Gegeben:

- Entscheidungsgrenze der Form
  $y = f(\vec{x}) = \vec{w}^T \vec{x} + b = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- Gewichte $\vec{w}^{alt}$ bestehend aus w_1^{alt} und w_2^{alt} sowie b^{alt}
- ein Deskriptor $\vec{x}$ bestehend aus x_1 und x_2 mit Label
  $t \in \{-1, 1\}$
- ein Loss-Funktion $(y - t)^2$, falls $\text{sgn}(t) \neq \text{sgn}(y)$

Vorgehen:

- Ist $\text{sgn}(t) \neq \text{sgn}(y)$, also die Klassifikation mit den aktuellen Gewichten falsch? Ja!
- partielle Ableitungen bilden
- Gewichte aktualisieren

Gradietenabstieg - partielle Ableitungen bilden

$$
\begin{aligned}
L(y) &= (y - t)^2 \\
L &= (w_1 \cdot x_1 + w_2 \cdot x_2 + b - t)^2 \\
\frac{\partial L}{\partial w_1} &= 2 \cdot (w_1 \cdot x_1 + w_2 \cdot x_2 + b - t) \cdot x_1 \\
\frac{\partial L}{\partial w_2} &= 2 \cdot (w_1 \cdot x_1 + w_2 \cdot x_2 + b - t) \cdot x_2 \\
\frac{\partial L}{\partial b} &= 2 \cdot (w_1 \cdot x_1 + w_2 \cdot x_2 + b - t)
\end{aligned}
$$

Gradietenabstieg - Gewichte aktualisieren

$$w_1^{neu} = w_1^{alt} - \alpha \frac{\partial L}{\partial w_1}$$
$$w_2^{neu} = w_2^{alt} - \alpha \frac{\partial L}{\partial w_2}$$
$$b^{neu} = b^{alt} - \alpha \frac{\partial L}{\partial b}$$

- die alten Gewichte werden entlang der partiellen Ableitungen (Teile des Gradienten) verschoben
- α ist die learning rate und bestimmt wie stark der Einfluss sein soll

Der Fehler wird so zurückpropagiert zu den Gewichten $\rightarrow$ Backpropagation.

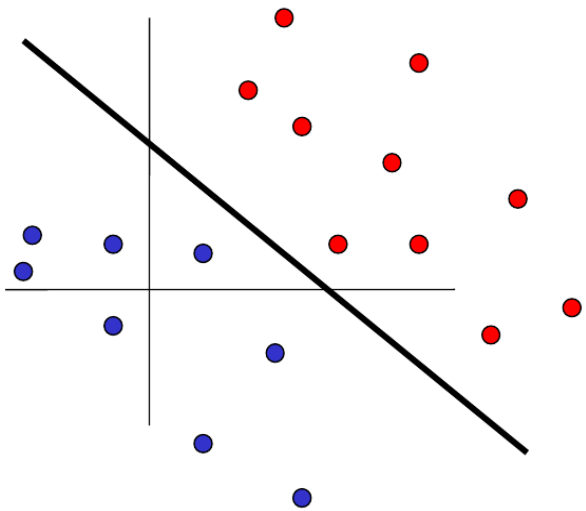
Übersicht

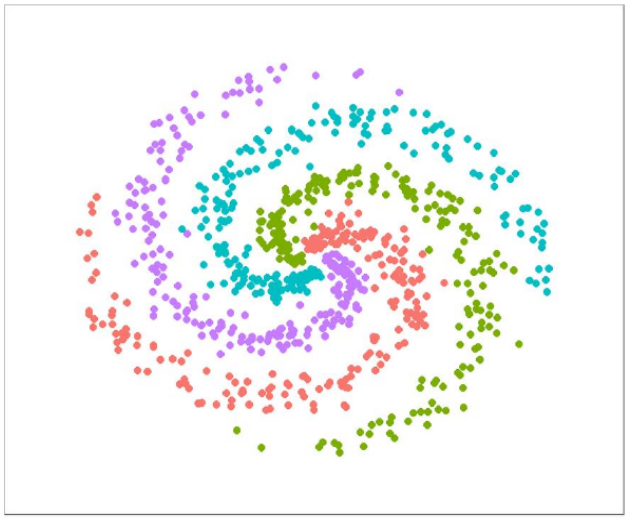
Was nun?

Wie lassen sich komplexere Entscheidungsgrenzen entwickeln?

Lösungen

- komplexere Entscheidungsfunktionen wählen
- einfach Entscheidungsfunktionen (hier Neuronen) kombinieren (+ plus ein paar Tricks)





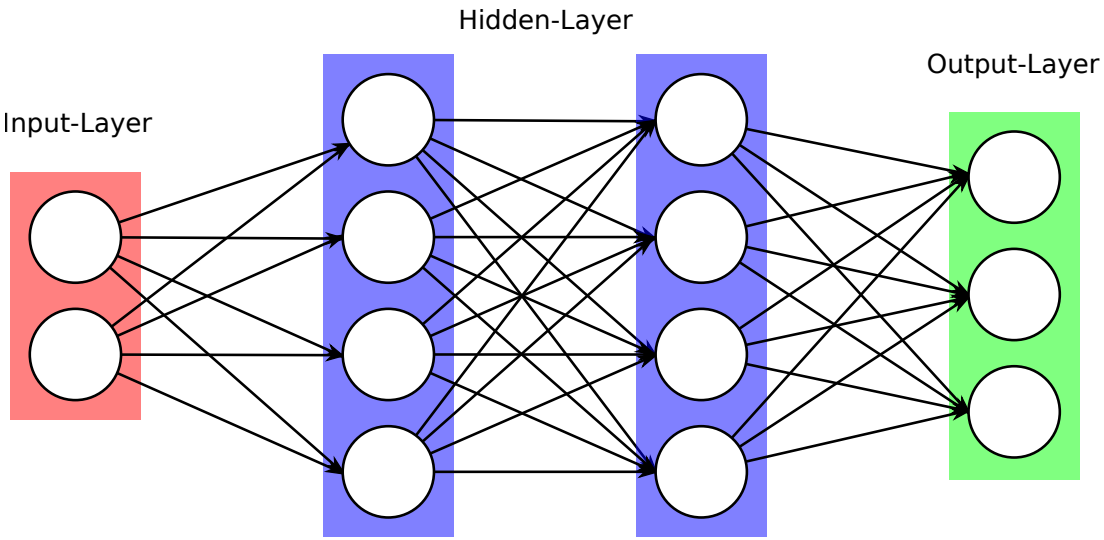
Neuronale Netze

Basis-Prinzip

- abgeleitet von der Funktionsweise eines Gehirn
- Neuronen (Perzeptron) bekommen viele Eingaben, die gewichtet und addiert werden
- das Ergebnis dieser Gewichtung ist die Ausgabe
- die Neuronen sind in Netzwerken mit verschiedenen Schichten angeordnet
- Entscheidungsgrenze wird durch die Gewichte an allen Neuronen bestimmt $\rightarrow$ 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



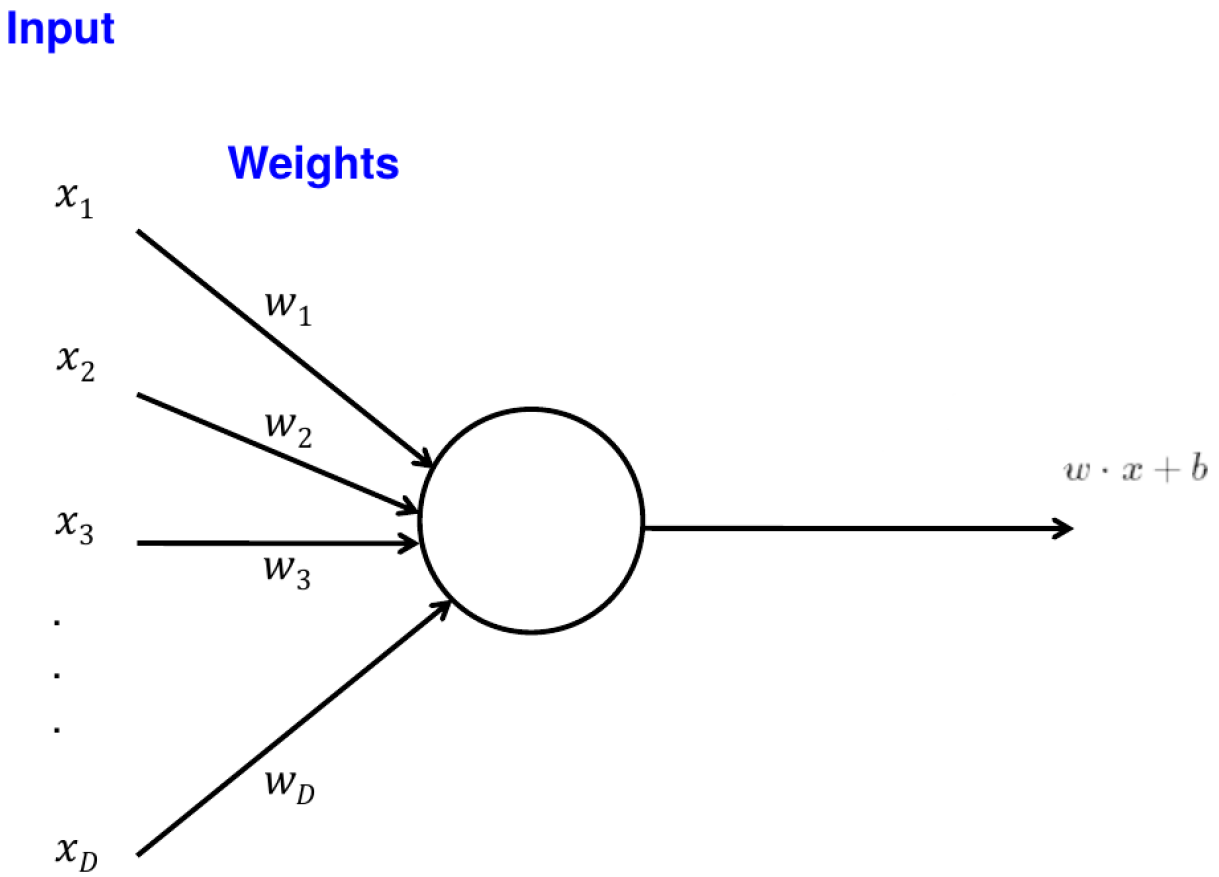
fully connected layer - jeder mit jedem verbunden

Input repräsentieren die Merkmale als Deskriptor oder die Pixel

Hidden kombinieren die Eingaben zu Merkmalskombinationen

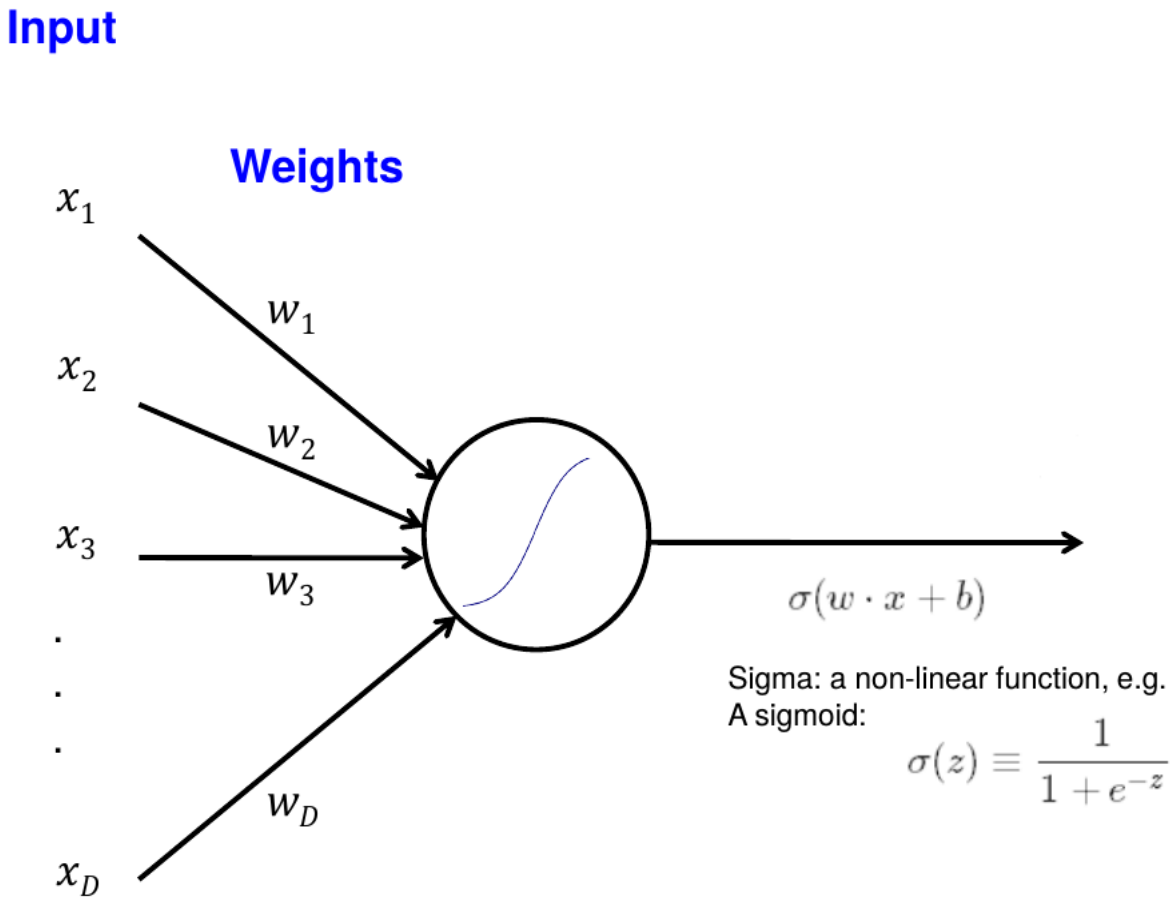
Output fassen die obersten Merkmale zu einem Ergebnis pro Klasse zusammen

Ein Neuron - linear



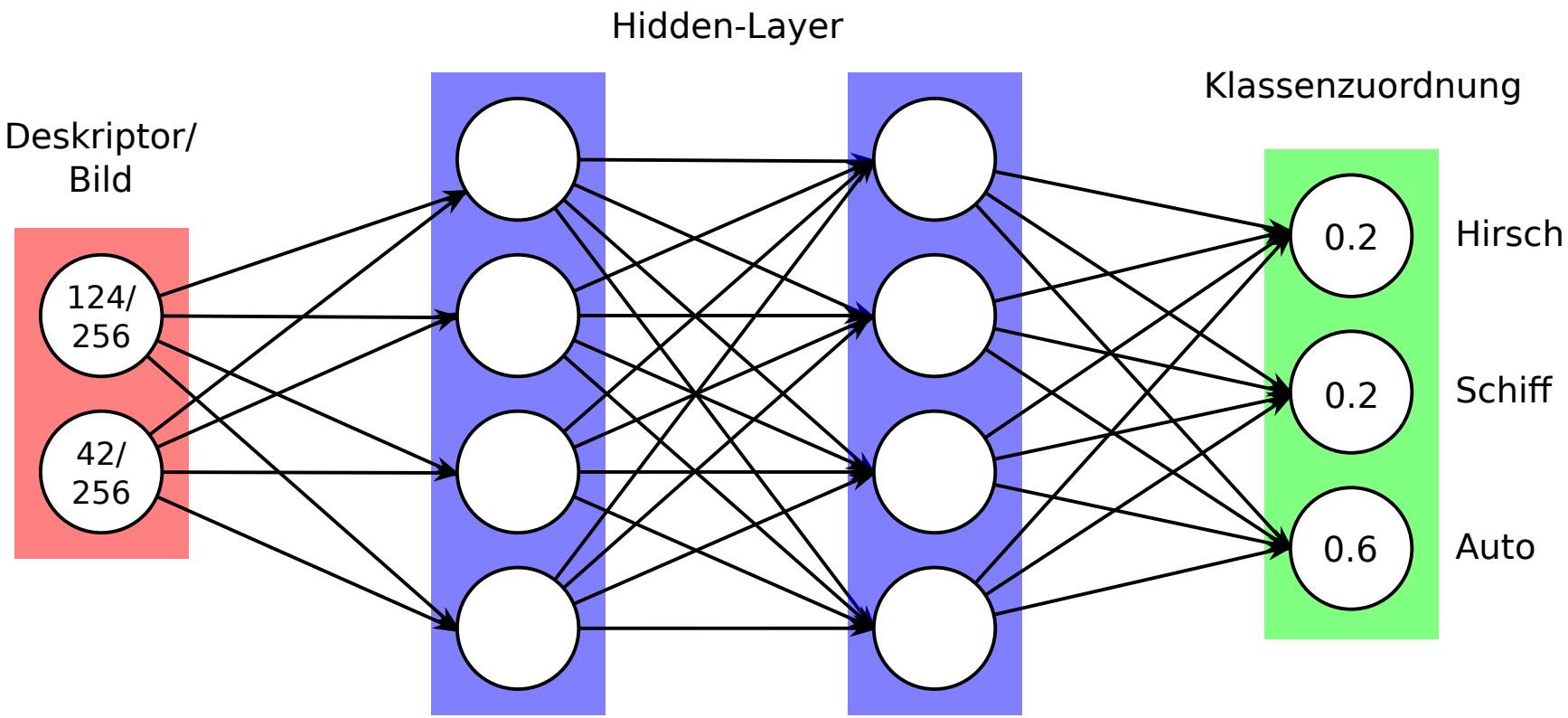
Noch immer ist die Ausgabe eine lineare Kombination der Eingabe
$\rightarrow$ Entscheidungsgrenze linear!

Ein Neuron - nicht-linear



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

Das Training verläuft im Prinzip wie vorher im einfachen Fall gezeigt.

Initialisierung der Parameter mit zufälligen Werten.

1. Teil der Trainingsmenge auswählen (Batch)
2. Batch durch das Netzwerk schicken
3. Loss l über das gesamte Batch berechnen
4. partielle Abl. der Loss-Funktion L bestimmen
5. alle Gewichte entspr. der partiellen Abl. updaten $\rightarrow$ ganz viel Kettenregel

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

Übersicht

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 $\rightarrow$ nächste Woche

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

- Deskriptoren erzeugen (Array mit Shape: Anz. Bilder×Anz. Merkmale) dazu ein 1D-Array mit den Labeln
- 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)

Ist der Input je Bild (bspw. der Deskriptor) nicht schon ein 1D-Array, muss dieser zunächst abgerollt werden.

FC-Layer (Dense)

- 128 Neuronen befinden sich in diesem Layer
- als Aktivierungsfunktion wurde die ReLU-Funktion gesetzt
- als Name wurde `fc1` gewählt

Dem ersten Layer muss stets die `input_shape` der Daten gegeben werden! Dies ist hier die Shape des Deskriptors pro Bild bspw. (2) für MW und STD.

Layer in Keras

Flatten-Layer (Abrollen)

`Flatten()` Ist der Input je Bild (bspw. der Deskriptor) nicht schon ein 1D-Array, muss dieser zunächst abgerollt werden.

FC-Layer (Dense)

- 128 Neuronen befinden sich in diesem Layer
- als Aktivierungsfunktion wurde die ReLU-Funktion gesetzt
- als Name wurde `fc1` gewählt

Dem ersten Layer muss stets die `input_shape` der Daten gegeben werden! Dies ist hier die Shape des Deskriptors pro Bild bspw. (2) für MW und STD.

Layer in Keras

Flatten-Layer (Abrollen)

Ist der Input je Bild (bspw. der Deskriptor) nicht schon ein 1D-Array, muss dieser zunächst abgerollt werden.

FC-Layer (Dense)

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

- 128 Neuronen befinden sich in diesem Layer
- als Aktivierungsfunktion wurde die ReLU-Funktion gesetzt
- als Name wurde `fc1` gewählt

Dem ersten Layer muss stets die `input_shape` der Daten gegeben werden! Dies ist hier die Shape des Deskriptors pro Bild bspw. (2) für MW und STD.

Model-Objekt in Keras

```
model = Sequential()
model.add(Dense(128, activation='relu', name='fc1',
    input_shape=(2,)))
model.add(Dense(128, activation='relu', name='fc2'))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer=SGD(lr=0.000005,
              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

Training mit dem `Model`-Objekt

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

`X_train` Deskriptoren als Array bspw. mit Shape (Anz. Bilder $\times$ Anz. Merkmale)

`Y_train` Label hier als Array mit Shape (Anz. Bilder $\times$ 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)

`verbose` Konsolenausgabe einschalten

Labels umformatieren

```
Y_train = np_utils.to_categorical(trLabels, 3)
```

$$\begin{pmatrix} 2 \\ 0 \\ 1 \\ 2 \\ 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

`trLabels` `Y_train`

- Die Klassen von `trLabels` müssen fortlaufend von 0 sein.
- In jeder Zeile von `Y_train` steht genau eine 1.
- Die Zeilen von `Y_train` entsprechen je einem Trainingsbild.
- Die Spalten von `Y_train` entsprechen je einer Klasse.

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

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

Übersicht

Fachtermini: Deutsch - Englisch

Nächster Nachbar Klassifikator - nearest neighbour classifier

Entscheidungsgrenze - decision boundary

Gradientenabstieg - gradient descent

partielle Ableitungen - partial derivative

(Künstliches) Neuronales Netz - (artificial) neural network oder multi-layer perceptron

Neuron - neuron oder perceptron

Klassifikation allgemein

Lineare Klassifikation

Gradientenabstieg

Neuronale Netze

Training Neuronaler Netze/Backpropagation

Keras