

Vorlesung 2: Learning Algorithms und Wahrscheinlichkeitstheorie

BA-INF 153: Einführung in Deep Learning für Visual Computing

Prof. Dr. Reinhard Klein

Nils Wandel

Informatik II, Universität Bonn

17.04.2024

Recap

Visual Computing \approx Computer Vision + Computer Grafik

Computer Vision

- Computer Vision ist ein Bereich der künstlichen Intelligenz, in dem interpretierbare Informationen aus Bildern und Videos extrahiert wird. Beispiele sind: image classification, object detection, action recognition, image captioning, semantic segmentation and scene understanding
 - Probleme in Computer vision sind: Deformationen, Verdeckungen, Beleuchtung, Background-Clutter and Intraclass Varianz

Computer Grafik

- Computer Grafik ist ein Bereich der Informatik, in dem aus Szenenbeschreibungen Bilder oder Videos generiert werden sollen. Dabei können die Szenenbeschreibungen konkret (z.B. Mesh-Geometrie und Textur) oder auch abstrakt (z.B. latenter Vektor oder Textbeschreibung) gehalten sein.
 - Probleme in Computer Grafik sind: hoher Rechenaufwand (z.B. für Rendering-Gleichung), schlecht gestellte inverse Probleme (z.B. Superresolution)

Recap

Deep Learning \approx Machine Learning + Deep Neural Networks

Machine Learning

Anstatt dem Computer einen detaillierten Algorithmus vorzugeben (z.B. mit manuell erstellten Features), soll der Computer selbst lernen, Aufgaben zu lösen (z.B. an Hand von Beispieldaten).

Neuronale Netze

- sind von biologischen neuronalen Netzen inspiriert
 - entsprechen parametrisierten Abbildungen
 - haben viele Anwendungen in der Computer Vision und Computer Grafik

Today's Lecture

- ① Machine Learning Algorithmen
- ② Recap Wahrscheinlichkeitstheorie
- ③ Statistische Schätzer

Part 1

Machine Learning Grundlagen

Chapter 5.1 - 5.3, Goodfellow et al (2016): Deep Learning [Link]

Goodfellow-et-al-2016-Book

Learning Algorithms and Tasks

A Learning Algorithm

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ."

"Man sagt, ein Computer Programm lernt von Erfahrung E bezüglich einer Klasse von Aufgaben T und Performanz-Maß P , wenn sich die Performanz auf den Aufgaben in T , gemessen durch P , mit der Erfahrung E verbessert."

(Mitchell, 1997¹)

¹Mitchell, T. M. (1997). Machine Learning. McGraw-Hill, New York.

Task T

Machine Learning Aufgaben T

Aufgaben beschreiben, wie ein Machine Learning System (z.B. ein neuronales Netz) Daten verarbeiten soll. Solche Daten könnten z.B. Pixel / Voxel / Features etc sein, die durch Sampling gemessen wurden. Typische Aufgaben sind z.B.:

- **Klassifikation:** Bestimme die Kategorie einer Eingabe, z.B.
 $f : \mathbb{R}^n \rightarrow \{1, \dots, k\}$
 - **Regression:** Sage numerische Werte für eine Eingabe vorraus, z.B.
 $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$
 - **Dichteschätzung:** Lerne Wahrscheinlichkeitsdichtefunktion $p_{\text{model}} : \mathbb{R}^n \rightarrow \mathbb{R}$
 - **Synthese:** Generiere neue Samples, die ähnlich zu Trainingsdaten sind
 - **Entrauschen:** Erstelle *entrausches* Sample $\mathbf{x} \in \mathbb{R}^n$ von *korrumptiertem* Sample $\tilde{\mathbf{x}} \in \mathbb{R}^n$
 - **Lochfüllung:** Erstelle eine Füllung für ein Sample $\mathbf{x} \in \mathbb{R}^n$ mit fehlenden Einträgen x_i

siehe auch Einsatzbeispiele von letzter Woche.

Erfahrung E

Die Erfahrung E spezifiziert die Informationen, die ein Algorithmus während des Lernprozesses verwenden kann. Üblicherweise in Form eines *Datensets (dataset)* – einer Sammlung von *Datenpunkten (data-points)*²

Im **supervised** learning besitzt jeder Datenpunkt x des Datensets ein zugehöriges *Label* bzw *Ziel (target)* y . Nach dem Training soll der Algorithmus dann zu den Daten x die passenden targets y bestmöglich vorherzusagen. Im probabilistischen Sinne entspricht dies einer Schätzung von $p_{\text{data}}(y|x)$.

Im *unsupervised* learning gibt es solche Labels nicht. Stattdessen werden Trainingsdaten x zufällig³ gezogen und das Ziel ist es, die zugrunde liegende Struktur der Daten, d.h. die Wahrscheinlichkeitsverteilung, welche die Daten generiert (*data generating distribution*) $p_{\text{data}}(x)$ zu lernen.

²Datenpunkte werden synonym verwendet mit Feature-Messungen der Daten.

³Hier bezieht sich "zufällig" darauf, dass die Trainingsdaten als Zufallsvariable modelliert werden!

Performanz-Maß P

Lernalgorithmen können quantitativ mit einem *aufgabenspezifischen* Performanz-Maß evaluiert werden. Zum Beispiel sind wir in Klassifizierungsproblemen an der *Akkuratheit (accuracy)* interessiert. Bei Wahrscheinlichkeitsdichtefunktionen macht es hingegen mehr Sinn, eine mittlere logarithmische Wahrscheinlichkeit von Samples anzugeben.

Wir möchten die Performanz unseres Systems auf Daten evaluieren, die *noch nicht betrachtet wurden*. Dadurch können wir abschätzen, wie gut das System auf neuen Daten funktioniert, im Vergleich zu solchen Daten, welche bereits während des Lernprozesses betrachtet wurden.

Die ungesesehenen Daten nennen wir auch *test set* oder Test Daten, während wir die zum Training verwendeten Daten als *training set* oder Training Daten bezeichnen.

Trainings-Fehler, Test-Fehler und Generalisierung

Generalisierung

Wir sind an Lernalgorithmen interessiert, welche gut *generalisieren*, d.h. auch auf noch nicht gesehenen Daten des Testdatensets eine gute Performanz zeigen. Während des Trainings möchten wir den Trainings-Fehler reduzieren – dies ist ein klassisches Optimierungsproblem. Was *Lernen* jedoch von Optimieren unterscheidet ist das Interesse daran, auch den Test-Fehler zu reduzieren.

Under- and Over-fitting

Die Performance eines Machine learning Algorithmus hängt von 2 Faktoren ab:

- ① Kleiner Trainings-Fehler
 - ② Kleiner Performance-Unterschied zwischen Trainings und Test-Fehlern (Generalization gap)

Algorithmen, welche *underfitten*, können das erste Kriterium nicht erfüllen; Algorithmen, welche *overfitten*, können das erste, jedoch nicht das zweite Kriterium erfüllen.

Modellkapazität

Under- und Over-fitting können durch die **Modellkapazität** des Lernalgorithmus beeinflusst werden. Modelle mit (zu) geringer Kapazität sind nicht aussagekräftig genug um die Trainingsdaten zu beschreiben und underfitten deshalb. Modelle mit (zu) hoher Kapazität lernen die Trainingsdaten einfach auswendig und overfitten deshalb.

Die Modellkapazität kann durch die Wahl des *Hypothesenraums* (*hypothesis space*) kontrolliert werden. Dieser Raum gibt die Menge der Lösungen vor, aus denen der Lernalgorithmus wählen kann.

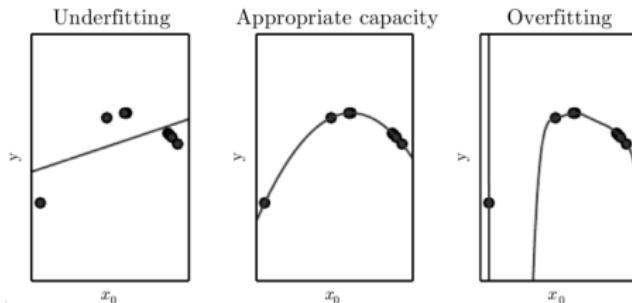


Figure: 5.2 von Goodfellow. In diesem Regressionsbeispiel wird die Modellkapazität durch den Grad des Polynoms vorgegeben.

Kapazität vs. Fehler

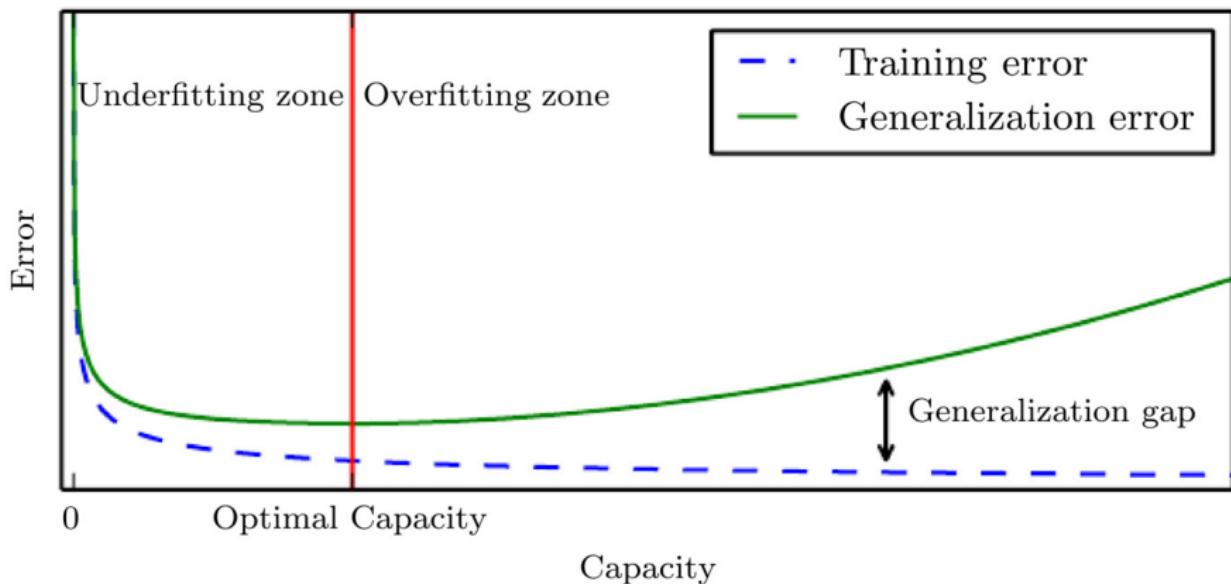


Figure: 5.3 von Goodfellow. Trainings- und Testfehler als Funktion der Modellkapazität.

Hyperparameter

Die meisten Lernalgorithmen besitzen parametrisierte Einstellungen. Diese **Hyperparameter** (z.B. Modellgröße, Learning Rate, etc.) kontrollieren das Gesamtverhalten des Lernalgorithmus.

Hyperparameter werden nicht während des Trainings verändert sondern entweder zu Beginn vorgegeben oder durch eine weitere äußere Lernprozedur (z.B. Gridsearch) bestimmt. Im vorherigen Regressionsbeispiel versucht der Lernalgorithmus die Polynomkoeffizienten zu finden, während der Grad des Polynoms ein Hyperparameter für die Modellkapazität ist.

Frage: Können wir den Kapazitäts-hyperparameter allein aus dem Trainingsset ableiten? Warum (nicht)?

Hyperparameter

Die meisten Lernalgorithmen besitzen parametrisierte Einstellungen. Diese **Hyperparameter** (z.B. Modellgröße, Learning Rate, etc.) kontrollieren das Gesamtverhalten des Lernalgorithmus.

Hyperparameter werden nicht während des Trainings verändert sondern entweder zu Beginn vorgegeben oder durch eine weitere äußere Lernprozedur (z.B. Gridsearch) bestimmt. Im vorherigen Regressionsbeispiel versucht der Lernalgorithmus die Polynomkoeffizienten zu finden, während der Grad des Polynoms ein Hyperparameter für die Modellkapazität ist.

Frage: Können ein Lernalgorithmus den Kapazitäts-hyperparameter allein aus dem Trainingsset ableiten? Warum (nicht)?

- Nein, sonst würde der Lernalgorithmus immer die maximale Kapazität wählen um das Traingsset optimal zu fitten. Dies würde zu Overfitting führen.

Data-driven Classification

Die Slides in diesem Abschnitt zur Bildklassifizierung sind aus den ersten 3 Vorlesungen des Stanford CS231n Kurses von Fei-Fei Li entnommen [Link].

Datenbasierter Ansatz:

- ① Erstelle ein Dataset mit Bildern und Labels
 - ② Trainiere einen Bildklassifikator mit Machine Learning
 - ③ Evaluiere den Klassifikator auf einem zurückgehaltenen Set von Testbildern

```
def train(train_images, train_labels):
    # build a model for images -> labels...
    return model

def predict(model, test_images):
    # predict test_labels using the model...
    return test_labels
```

Example training set



Nearest Neighbour Classification

First classifier: **Nearest Neighbor Classifier**

```
def train(train_images, train_labels):
    # build a model for images -> labels...
    return model

def predict(model, test_images):
    # predict test_labels using the model...
    return test_labels
```

Remember all training images and their labels

Predict the label of the most similar training image

Nearest Neighbour Classification - Hyperparameter

Example dataset: **CIFAR-10**

10 labels

50,000 training images

10,000 test images.

The image displays a 10x40 grid of small square images, each representing a different object from the CIFAR-10 dataset. The objects are categorized by row: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each category contains 40 images, showing variations in color, orientation, and perspective. The images are arranged in a single horizontal row.

For every test image (first column),
examples of nearest neighbors in rows



Nearest Neighbour Classification

How do we compare the images? What is the **distance metric**?

L1 distance:

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

test image			
56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

1

training image			
10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

add
→ 456

Nearest Neighbour Classification

Die Wahl der Distanz ist ein *Hyperparameter*.

Typische Distanzen sind:

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

Die Wahl von k ist ebenfalls ein *Hyperparameter*.

Wie bestimmt man solche Hyperparameter?

Was ist die am besten geeignete *Distanz*?

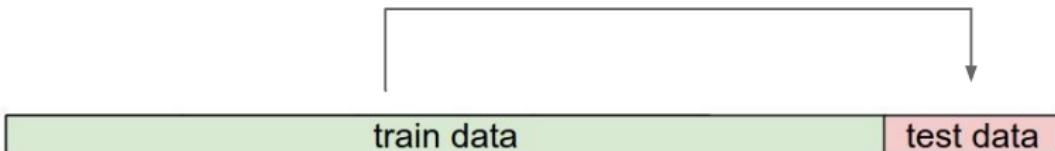
Was ist der am besten geeignete Wert für k ?

D.h. wie sollen wir diese *Hyperparameter* bestimmen?

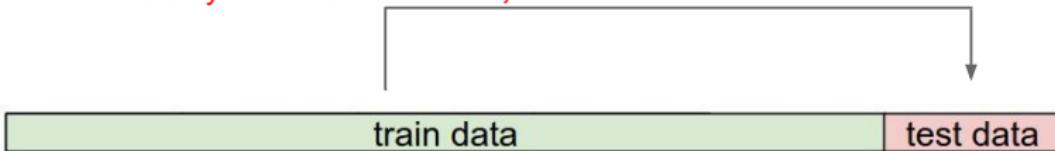
⇒ Dies ist sehr problemabhängig und wird üblicherweise durch trial and error gelöst.

Bestimme Hyperparameter nicht mit Hilfe der Testdaten!

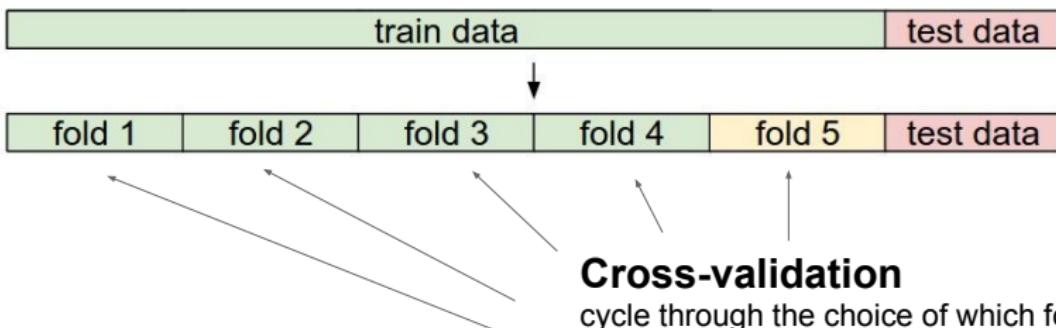
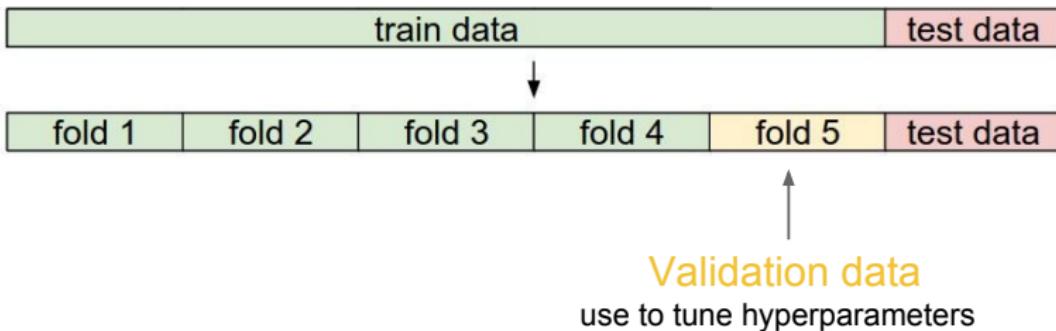
Try out what hyperparameters work best on test set.



Very bad idea. The test set is a proxy for the generalization performance!
Use only **VERY SPARINGLY**, at the end.



Bestimme Hyperparameter auf Validierungs-Daten



Part 2

Recap Wahrscheinlichkeitstheorie

Die nachfolgenden Slides sind den beigelegten Slides von Kapitel 2 aus dem Buch "Computer vision: models, learning, and inference" von Simon JD Prince entnommen. prince2012computer

Motivation

Warum müssen wir uns mit Wahrscheinlichkeitstheorie beschäftigen?

Learning Task T besteht oft darin, eine (bedingte) Wahrscheinlichkeitsfunktion zu lernen.

Beispiele:

- Gegeben eines Eingabetextes - wie hoch ist die Wahrscheinlichkeit für das nächste Wort?
 - Gegeben ein Bild, wie sicher können wir uns sein, dass es ein Auto enthält?
 - Generiere mir ein Bild aus der Wahrscheinlichkeitsfunktion der Bilder, die ein Porträt-Foto enthalten.

Zu diesem Zweck wird das Performanz Mass P meistens aus der Wahrscheinlichkeitstheorie hergeleitet. Außerdem können wir somit over- und underfitting theoretisch besser analysieren.

Recap Stochastik

In der Stochastik-Vorlesung (wird hier nicht vorausgesetzt) hatten wir u.a. folgende Begriffe kennengelernt:

- Ergebnisraum: Ω (Bsp.: Würfel: $\Omega = \{\square, \bullet\square, \circ\square, \square\square, \bullet\square\square, \circ\square\square\}$)
 - Ereignis: $A \subset \Omega$ (Bsp.: Gerade Augenzahl: $A = \{\square, \square\square, \bullet\square\square\}$)
 - Sigma-Algebra: Ereignis-System $\mathcal{A} \subset 2^\Omega$, welches:
 - ... das sichere Ereignis enthält: $\Omega \in \mathcal{A}$
 - ... abgeschlossen unter Komplementbildung ist: $A \in \mathcal{A} \Rightarrow A^c := \Omega \setminus A \in \mathcal{A}$
 - ... abgeschlossen unter Vereinigungsbildung ist:

$$A_1, A_2, \dots \in \mathcal{A} \Rightarrow \bigcup_{i \geq 1} A_i \in \mathcal{A}$$

(Bsp: $\mathcal{A} = \{\emptyset, \Omega\}$ und $\mathcal{A} = 2^\Omega$ sind Sigma-Algebren)
 - Messraum: (Ω, \mathcal{A}) ist ein Messraum, wenn Ω ein Ergebnisraum und \mathcal{A} eine dazugehörige Sigma-Algebra ist.

Recap Stochastik

- Zufallsvariable: Seien (Ω, \mathcal{A}) und (Ω', \mathcal{A}') Messräume. Dann heißt eine Abbildung:

$$X : \Omega \rightarrow \Omega'$$

eine Zufallsvariable, wenn:

$$\forall A' \in \mathcal{A}' : X^{-1}[A'] := \{\omega \in \Omega | X(\omega) \in A'\} \in \mathcal{A}$$

Intuitiv bedeutet dies, dass zu jedem Ereignis $A' \in \mathcal{A}'$ ein passendes Ereignis in $X^{-1}[A'] \in \mathcal{A}$ existiert. Oft entspricht die Zielmenge Ω' einer Zahlenmenge (dann spricht man auch von einer "Zufallszahl"). Somit erlaubt X Aussagen z.B. über einen Erwartungswert oder eine Varianz zu treffen.

(Beispiel: Eine Zufallszahl X könnte zum Beispiel jedem Ausgang eines Würfelexperiments eine Zahl zuordnen, welche der Augenzahl des Würfels entspricht. Hier wäre dann $\Omega = \{\square, \square\cdot, \square\cdot\cdot, \square\cdot\cdot\cdot, \square\cdot\cdot\cdot\cdot, \square\cdot\cdot\cdot\cdot\cdot\}$ und $\Omega' = \{1, 2, 3, 4, 5, 6\}$. Beachte: mit den Würfelergebnissen in Ω selbst könnte man noch nicht rechnen!)

Recap Stochastik

- Wahrscheinlichkeitsfunktion falls Ω diskret ist:
 $p : \Omega \rightarrow [0, 1]$ mit $\sum_{\omega \in \Omega} p(\omega) = 1$
 - Wahrscheinlichkeitsdichtefunktion falls Ω kontinuierlich ist:
 $p : \Omega \rightarrow [0, \infty)$ mit $\int p(\omega) d\omega = 1$
 - Wahrscheinlichkeitsverteilung: $P : \mathcal{A} \rightarrow [0, 1]$:
 - Normierung: $P(\Omega) = 1$
 - σ -Additivitat: Fur paarweise disjunkte Ereignisse $A_1, A_2, \dots \in \mathcal{A}$ gilt:

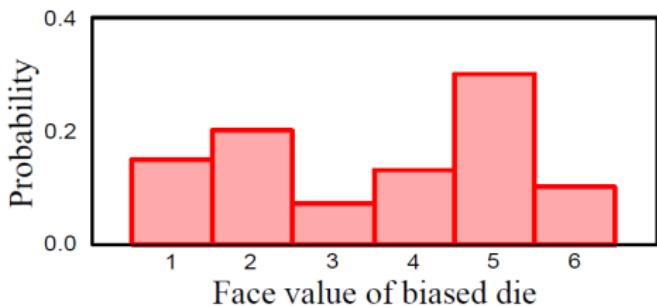
$$P\left(\bigsqcup_{i>1} A_i\right) = \sum_{i>1} P(A_i)$$

(Bsp: Aus einer Wahrscheinlichkeitsfunktion p lässt sich für einen diskreten Ergebnisraum Ω einfach eine Wahrscheinlichkeitsverteilung P konstruieren:
 $P(A) = \sum_{\omega \in A} p(\omega) \forall A \in \mathcal{A}$)

... in dieser Vorlesung liegt der Fokus auf Wahrscheinlichkeits(-dichte-)funktionen

Wahrscheinlichkeitsfunktion

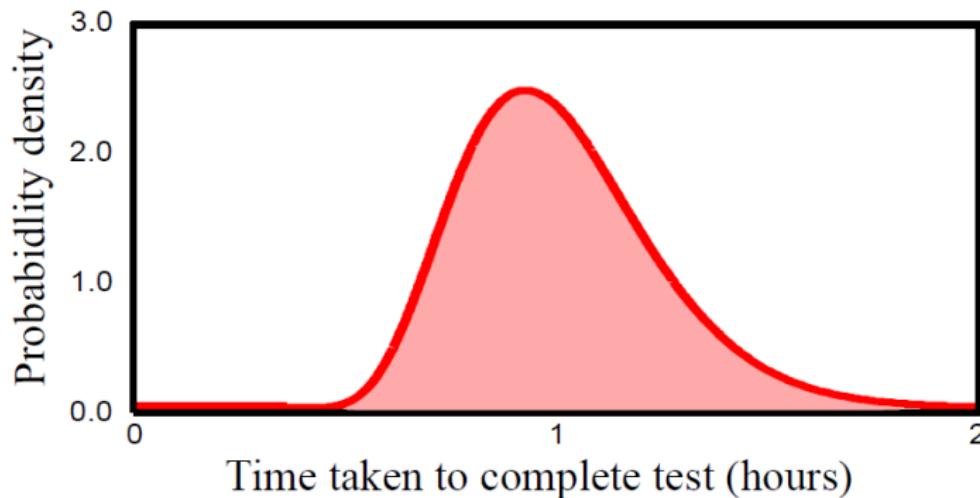
Beispiel: für diskrete Zufallsvariablen können wir eine Wahrscheinlichkeitsfunktion $p : \Omega \rightarrow [0, 1]$ mit $\sum_{\omega \in \Omega} p(\omega) = 1$ verwenden:



Rain Drizzles Cloud Snow Sleet Sun Wind

Wahrscheinlichkeitsdichtefunktion

Beispiel: für kontinuierliche Zufallsvariablen können wir eine Wahrscheinlichkeitsdichtefunktion $p : \Omega \rightarrow [0, \infty)$ mit $\int p(\omega) d\omega = 1$ verwenden:



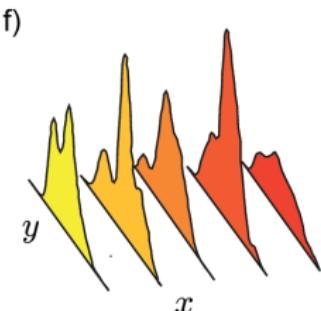
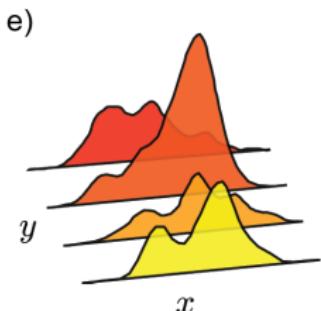
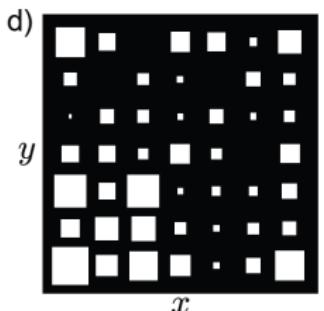
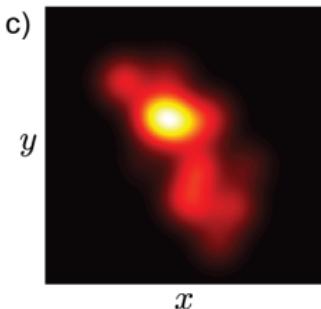
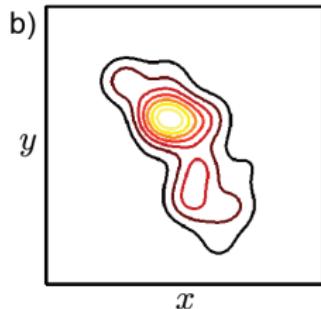
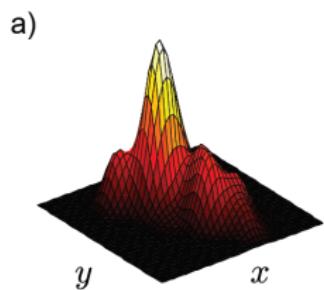
Multivariate Wahrscheinlichkeit

Wenn wir Paarinstanzen von zwei Zufallsvariablen X und Y beobachten, dann können manche Kombinationen häufiger vorkommen als andere. Dies wird durch die multivariate Wahrscheinlichkeitsfunktion erfasst:

- ... geschrieben als $p(x, y)$
 - ... gesprochen als "Wahrscheinlichkeit von x und y "

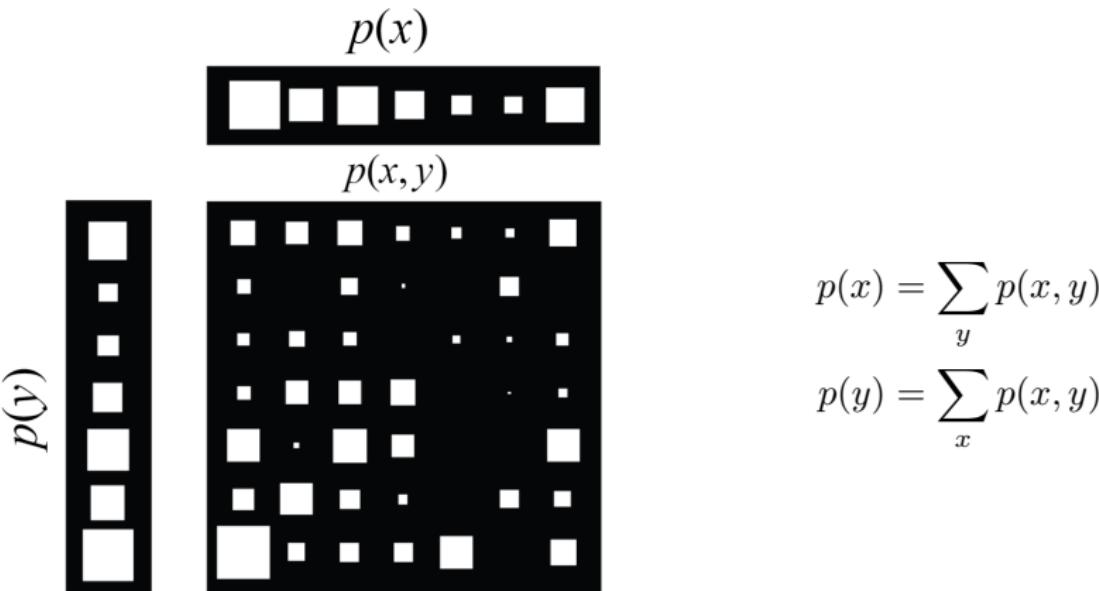
Multivariate Wahrscheinlichkeit

Verschiedene Darstellungsmöglichkeiten für multivariate Wahrscheinlichkeits(-dichte-)funktionen mit kontinuierlichen und diskreten Zufallsvariablen:



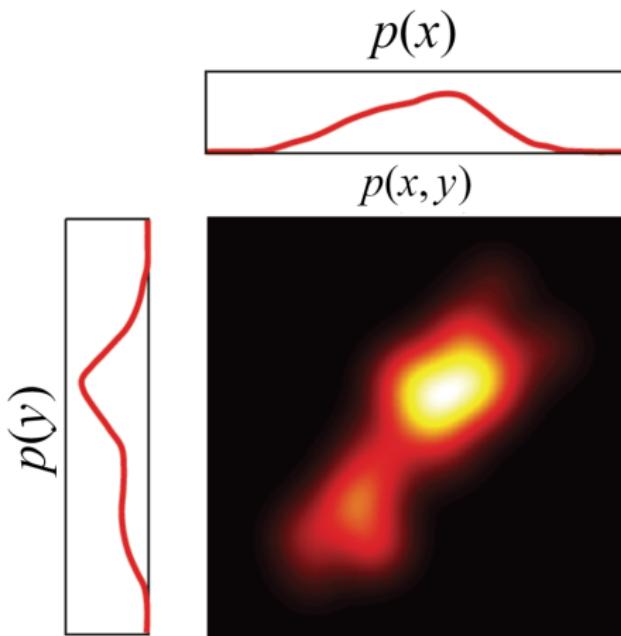
Marginalisierung im diskreten Fall

Wir können die Wahrscheinlichkeitsfunktion für jede Zufallsvariable einer multivariaten Wahrscheinlichkeitsfunktionen erhalten, indem wir über die anderen Variablen integrieren (oder summieren):



Marginalisierung im kontinuierlichen Fall

Wir können die Wahrscheinlichkeitsfunktion für jede Zufallsvariable einer multivariaten Wahrscheinlichkeitsfunktionen erhalten, indem wir über die anderen Variablen integrieren (oder summieren):

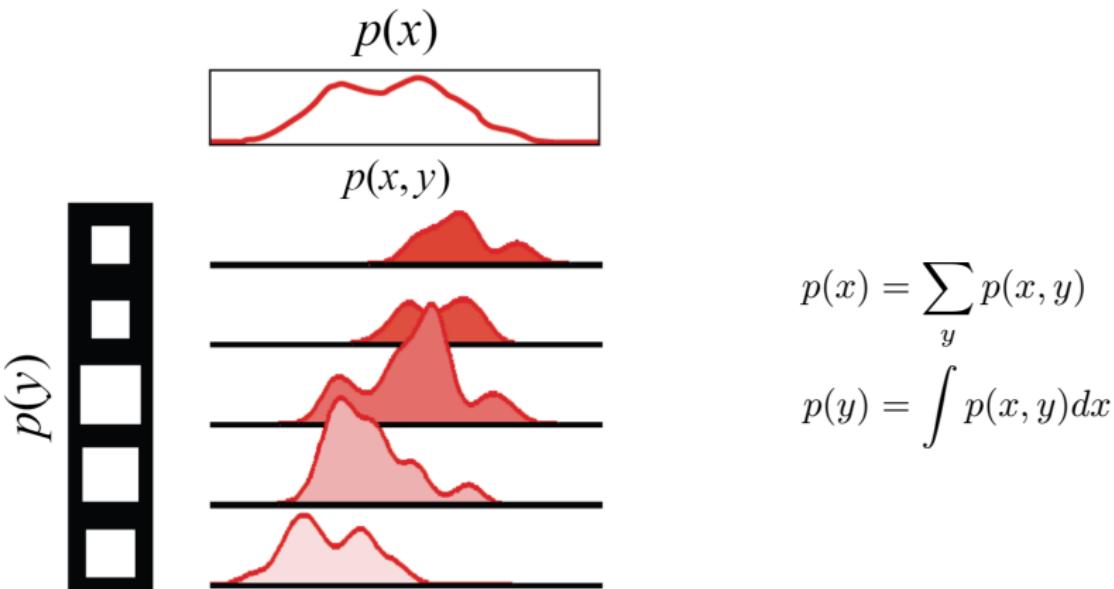


$$p(x) = \int p(x, y) dy$$

$$p(y) = \int p(x, y) dx$$

Marginalisierung im gemischten Fall

Wir können die Wahrscheinlichkeitsfunktion für jede Zufallsvariable einer multivariaten Wahrscheinlichkeitsfunktionen erhalten, indem wir über die anderen Variablen integrieren (oder summieren):



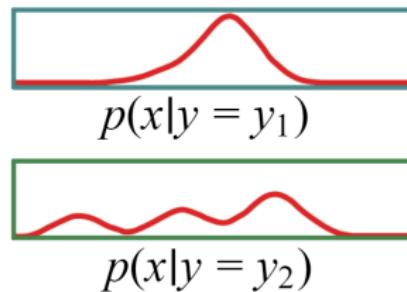
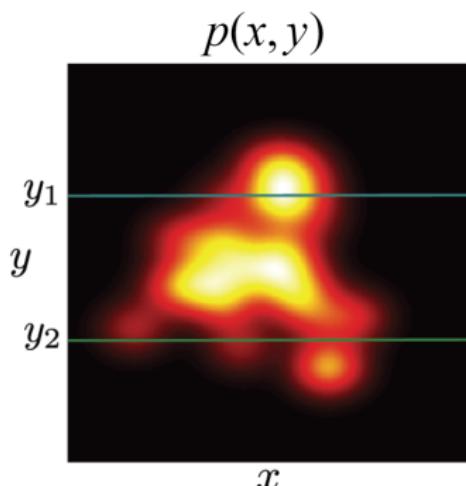
Marginalisierung in mehreren Dimensionen

Marginalisierung funktioniert auch in mehreren Dimensionen und resultiert in einer multivariaten Wahrscheinlichkeitsfunktion der verbleibenden Variablen:

$$p(x, y) = \sum_w \int p(x, y, z, w) dz$$

Bedingte Wahrscheinlichkeit

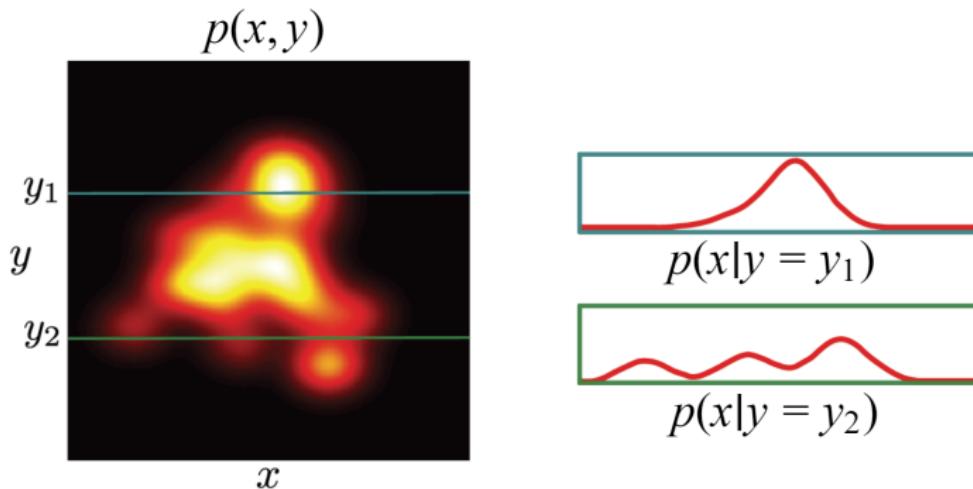
Die bedingte Wahrscheinlichkeit von x gegeben $y = y_1$ ist die relative Wahrscheinlichkeit, dass x verschiedene Werte annimmt unter der Bedingung, dass y auf y_1 festgelegt wurde. Diese bedingte Wahrscheinlichkeit wird auch als $p(x|y = y_1)$ geschrieben.



Bedingte Wahrscheinlichkeit

Die bedingte Wahrscheinlichkeit $p(x|y = y^*)$ kann aus der multivariaten Wahrscheinlichkeitsfunktion $p(x, y)$ berechnet werden, indem der entsprechende Querschnitt aus $p(x, y)$ extrahiert und normiert wird:

$$p(x|y = y^*) = \frac{p(x, y = y^*)}{\int p(x, y = y^*) dx} = \frac{p(x, y = y^*)}{p(y = y^*)}$$



Bedingte Wahrscheinlichkeit

$$p(x|y = y^*) = \frac{p(x, y = y^*)}{\int p(x, y = y^*) dx} = \frac{p(x, y = y^*)}{p(y = y^*)}$$

Dies wird üblicherweise wie folgt kompakt geschrieben:

$$p(x|y) = \frac{p(x,y)}{p(y)}$$

Durch Umformung erhalten wir:

$$p(x, y) = p(x|y)p(y)$$

$$p(x, y) = p(y|x)p(x)$$

Diese Idee kann auch auf mehr als 2 Variablen erweitert werden:

$$\begin{aligned} p(w, x, y, z) &= p(w, x, y|z)p(z) \\ &= p(w, x|y, z)p(y|z)p(z) \\ &= p(w|x, y, z)p(x|y, z)p(y|z)p(z) \end{aligned}$$

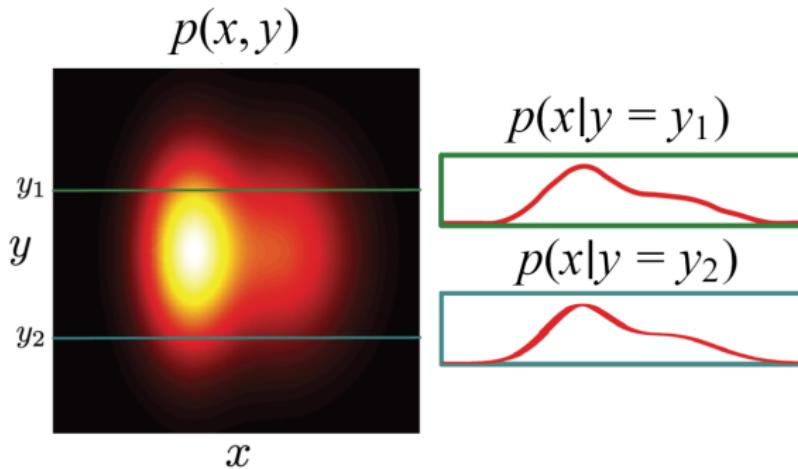
Unabhängigkeit

Wenn 2 Variablen x und y unabhängig sind, dann sagt x nichts über y aus (und anders herum):

$$p(x|y) = p(x) \text{ und } p(y|x) = p(y) \Rightarrow p(x,y) = p(x|y)p(y) = p(x)p(y)$$

... $p(x, y)$ zerfällt also in das Produkt der marginalisierten Verteilungen $p(x)p(y)$

Beispiel mit kontinuierlichen Variablen:



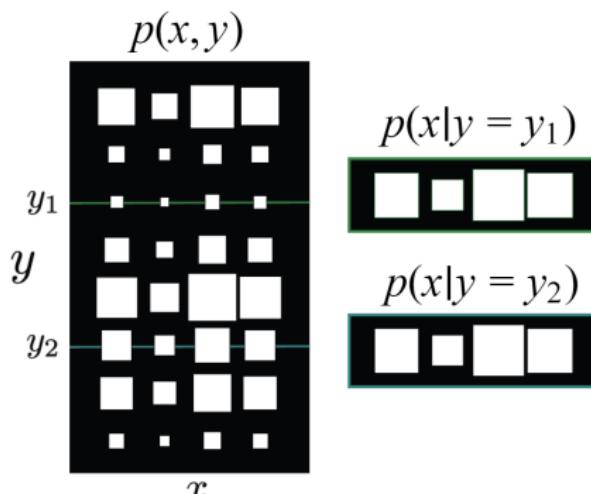
Unabhängigkeit

Wenn 2 Variablen x und y unabhängig sind, dann sagt x nichts über y aus (und anders herum):

$$p(x|y) = p(x) \text{ und } p(y|x) = p(y) \Rightarrow p(x,y) = p(x|y)p(y) = p(x)p(y)$$

... $p(x, y)$ zerfällt also in das Produkt der marginalisierten Verteilungen $p(x)p(y)$

Beispiel mit diskreten Variablen:



i.i.d. (independent and identical distributed)

... im deutschen auch "u.i.v." für "unabhängig und identisch verteilt"

Wenn wir Daten (z.B. x_1, x_2, x_3, \dots) i.i.d. messen, ziehen wir mehrere Samples unabhängig voneinander aus der gleichen Verteilung. Aus der Definition der Unabhängigkeit folgt dann, dass:

$$p(x_1, x_2, x_3, \dots) = p(x_1)p(x_2)p(x_3)\dots = \prod_i p(x_i)$$

Beispiel

Wir ziehen zufällig farbige Kugeln aus einer Urne und legen diese nach jedem Zug wieder zurück (und mischen die Urne danach nochmal gut durch).

⇒ die Wahrscheinlichkeitsverteilung über die verschiedenen Kugeln bleibt identisch und ist unabhängig von den davor gezogenen Kugeln.

Gegenbeispiel

Wir ziehen zufällig farbige Kugeln aus einer Urne und behalten diese danach.

⇒ die Wahrscheinlichkeitsverteilung ändert sich je nachdem, welche Kugeln bereits gezogen wurden.

Satz von Bayes

Wir wissen bereits (siehe oben):

$$p(x, y) = p(x|y)p(y)$$

$$p(x, y) = p(y|x)p(x)$$

Durch Gleichsetzen erhalten wir:

$$p(y|x)p(x) = p(x|y)p(y)$$

Durch Umformung erhalten wir den Satz von Bayes:

$$\begin{aligned} p(y|x) &= \frac{p(x|y)p(y)}{p(x)} \\ &= \frac{p(x|y)p(y)}{\int p(x,y)dy} \\ &= \frac{p(x|y)p(y)}{\int p(x|y)p(y)dy} \end{aligned}$$

Satz von Bayes - Terminologie

Likelihood - Wahrscheinlichkeit, einen gewissen Wert für x bei gegebenen y zu beobachten

Prior - was wir bereits über die Verteilung von y wissen, *bevor* wir x sehen

$$p(y|x) = \frac{p(x|y)p(y)}{\int p(x|y)p(y)dy}$$

Posterior - was wir über die Verteilung von y wissen, nachdem wir x gesehen haben

Evidenz - eine Konstante, welche garantiert, dass der Posterior normiert ist

Erwartungswert

Der Erwartungswert gibt uns den erwarteten bzw Mittelwert einer Funktion $f[x]$ unter Berücksichtigung einer Verteilung $p(x)$ über x zurück.

Definition (im Diskreten):

$$\mathbb{E}[f(x)] = \sum_x f(x)p(x)$$

Definition (im Kontinuierlichen):

$$\mathbb{E}[f(x)] = \int f(x)p(x)dx$$

Varianz und Standardabweichung

Varianz

Die Varianz gibt uns die erwartete quadrierte Abweichung zum Mittelwert einer Funktion $f[x]$ unter Berücksichtigung einer Verteilung $p(x)$ über x zurück.

Definition:

$$\begin{aligned}
 Var(f(x)) &= \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2] \\
 &= \mathbb{E}[f(x)^2 - 2f(x)\mathbb{E}[f(x)] + \mathbb{E}[f(x)]^2] \\
 &= \mathbb{E}[f(x)^2] - 2\mathbb{E}[f(x)]\mathbb{E}[f(x)] + \mathbb{E}[f(x)]^2 \\
 &= \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2
 \end{aligned}$$

Standardabweichung

Die Standardabweichung entspricht der Quadratwurzel der Varianz:

$$SD(f(x)) = \sqrt{Var(f(x))}$$

Somit hat die Standardabweichung die selbe Einheit wie die Zufallsvariable x .

Shannon-Entropie

Wieviele Bits benötigen wir bei optimaler Kodierung im Durchschnitt mindestens, um Samples aus einer Wahrscheinlichkeitsfunktion p zu beschreiben?

$$H(p) = - \sum_{x \in X} p(x) \log_2(p(x)) = \mathbb{E}[-\log_2(p(X))]$$

(Bei optimaler Kodierung für p benötigen wir mindestens $-\log_2(p(x))$ Bits für ein Sample x)

Beispiel 1

Fairer Münzwurf:

$$p(x_k) = 0.5; p(x_z) = 0.5$$

$$\Rightarrow H(X) = -(0.5 \cdot (-1) + 0.5 \cdot (-1)) = 1$$

Wir benötigen also im Durchschnitt 1 Bit, um ein Sample dieser Verteilung zu beschreiben. Dazu können wir zum Beispiel "1" für Kopf und "0" für Zahl wählen.

Shannon-Entropie

Wieviele Bits benötigen wir bei optimaler Kodierung im Durchschnitt mindestens, um Samples aus einer Wahrscheinlichkeitsfunktion p zu beschreiben?

$$H(p) = - \sum_{x \in X} p(x) \log_2(p(x)) = \mathbb{E}[-\log_2(p(X))]$$

(Bei optimaler Kodierung für p benötigen wir mindestens $-\log_2(p(x))$ Bits für ein Sample x)

Beispiel 2

Etwas kompliziertere Verteilung:

$$\begin{aligned} p(x_1) &= 0.5; p(x_2) = 0.25; p(x_3) = 0.125; p(x_4) = 0.125 \\ \Rightarrow H(p) &= -(0.5 \cdot (-1) + 0.25 \cdot (-2) + 0.125 \cdot (-3) + 0.125 \cdot (-3)) \\ &= 1.75 \end{aligned}$$

Wir benötigen also im Durchschnitt 1.75 Bits, um ein Sample dieser Verteilung zu beschreiben. Dazu können wir zum Beispiel "0" für x_1 , "10" für x_2 , "110" für x_3 und "111" für x_4 mit Hilfe der Huffman-Kodierung wählen.

Kreuzentropie

Wieviele Bits benötigen wir im Durchschnitt, wenn wir anstatt einer optimalen Kodierung für p nun eine optimale Kodierung für eine andere, "falsche" Wahrscheinlichkeitsfunktion q verwenden?

$$H(p, q) = - \sum_{x \in X} p(x) \log_2(q(x)) = \mathbb{E}_p[-\log_2(q(X))]$$

Beispiel

Sei p wie im vorherigen Beispiel und q wie folgt:

$$q(x_1) = 0.25; q(x_2) = 0.5; q(x_3) = 0.125; q(x_4) = 0.125$$

$$\Rightarrow H(p, q) = -(0.5 \cdot (-2) + 0.25 \cdot (-1) + 0.125 \cdot (-3) + 0.125 \cdot (-3))$$

$$\equiv 2$$

Wie man sieht, werden nun im Durchschnitt mehr Bits benötigt (2 statt nur 1.75), als wenn wir die optimale Kodierung für p wählen würden.

Bemerkung: Die Kreuzentropie von $H(p, q)$ ist immer grösser oder gleich der Entropie $H(p)$. Falls $p = q$ gilt, ist $H(p, q) = H(p)$.

Kullback-Leibler Divergenz

Die Differenz zwischen den benötigten Bits mit der "falschen" Kodierung für q und der optimalen Kodierung für p wird auch als Kullback-Leibler Divergenz bezeichnet:

$$D_{KL}(p, q) = H(p, q) - H(p)$$

Die KL-Divergenz ist 0, wenn p und q gleich sind und wird deshalb manchmal auch als eine Art Distanz zwischen p und q interpretiert.

Beachte allerdings, dass die KL-Divergenz im Allgemeinen nicht symmetrisch ist ($D_{KL}(p, q) \neq D_{KL}(q, p)$) und somit nicht den Anforderungen an ein Distanz-Maß genügt.

Beispiel

Seien p und q wie im vorherigen Beispiel. Dann gilt:

$$D_{KL}(p, q) = H(p, q) - H(p) = 2 - 1.75 = 0.25$$

Part 3

Statistische Schätzer

Chapter 5.4 - 5.5, Goodfellow et al (2016): Deep Learning [Link]
Goodfellow-et-al-2016-Book

Punktschätzer

Seien m Datenpunkte $\{x_1, x_2, \dots, x_m\}$ i.i.d. aus einer parametrisierten Wahrscheinlichkeitsfunktion $p(x|\theta)$ mit unbekanntem Parameter θ gezogen. Dann liefert ein Punktschätzer genau eine Vorhersage $\hat{\theta}_m$ mit dem Ziel, den wahren, jedoch unbekannten Parameter θ zu schätzen:

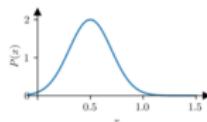
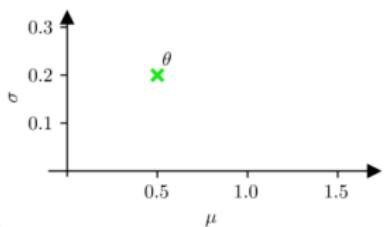
$$\hat{\theta}_m = g(x_1, \dots, x_m).$$

Diese Definition verlangt weder, dass $g(\cdot)$ einen Wert nahe am wahren Wert von θ zurückgibt, noch, dass $g(\cdot)$ im selben Wertebereich der für θ erlaubten Werte liegt. Dennoch sollte ein "guter" Punktschätzer natürlich Werte für $\hat{\theta}$ zurückgeben, welche möglichst nahe am wahren Wert von θ liegen. Wir können $g(\cdot)$ auch als einen Lernalgorithmus auffassen, mit dem die Parameter $\hat{\theta}$ gelernt werden sollen. Da die gezogenen Datenpunkte $\{x_1, x_2, \dots, x_m\}$ Zufallsvariablen sind und $\hat{\theta}_m = g(x_1, \dots, x_m)$ eine Funktion aus diesen Datenpunkten ist, kann auch $\hat{\theta}_m$ selbst als eine Zufallsvariable aufgefasst werden.

Punktschätzer

Beispiel:

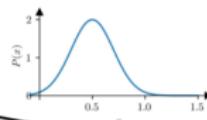
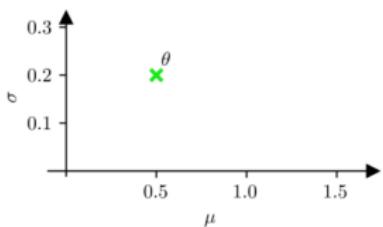
Unbekannter "wahrer" Parameter einer normalverteilten Wahrscheinlichkeitsdichte



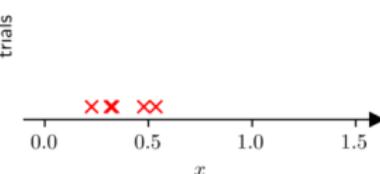
Punktschätzer

Beispiel:

Unbekannter "wahrer" Parameter einer normalverteilten Wahrscheinlichkeitsdichte



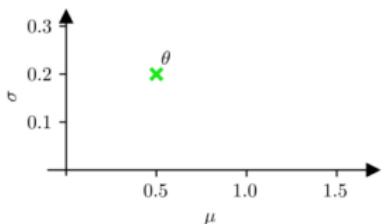
Wir führen Experimente (trials) mit m (hier: $m=5$) Datenpunkten durch



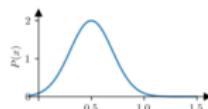
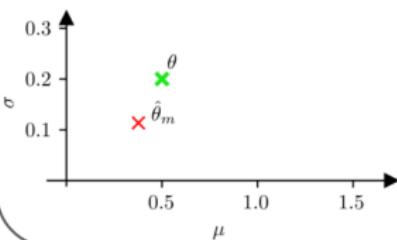
Punktschätzer

Beispiel:

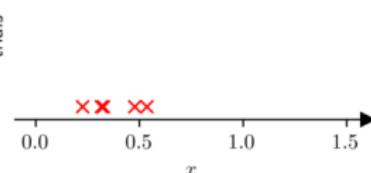
Unbekannter "wahrer" Parameter einer normalverteilten Wahrscheinlichkeitsdichte



Die geschätzten Parameter sollten möglichst nahe am "wahren" Parameter liegen



Wir führen Experimente (trials) mit m (hier: $m=5$) Datenpunkten durch



Für jedes Experiment schätzen wir den Parameter:

$$\hat{\theta}_m = g(x_1, x_2, \dots, x_m)$$

zum Beispiel:

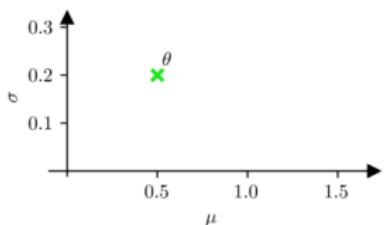
$$\hat{\mu}_m = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\hat{\sigma}_m^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu}_m)^2$$

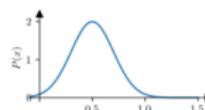
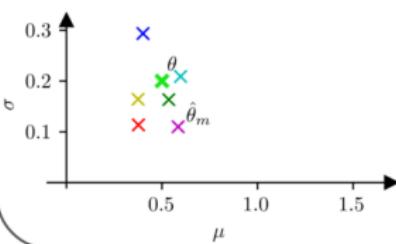
Punktschätzer

Beispiel:

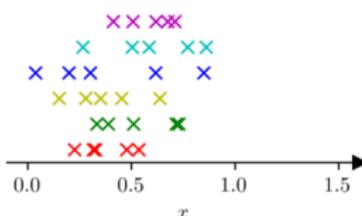
Unbekannter "wahrer" Parameter einer normalverteilten Wahrscheinlichkeitsdichte



Die geschätzten Parameter sollten möglichst nahe am "wahren" Parameter liegen



Wir führen Experimente (trials) mit m (hier: $m=5$) Datenpunkten durch



Für jedes Experiment schätzen wir den Parameter:

$$\hat{\theta}_m = g(x_1, x_2, \dots, x_m)$$

zum Beispiel:

$$\hat{\mu}_m = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\hat{\sigma}_m^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu}_m)^2$$

Punktschätzer - Erwartungswert / Varianz / Standardfehler

Wann ist ein Punktschätzer "gut"?

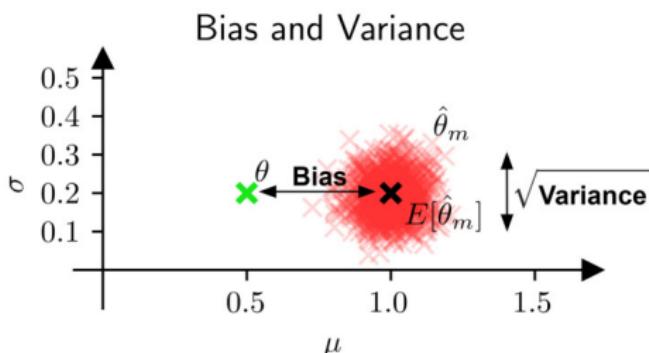


Figure: Beispiel eines Sch鋐zers mit Bias und Varianz

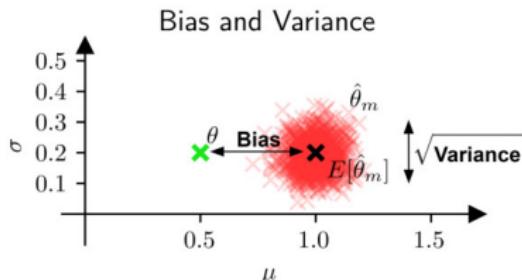
Da $\hat{\theta}_m$ ebenfalls eine Zufallsvariable ist, können wir auch für sie Erwartungswert $\mathbb{E}[\hat{\theta}_m]$, Varianz und Standardabweichung bestimmen:

$$\text{Var}(\hat{\theta}_m) = \mathbb{E}[(\hat{\theta}_m - \mathbb{E}[\hat{\theta}_m])^2] = \mathbb{E}[\hat{\theta}_m^2] - \mathbb{E}[\hat{\theta}_m]^2$$

$$\text{SD}(\hat{\theta}_m) = \sqrt{\text{Var}(\hat{\theta}_m)}$$

Punktschätzer - Bias

Wann ist ein Punktschätzer "gut"?



Der Bias eines Punktschätzers $\hat{\theta}_m$ ergibt sich aus der Differenz zwischen seinem Erwartungswert $\mathbb{E}[\hat{\theta}_m]$ und dem wahren Parameter θ :

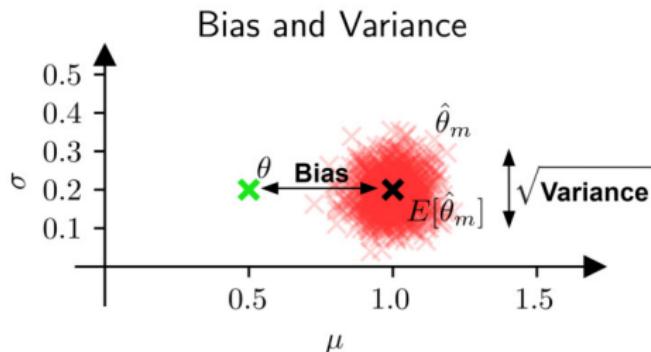
$$\text{Bias}(\hat{\theta}_m) = \underbrace{\mathbb{E}[\hat{\theta}_m]}_{\substack{\text{expectation} \\ \text{over data}}} - \underbrace{\theta}_{\text{true value}}.$$

Wenn der Bias des Punktschätzers gleich Null ist, so nennen wir den Schätzer *unbiased*. Wenn der Bias nur im Limit für viele Datenpunkte ($\lim_{m \rightarrow \infty}$) gegen Null geht, dann nennen wir den Schätzer *asymptotisch unbiased*, d.h.:

$$\lim_{m \rightarrow \infty} \mathbb{E}(\hat{\theta}_m) - \theta = 0, \quad \text{bzw.} \quad \lim_{m \rightarrow \infty} \mathbb{E}(\hat{\theta}_m) = \theta.$$

Punktschätzer - Mean Squared Error

Wann ist ein Punktschätzer "gut"?



Das Ziel eines guten Schätzers ist es, den *Mean Squared Error (MSE)* zu minimieren:

$$\text{MSE}(\hat{\theta}_m) = \mathbb{E}[(\hat{\theta}_m - \theta)^2]$$

Durch Einsetzen lässt sich zeigen, dass:

$$\text{MSE}(\hat{\theta}_m) = \text{Bias}(\hat{\theta}_m)^2 + \text{SD}(\hat{\theta}_m)^2 = \text{Bias}(\hat{\theta}_m)^2 + \text{Var}(\hat{\theta}_m)$$

⇒ ein guter Schätzer minimiert sowohl den Bias als auch die Varianz / SD.

Punktschätzer - Bias / Varianz Trade-off

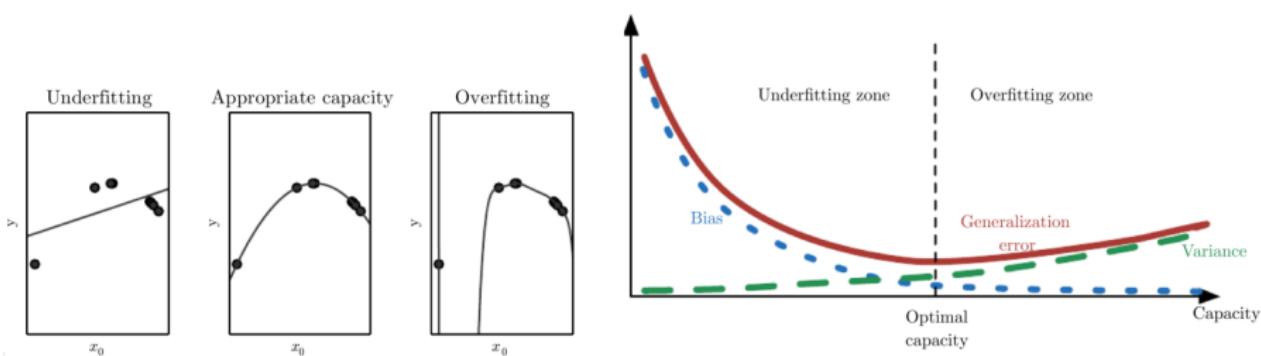


Figure: Figuren 5.2 und 5.6 von Goodfellow zu under- und overfitting.

Wenn ein Modell zu geringe Kapazität hat, so wird der Bias hoch, weil die wahren Parameter θ nicht korrekt durch $\hat{\theta}$ approximiert werden können.

Wenn ein Modell zu hohe Kapazität hat, so wird die Varianz hoch, weil es viele verschiedene Möglichkeiten für $\hat{\theta}$ gibt, die selben Daten zu approximieren.

⇒ wir müssen einen optimalen Trade-off zwischen geringer Kapazität (underfitting / hoher Bias) und hoher Kapazität (overfitting / hohe Varianz) finden.

Punktschätzer - Konsistenz

Was passiert, wenn wir die Anzahl der Datenpunkte m erhöhen?

Definition: Ein Punkt schätzer $\hat{\theta}_m$ des Parameters θ ist genau dann *konsistent*, wenn:

$$\text{p-lim}_{m \rightarrow \infty} \hat{\theta}_m = \theta$$

Wobei m der Anzahl der zur Verfügung stehenden Datenpunkte entspricht und der p -lim-Operator "Konvergenz in Wahrscheinlichkeit" bedeutet, d.h.:

$$\forall \epsilon > 0 : \lim_{m \rightarrow \infty} P(|\hat{\theta}_m - \theta| > \epsilon) = 0$$

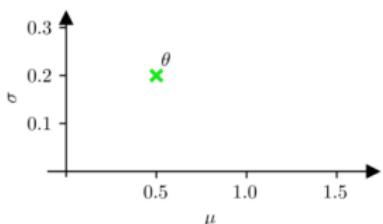
⇒ Wenn ein Schätzer konsistent ist, dann wird der MSE bei genügend vielen Datenpunkten ($m \rightarrow \infty$) beliebig klein

⇒ Konsistenz ist eine wichtige Eigenschaft von "guten" Punktschätzern!

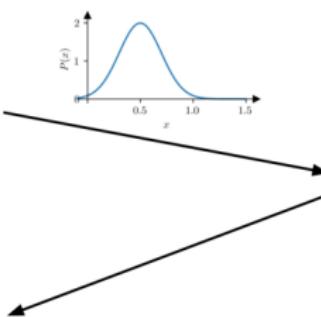
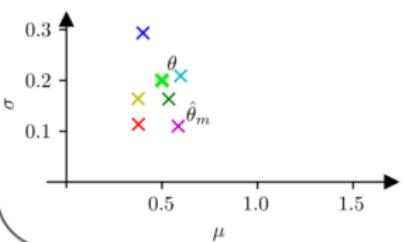
Punktschätzer - Beispiel

Beispiel (Wiederholung von oben):

Unbekannter "wahrer" Parameter einer normalverteilten Wahrscheinlichkeitsdichte



Die geschätzten Parameter sollten möglichst nahe am "wahren" Parameter liegen



Für jedes Experiment schätzen wir den Parameter:

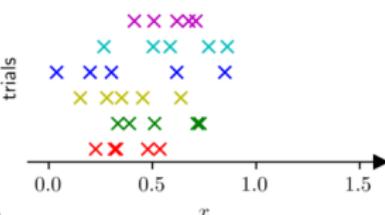
$$\hat{\theta}_m = g(x_1, x_2, \dots, x_m)$$

zum Beispiel:

$$\hat{\mu}_m = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\hat{\sigma}_m^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \hat{\mu}_m)^2$$

Wir führen Experimente (trials) mit m (hier: $m=5$) Datenpunkten durch.

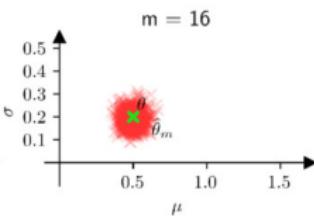
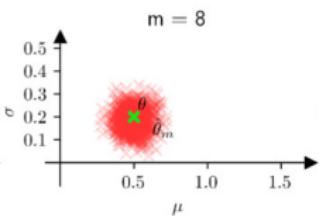
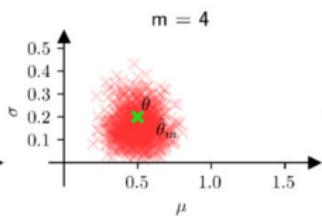
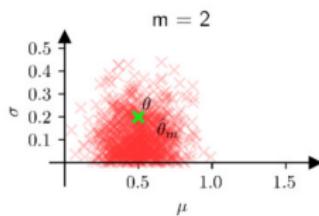


Punktschätzer - Beispiel

Hier möchten wir das Verhalten von Parameter-Schätzern für die Parameter μ und σ einer Gaußverteilung bei unterschiedlichen Datenpunktanzahlen m betrachten. Dazu führen wir für $m = 2, 4, 8, 16$ jeweils viele Experimente durch und plotten die Verteilungen der geschätzten $\hat{\theta}_m$:

Unbiased Mean: $\hat{\mu}_m \equiv \frac{1}{m} \sum_{i=1}^m x_i$

Biased Variance: $\hat{\sigma}_m^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \hat{\mu}_m)^2$

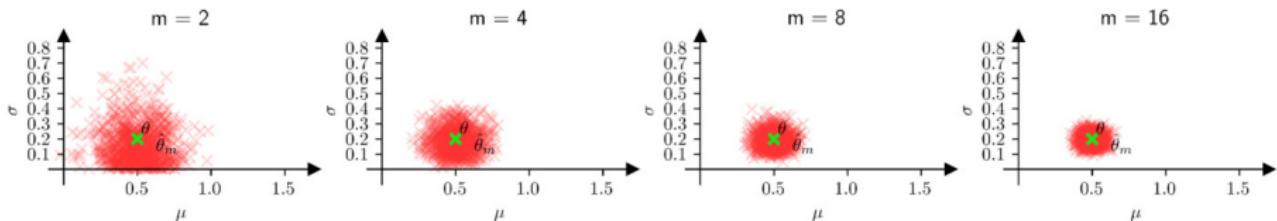


Man sieht, dass die Schätzer für $\hat{\mu}_m$ und $\hat{\sigma}_m$ konsistent sind. Während der Mean-Schätzer unbiased ist, wird die Varianz jedoch bei kleinen m im Mittel unterschätzt (der Varianzschätzer hat einen negativen Bias). Der Varianzschätzer ist also nur asymptotisch unbiased.

Punktschätzer - Beispiel

Unbiased Mean: $\hat{\mu}_m = \frac{1}{m} \sum_{i=1}^m x_i$

$$\text{Unbiased Variance: } \hat{\sigma}_m^2 = \frac{1}{m-1} \sum_{i=1}^m (x_i - \hat{\mu}_m)^2$$



Wenn wir im Varianzschätzer $\frac{1}{m-1}$ anstatt $\frac{1}{m}$ als Vorfaktor verwenden, dann wird er ebenfalls unbiased.

Beweis: siehe Übungen.

Maximum Likelihood Schätzer

Bisher haben wir einen Punktschätzer $g(\cdot)$ einfach als eine Funktion von i.i.d. gezogenen Daten $\mathbf{x} = \{x_1, x_2, \dots, x_m\}$ definiert, d.h.:

$$\hat{\theta}_m = g(x_1, \dots, x_m).$$

Aber wie können wir systematisch einen "guten" Schätzer $g(\cdot)$ finden?

Eine weit verbreitete Methode ist der *Maximum Likelihood Schätzer*, welcher $\hat{\theta}_{ML}$ mit Hilfe des Maximum-Likelihood Prinzips bestimmt:

$$\hat{\theta}_{\text{ML}} = \underset{\theta}{\operatorname{argmax}} p_{\text{model}}(\mathbf{X}|\theta) = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^m p_{\text{model}}(x_i|\theta)$$

Die Zerlegung von $p_{\text{model}}(\mathbf{X}|\theta)$ in ein Produkt über die Datenpunkte ist möglich, da die Daten i.i.d. gezogen wurden.

Man kann zeigen, dass dieser Schätzer konsistent und effizient ist, d.h. schnell gegen das echte θ konvergiert, wenn:

- Die wahre Verteilung p_{data} in der Modellfamilie liegt
 - p_{data} genau einem Wert von θ entspricht

Negative Log-Likelihood (NLL)

Da das Produkt im ML-Schätzer zu numerischen Problemen (overflow / underflow) führen kann, wendet man üblicherweise den Logarithmus an:

$$\begin{aligned}
 \hat{\theta}_{\text{ML}} &= \underset{\theta}{\operatorname{argmax}} p_{\text{model}}(\mathbf{X}|\theta) = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^m p_{\text{model}}(x_i|\theta) \\
 &= \underset{\theta}{\operatorname{argmax}} \log \left(\prod_{i=1}^m p_{\text{model}}(x_i|\theta) \right) // \text{Log ist streng monoton steigend} \\
 &= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^m \log p_{\text{model}}(x_i|\theta) // \text{Produkt aus Log ziehen ergibt Summe} \\
 &= \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^m -\log p_{\text{model}}(x_i|\theta) // \text{Optimierer minimieren üblicherweise}
 \end{aligned}$$

Der letzte Term wird auch *Negative Log-Likelihood (NLL)* genannt.

Negative Log-Likelihood und Kreuzentropie

Wenn man beachtet, dass die Samples x_i aus der "wahren" datengenerierenden Verteilung $p_{\text{data}}(x)$ gezogen werden und die Negative Log-Likelihood durch die Anzahl der gezogenen Samples m dividiert, so entspricht dieser Ausdruck für $m \rightarrow \infty$ dem gesamten Integral der *Kreuzentropie* zwischen der modellierten Verteilung $p_{\text{model}}(x|\theta)$ und der "wahren" datengenerierenden Ziel-Verteilung $p_{\text{data}}(x)$:

$$\lim_{m \rightarrow \infty} \frac{1}{m} \sum_{i=1}^m -\log p_{\text{model}}(x_i | \theta) = \int_{\Omega} -p_{\text{data}}(x) \log p_{\text{model}}(x | \theta) dx \approx H(p_{\text{data}}, p_{\text{model}})$$

Somit entspricht die Minimierung der negativen Log-Likelihood auch der Minimierung der Kreuzentropie zwischen p_{data} und p_{model} . Wie oben bereits erwähnt, wird die Kreuzentropie minimiert, wenn p_{data} und p_{model} so ähnlich wie möglich sind.

Bedingte Negative Log-Likelihood

Wir können die Maximum-Likelihood Schätzung von θ auch auf bedingte Verteilungen $p(\mathbf{Y}|\mathbf{X}, \theta)$ verallgemeinern, welche \mathbf{Y} für gegebene \mathbf{X} vorhersagen:

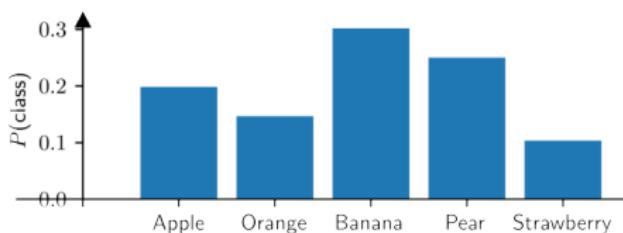
$$\begin{aligned}\hat{\theta}_{\text{ML}} &= \underset{\theta}{\operatorname{argmax}} p(\mathbf{Y}|\mathbf{X}, \theta) = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^m p(y_i|x_i, \theta) \\ &= \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^m -\log p(y_i|x_i, \theta)\end{aligned}$$

In vielen Fällen (z.B.: Klassifikation / Segmentierung / Regression) möchten wir die Ausgabe eines neuronalen Netzes als bedingte Wahrscheinlichkeitsfunktion interpretieren.

Tatsächlich können wir die Optimierung bezüglich einigen der wichtigsten Loss-Funktionen als Maximum-Likelihood Schätzer auffassen. Darunter:

- Cross-Entropy zur Klassifikation / Segmentierung
 - Mean Squared Error (L_2 -Loss) zur Regression
 - Mean Absolute Error (L_1 -Loss) zur Regression

Cross-Entropy zur Klassifikation / Segmentierung



In Klassifikations- und Segmentierungsproblemen lernt ein neuronales Netz üblicherweise direkt eine Wahrscheinlichkeitsfunktion $p_\theta(y|x)$. Daraus ergibt sich dann:

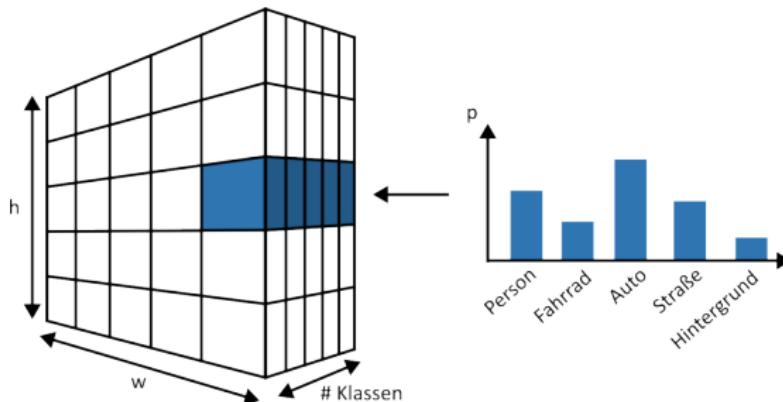
$$\text{Likelihood: } p(\mathbf{Y}|\mathbf{X}, \theta) = \prod_i p_{\theta}(y_i|x_i)$$

$$\text{NLL: } -\log p(\mathbf{Y}|\mathbf{X}, \theta) = \sum_i -\log p_\theta(y_i|x_i)$$

Daraus folgt direkt der Cross-Entropy Loss: $H = \frac{1}{m} \sum_{i=1}^m -\log p_\theta(y_i|x_i)$

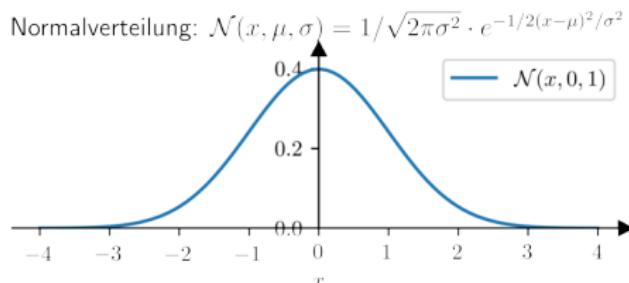
Segmentierung \approx Pixel-weise Klassifikation

Bei Segmentierungsproblemen gibt ein Netzwerk üblicherweise für jeden Pixel eine Wahrscheinlichkeit für jede Klasse an:



Ein Loss kann dann analog zur Klassifizierung aus der gemittelten Cross-Entropy über alle Pixel gebildet werden.

Mean Squared Error (MSE) zur Regression



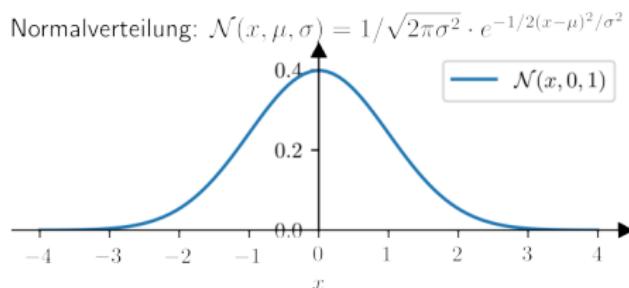
In Regressionsproblemen lernt ein neuronales Netz üblicherweise den Mean $\mu_\theta(x)$ einer Wahrscheinlichkeitsdichte. Wenn wir die Normalverteilung betrachten, folgt daraus:

$$\text{Likelihood: } p(\mathbf{Y}|\mathbf{X}, \theta, \sigma) = \prod_i \mathcal{N}(y_i, \mu_\theta(x_i), \sigma)$$

$$\text{NLL: } -\log p(\mathbf{Y}|\mathbf{X}, \theta, \sigma) = \sum_i \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2} \frac{(y_i - \mu_\theta(x_i))^2}{\sigma^2}$$

Wenn wir den Faktor σ als konstant ($\sigma(x) = \sigma$) betrachten, ergibt sich daraus - abgesehen von einem konstanten Vorfaktor und Offset - der Mean Squared Error:

Mean Squared Error (MSE) zur Regression

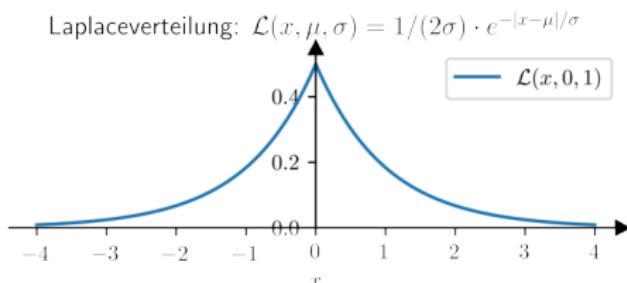


Der Mean Squared Error (oder auch L_2 Error) ist definiert als:

$$\text{MSE} = \frac{1}{m} \sum_{i=1}^m (y_i - \mu_\theta(x_i))^2$$

Das Minimum des MSE ist genau dann erreicht, wenn $\mu_\theta(x)$ dem **Mean** der Samples aus der datengenerierenden bedingten Verteilung $p_{\text{data}}(y|x)$ entspricht. Dies kann man zeigen, indem man die Ableitung des MSE nach $\mu_\theta(x)$ gleich Null setzt und argumentiert, dass der MSE konvex bezüglich μ ist. Somit ist der MSE *sensitiv gegenüber Outliern*.

Mean Absolute Error (MAE) zur Regression



In Regressionsproblemen lernt ein neuronales Netz üblicherweise den Mean $\mu_\theta(x)$ einer Wahrscheinlichkeitsdichte. Wenn wir eine Laplaceverteilung betrachten, folgt daraus:

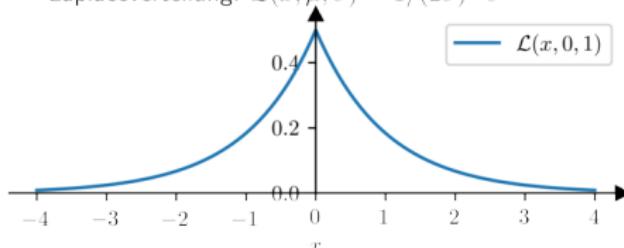
$$\text{Likelihood: } p(\mathbf{Y}|\mathbf{X}, \theta, \sigma) = \prod_i \mathcal{L}(y_i, \mu_\theta(x_i), \sigma)$$

$$\text{NLL: } -\log p(\mathbf{Y}|\mathbf{X}, \theta, \sigma) = \sum_i \log(2\sigma) + \frac{|y_i - \mu_\theta(x_i)|}{\sigma}$$

Wenn wir den Faktor σ als konstant ($\sigma(x) = \sigma$) betrachten, ergibt sich daraus - abgesehen von einem konstanten Vorfaktor und Offset - der Mean Absolute Error,

Mean Absolute Error (MAE) zur Regression

Laplaceverteilung: $\mathcal{L}(x, \mu, \sigma) = 1/(2\sigma) \cdot e^{-|x-\mu|/\sigma}$



Der Mean Absolute Error (oder auch L_1 Error) ist definiert als:

$$\text{MAE} = \frac{1}{m} \sum_{i=1}^m |y_i - \mu_\theta(x_i)|$$

Das Minimum des MAE ist genau dann erreicht, wenn $\mu_\theta(x)$ dem *Median* der Samples aus der datengenerierenden bedingten Verteilung $p_{\text{data}}(y|x)$ entspricht. Dies kann man zeigen, indem man die Ableitung des MAE nach $\mu_\theta(x)$ gleich Null setzt und argumentiert, dass der MAE konvex bezüglich μ ist. Somit ist der MAE *robust* gegenüber *Outliern*.

Maximum A Posteriori Estimation

Eine Alternative zum ML-Schätzer ist der *Maximum A Posteriori* (MAP) Schätzer, welcher wie folgt definiert ist:

$$\hat{\theta}_{\text{MAP}} = \underset{\theta}{\operatorname{argmax}} p(\theta | \mathbf{X}).$$

Indem wir θ als Zufallsvariable behandeln und Bayes' Theorem anwenden, bekommen wir:

$$\begin{aligned}\hat{\theta}_{\text{MAP}} &= \underset{\theta}{\operatorname{argmax}} \frac{p(\mathbf{X}|\theta)p(\theta)}{p(\mathbf{X})} // p(\mathbf{X}) \text{ hängt nicht von } \theta \text{ ab} \\ &= \underset{\theta}{\operatorname{argmax}} \left[\underbrace{\log p(\mathbf{X}|\theta)}_{\text{log-likelihood}} + \underbrace{\log p(\theta)}_{\text{log prior}} \right] // \text{wende logarithmus an} \\ &= \underset{\theta}{\operatorname{argmax}} \left[\sum_{i=1}^m \log p(x_i|\theta) + \log p(\theta) \right].\end{aligned}$$

Maximum A Posteriori Estimation

Analog kann der MAP Schätzer auch für bedingte Wahrscheinlichkeiten $P(\mathbf{Y}|\mathbf{X}, \theta)$ angewendet werden:

$$\hat{\theta}_{\text{MAP}} = \operatorname{argmax}_{\theta} \left[\underbrace{\log p(\mathbf{Y} | \mathbf{X}, \theta)}_{\text{conditional log-likelihood}} + \underbrace{\log p(\theta)}_{\text{log prior}} \right]$$

$$= \operatorname{argmax}_{\theta} \left[\sum_{i=1}^m \log P(y_i | x_i, \theta) + \log p(\theta) \right].$$

⇒ Der Prior ist üblicherweise unbekannt. Dennoch möchte man manchmal die Parameter θ auf "realistische Werte" mit Hilfe eines Priors einschränken. Dies wird später noch ein wichtiges Konzept für die *Regularisierung* der Gewichte eines neuronalen Netzes werden.

..../deeplearn deeplearn /deeplearn