

1 Einführung

Mittwoch, 24. Mai 2023 14:23

Definition Lernen:

„Lernen ist jede Veränderung in einem System, die es ihm erlaubt, beim nächsten Mal dieselbe Aufgabe oder eine Aufgabe derselben Art effizienter oder effektiver auszuführen oder überhaupt auszuführen.“

Definition Maschinelles Lernen:

| Ein System lernt aus **Erfahrung E** in Hinblick auf eine Klasse von **Aufgaben T** und einem **Performanzmaß P**, wenn seine Leistungen bei Aufgaben aus **T** gemessen mit **P** durch **Erfahrung aus E** steigt. – Tom Mitchell

| Ein lernendes System generiert eine oder mehrere **Lösungshypothese(n) h** um **Aufgabe T** zu lösen.

| **Beispiel:** Schach

- **T:** Schach spielen
- **P:** Schach gewinnen (z.B. Prozent gewonnener Spiele)
- **E:** Spiele gegen sich selbst und andere Gegner
- **h:** Verfahren, welches gute Spielzüge anhand der aktueller Situation prädiziert



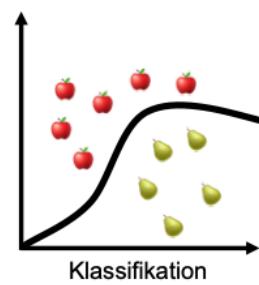
Lernarten



Überwachtes Lernen (Supervised Learning)

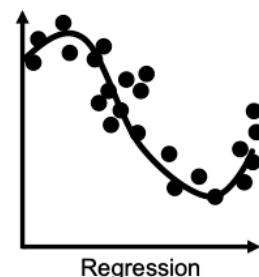
Definition:

- **Geg.:** N Trainingsbeispiele $\{(x_1, y_1), \dots, (x_N, y_N)\}$ mit Eingangsdaten x_i und Ausgangsdaten y_i , Modell, Kostenfunktion
- **Ges.:** Trainiertes Modell, welches den Zusammenhang zwischen x_i (Eingabe) und y_i (Ausgabe) korrekt lernt.



Prädiktionsart: Welche Art von Ausgabewert hat mein Modell?

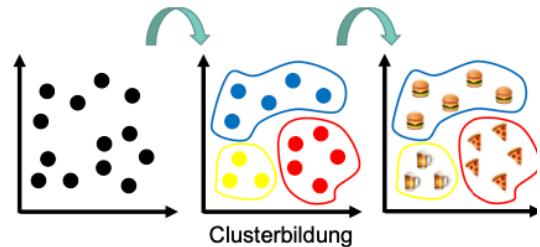
- **Klassifikation:** Diskrete Zufallsvariable
 - **Beispiel:** Zufallsvariable nimmt die diskrete Klasse Apfel oder Birne an.
- **Regression:** Kontinuierliche Zufallsvariable
 - **Beispiel:** Zufallsvariable nimmt den kontinuierlichen Preis für einen Apfel an.



Unüberwachtes Lernen (Unsupervised Learning)

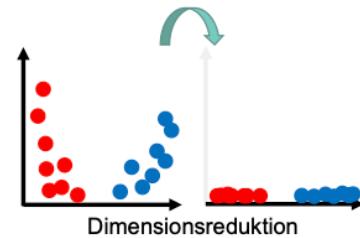
Definition:

- **Geg.:** N Trainingsbeispiele $\{x_1, \dots, x_N\}$ mit Eingangsdaten x_i , Modellklasse, Kostenfunktion
- **Ges.:** Trainiertes Modell, welches nützliche Muster oder strukturelle Eigenschaften der Daten erlernt hat.



Aufgaben:

- **Clusterbildung/Segmentierung:** Gruppiere eine Menge von Objekten so, dass die Objekte in derselben Gruppe (Cluster) einander ähnlicher sind als die Objekte in anderen Gruppen.
- **Dimensionsreduktion:** Lerne eine Transformation der Daten von einem höher- in einen niedrig-dimensionalen Raum unter Beibehaltung der relevanten Eigenschaften der Daten.
- ...



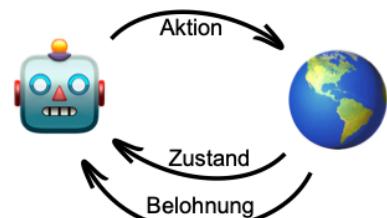
Bestärkendes Lernen (Reinforcement Learning)

Definition:

- **Geg.:** RL-Agent, Belohnungsfunktion, Umgebung
- **Ges.:** Trainierter RL-Agent, welcher die Belohnung (Reward) auf lange Sicht maximiert.

Unterschiede zu bisherigen Lernarten:

- Lernen ist **aktiv statt passiv** – Kein fester Datensatz, Daten/Erfahrung wird gesammelt durch Interaktion mit der Umgebung.
- **Keine Überwachung**, nur ein Belohnungssignal
- **Interaktionen sind oft sequentiell** – Zukünftige Entscheidungen können von vorherigen abhängen.
- Wir können **optimales Verhalten lernen ohne Beispiele davon**.

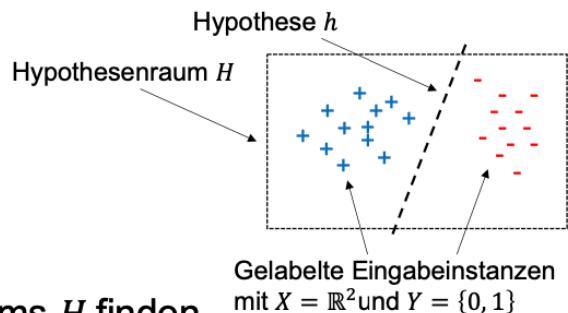


2 Induktives Lernen

Mittwoch, 24. Mai 2023 14:33

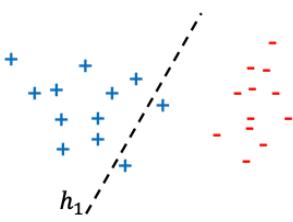
Lernmaschine

- Jedes maschinelle Lernverfahren benötigt Daten aus einem Eingaberaum X
- Eine Eingabeinstanz $x_i \in X$ besteht aus Attributen $x_i = \begin{pmatrix} x_{i1} \\ \vdots \\ x_{im} \end{pmatrix}$
- Alle Eingabeinstanzen sollen in den Lösungsraum Y abgebildet werden
 - Häufig Klassifikation oder Regression
 - Für eine Wahr/Falsch Klassifikation wäre $Y = \{0, 1\}$
- **Annahme:** Es existiert eine Zielfunktion $t : X \rightarrow Y$ die alle möglichen Eingabeinstanzen in X in den Zielraum Y perfekt abbildet
 - t ist unbekannt
 - Die Zahl aller möglichen Eingabeinstanzen ist meist exorbitant
- Eine Hypothese h ist ein „beliebiges“ Mapping $h: X \rightarrow Y$
- Der Hypothesenraum H , enthält alle möglichen Hypothesen $h \in H$ die durch ein Modell dargestellt werden können

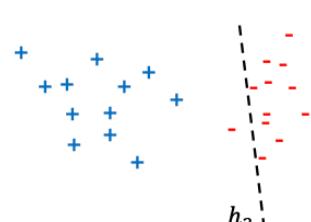


- **Aufgabe:** Ein Modell soll mithilfe maschinellen Lernens die Hypothese $h \approx t$ innerhalb seines Hypothesenraums H finden

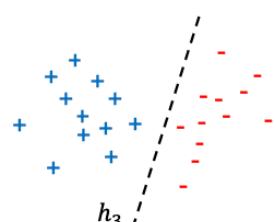
- **Konsistenz**
 - Keine negativen Beispiele werden positiv klassifiziert
- **Vollständigkeit**
 - Alle positiven Beispiele werden positiv klassifiziert



Konsistente Hypothese h_1
(nicht vollständig)



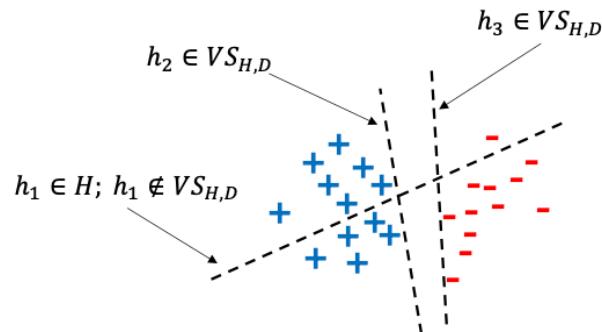
Vollständige Hypothese h_2
(nicht konsistent)



Vollständige und konsistente Hypothese h_3

Versionsraum

Definition: Der Versionsraum $VS_{H,D}$ bezüglich des Hypothesenraums H und der Menge von Trainingsbeispielen D ist die Untermenge der Hypothesen von H , die mit den Trainingsbeispielen in D **konsistent und vollständig** sind.



Induktion vs Deduktion

Induktion:	Deduktion:
<ul style="list-style-type: none"> - Prozess des plausiblen Schließens vom Speziellen zum Allgemeinen (Generalisieren) - Basis: große Anzahl zutreffender Fälle 	<ul style="list-style-type: none"> - Prozess des korrekten Schließens vom Allgemeinen zum Speziellen - Basis: hauptsächlich vorhandenes Hintergrundwissen (Regeln)
Wahrheitserweiternd (i.A. Generierung neuer Hypothesen)	Wahrheitserhaltend (i.A. Ableiten neuer Regeln / Fakten)
Grundsatz angelehnt an s.g. induktiven Lernhypothese	Logischer Schluss
Plausibilität	Korrektheit
Sehr weit verbreiteter Lernansatz (an Biologie angelehnt)	

Die induktive Lernhypothese

Herausforderung: Dimension von Eingaberaum X oft sehr hoch. Bsp.: Für ein 8-Bit-RGB-Bild mit 20x20px ist $\dim(X) = (256 \cdot 256 \cdot 256)^{20 \cdot 20}$. Zusätzlich ist t unbekannt

Lösung:

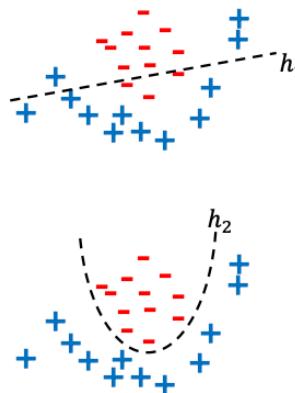
- Erstelle Datensatz, der Teilmenge von X ist
- Erstelle Fehlerfunktion E , die den Abstand einer Hypothese zu t schätzt
- Lerne Hypothese für die E minimal ist

Definition: Jede Hypothese h , die die Zielfunktion t über einer genug großen Menge von Trainingsinstanzen $x_i \in D$ gut genug approximiert, wird die Zielfunktion auch über unbekannte Beispiele gut approximieren.

Wenn eine Lernmaschine auch bei unbekannten Daten gute Prädiktionen tätigt nennen wir das **Generalisierung**

Hypothesenraum und Versionenraum

- Probleme bei Modellen mit geringem Hypothesenraum
 - Der Hypothesenraum H_{lin} eines linearen Modells enthält nur lineare Hypothesen.
 - In **Beispiel 1** existiert keine konsistente und vollständige lineare Hypothese.
 - Versionsraum $VS_{H,D} = \emptyset$
 - Zielfunktion t nicht in H_{lin} enthalten!
- Modell mit größerem Hypothesenraum benötigt, z.B. Raum der Polynome H_{pol} . Zu sehen in **Beispiel 2**.
 - Modelle mit unendlichem Hypothesenraum immer am besten?



Beispiel 1:
Linearer Hypothesenraum H_{lin} enthält keine **konsistente** und **vollständige** Hypothese um die Zielfunktion abzubilden.
 $h_1 \in H_{lin}$

Beispiel 2:
Polynomieller Hypothesenraum H_{pol} enthält die Zielfunktion.
 $h_2 \notin H_{lin}$
 $h_1, h_2 \in H_{pol}$

Induktiver Bias

更喜欢: 提升 优先选择 优先权标准

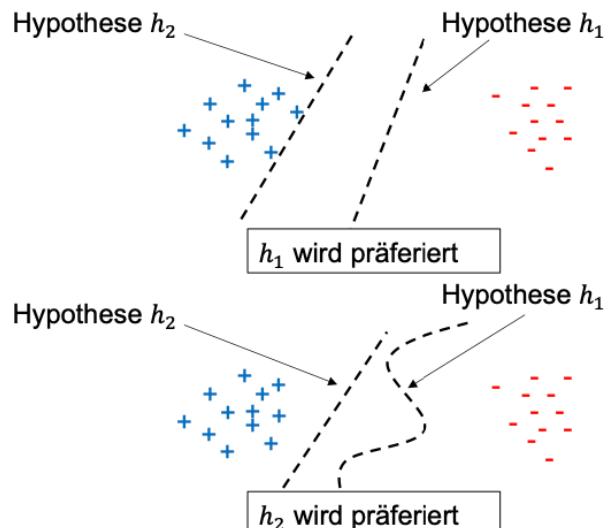
Bestimmte Hypothesen werden über anderen **präferiert** (Vorzugskriterium)

-> Erleichtert Lernprozess und Generalisierung

Bsp.:

Die Hypothese die den Abstand zwischen Eingabeinstanzen unterschiedlicher Klassen maximiert ist zu bevorzugen

Wenn eine einfache Hypothese und eine komplexe Hypothese gleichermaßen die Fehlerfunktion minimiert ist die einfachere Hypothese zu bevorzugen



3 Lerntheorie

Donnerstag, 25. Mai 2023 10:21

Ockhamsche Prinzip

"Löse nie ein Problem komplizierter als nötig, denn die einfachste, richtige Erklärung, ist die Beste."

In der Lerntheorie wird versucht die Beste Lösung zu finden

Lernmaschine

- **Definition Lernmaschine:** Eine lernende Maschine wird bestimmt durch ...
 - **Hypothesenraum** H mit den Hypothesen $h_\theta \in H$, wobei θ Parameter sind
 - **Beispiel:** Lineares Modell $h_\theta(x) = \text{sgn}(mx + c)$, mit $\theta = (m, c)$ wobei $m \in M$ und $c \in C$
 - Die Menge von allen Parameterkombinationen $M \times C$ spannt den Hypothesenraum H_{lin} auf.
 - Eine bestimmte Parameterkombination θ gilt als eine Hypothese $h_\theta \in H_{lin}$.
 - **Lernverfahren:** Die Methode, um die optimale Hypothese h_{opt} mit Hilfe von Lernbeispielen zu finden (benötigt Fehlerfunktion, Optimierungsmethode, ...)
 - **Herausforderung:** Welche Lernmaschine ist zu wählen?
 - Welcher Hypothesenraum? Linear? Nicht-linear? Parametrisch? Nicht-parametrisch? ...
 - Welches Lernverfahren? Welche Optimierung? Welche Fehlerfunktion? ...
 - Welche „guten“ Hypothese(n) soll(en) gewählt werden? Welche Metriken?

Formale Beschreibung - Überwachtes Lernen

Lernen = Finden einer optimalen Hypothese $h_{opt}: X \rightarrow Y$ im Hypothesenraum H

Dazu gibt es Trainingsdatensatz bestehend aus Eingangsdaten $x_i \in X$ und Ausgangsdaten $y_i \in Y$

Fehlerminimierung

Motivation: Finde die optimale Hypothese h_{opt} durch Minimierung der Fehler-/Kostenfunktion $J(h_\theta)$.

Verlustfunktion (Loss function): $L(h_\theta(x_i), y_i)$

- Fehler zwischen der prädizierten Ausgabe $h_\theta(x_i)$ der Hypothese und der Sollausgabe y_i für ein Beispiel, bzw. einen Datenpunkt

Kostenfunktion (Cost function):

$$J(h_\theta) = \mathbb{E}_{(x,y) \sim p}[L(h_\theta(x), y)] = \int L(h_\theta(x), y) p(x, y) dx dy$$

- Erwarteter Fehler über alle Daten der Wahrscheinlichkeitsdichtefunktion $p(x, y)$.
- Auch genannt erwarteter Generalisierungsfehler oder Risiko

Probleme: Viele Werte müssen nicht berücksichtigt werden!

Problem: $J(h_\theta)$ kann oft nicht berechnet werden!

- Kein Zugang zu der zu Grunde liegenden Datenverteilung $p(x, y)$, sondern nur einer kleinen Stichprobe an Daten $\hat{p}(x, y)$
- Mit allen Daten nicht berechenbar in endlicher Zeit.
- ...

Approximiere $J(h_\theta)$! → Empirische Fehler-/Risikominimierung

Empirischer Fehler:

Empirische Verteilung $\hat{p}(x, y)$ definiert durch die Daten $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$

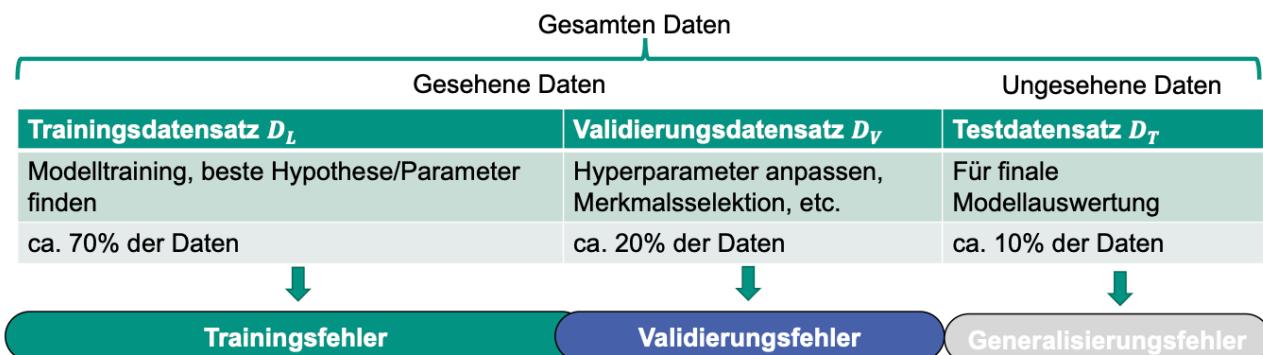
Empirischer Fehler:

$$\hat{J}_D(h_\theta) = \mathbb{E}_{(x,y) \sim \hat{p}}[L(h_\theta(x), y)] = \frac{1}{|D|} \sum_{(x,y) \in D} L(h_\theta(x), y)$$

- Erwarteter Fehler über die geschätzte Datenverteilung $\hat{p}(x, y)$ statt über die zu Grunde liegende Verteilung $p(x, y)$

Folglich: Statt den realen Fehler direkt zu minimieren, minimieren wir den empirischen Fehler und hoffen dass der tatsächliche Fehler in gleichem Maße sinkt

Daher werden die Daten aufgeteilt:



Das Lernen ergibt sich somit aus der Minimierung des Fehlers. Um dies zu erreichen, können Optimierungsansätze genutzt werden, wie zum Beispiel das Gradientenabstiegsverfahren. Hier wird der Fehler iterativ kleiner. Problem lokale Minima und Overfitting.

Herausforderungen beim Lernen

Statistisches Problem

- Das Verfahren betrachtet einen – gemessen an der Menge von Trainingsdaten – „zu großen“ Hypothesenraum.
- Auf Basis der Trainingsdaten eignen sich mehrere Hypothesen gleichermaßen gut.

Komplexitätsproblem

- Aufgrund der Komplexität des Problems kann das Lernverfahren nicht das Finden einer optimalen Lösung innerhalb des Hypothesenraumes garantieren.
- Bei Verwendung von speziellen Verfahren oder Heuristiken besteht die Gefahr einer suboptimalen Lösung.

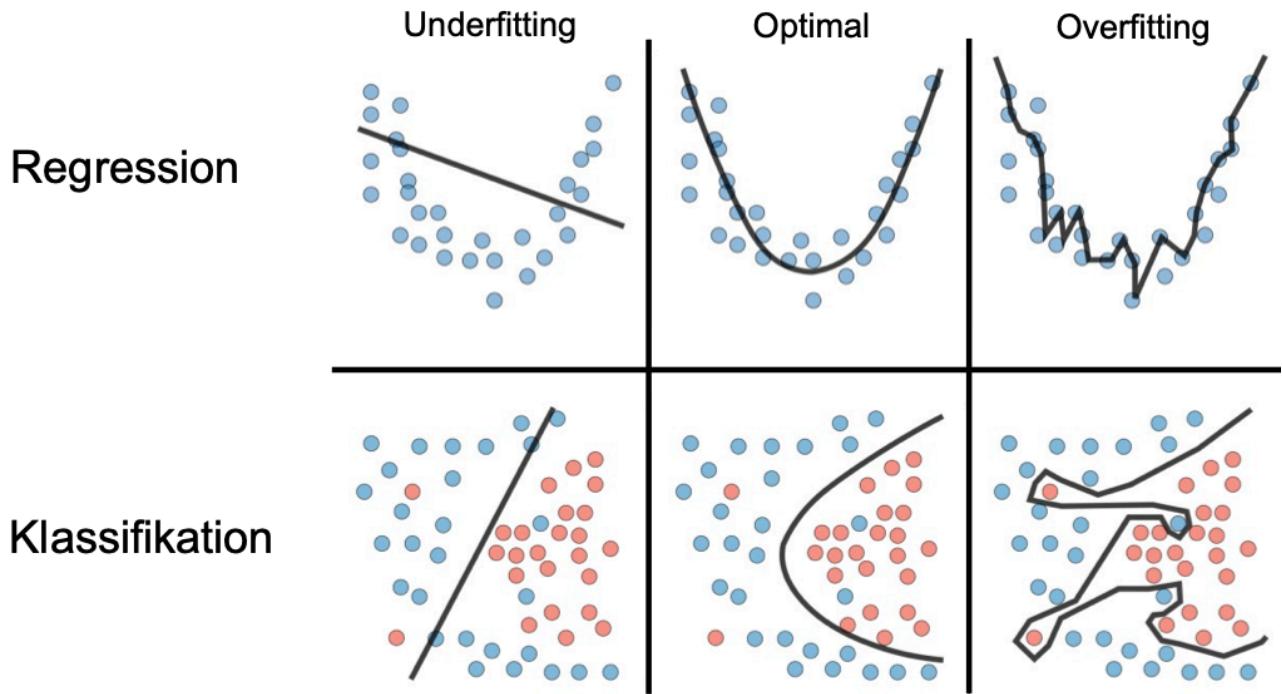
假设空间不含有足够好的目标方程的近似

Repräsentationsproblem

- Der Hypothesenraum enthält keine ausreichend gute Approximationen der Zielfunktion/Konzept etc...
- Das Lernverfahren kann einen gewünschten Approximationsgrad nicht liefern.

学习方法不能提供期望的近似程度

Overtitting - Undertitting



Overfitting:

Modell lernt überwiegend Lernbeispiele auswendig -> Lernfehler fällt, Testfehler steigt 模型主要记忆学习实例

Lösung:

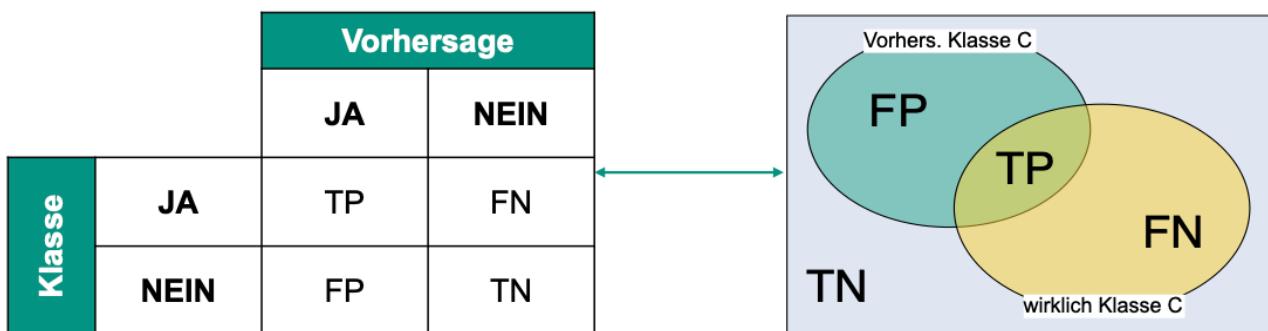
- Repräsentative Beispiele
- Lernprozess durch Verifikationsfehler steuern
- Richtige Wahl und Suche der optimalen Hypothesen

Modellgüte

Unterscheidung von 4 Ergebnisklassen in der Klassifikation:

1. Korrekte Klassifikation von positiven Instanzen: true positive (TP)
2. Korrekte Klassifikation von negativen Instanzen: true negative (TN)
3. Falsche Klassifikation von positiven Instanzen: false positive (FP)
4. Falsche Klassifikation von negativen Instanzen: false negative (FN)

-> Wunschergebnis: wenig FP und FN



- Klassifikationsfehler (sollte klein sein)

$$\frac{errors}{total} = \frac{FP + FN}{TP + FN + FP + TN}$$

■ Güte (= 1 - Fehler)

$$\frac{correct}{total} = \frac{TP + TN}{TP + FN + FP + TN}$$

■ Falsch Positiv Rate (FPR), Falsch Alarm Rate(FA) (sollte klein sein)

$$FPR = \frac{FP}{FP + TN}$$

■ Falsch Negativ Rate (FNR), Miss Rate (MR) (sollte klein sein)

$$FNR = \frac{FN}{TP + FN} = 1 - TPR$$

■ Genauigkeit (Precision) (sollte hoch sein)

$$P = \frac{TP}{TP + FP}$$

■ True Positive Rate (TPR), Positive Rückmeldung (Recall) (sollte hoch sein)

$$TPR = R = \frac{TP}{TP + FN} = 1 - FNR$$

■ F1 – Maß: harmonisches Mittel von Recall (R) und Precision (P) (sollte groß sein)

$$F_1 = \frac{2}{\frac{1}{R} + \frac{1}{P}}$$

- TPR/FPR-Graph (Diagonale -> random performance, Receiver-Operating Characteristic ROC)
- Precision/Recall Graph

Modellverbesserung

- Cross Validation: Teile die Daten in Lern- und Validierungsdaten - bestimme jeweils gute Hypothesen - Berechne jeweilige Metrik - Wiederhole
- Bootstrap:
 - o ziehe mit zurücklegen aus D jeweils $|D|$ Beispiele k-mal
 - o Bestimme jeweils die optimale Hypothese
 - o Wiederhole

- Bestimme den Mittelwert, Varianz, ... der Metriken des Modells um Güte/Stabilität zu analysieren. So kann Modell mit höherer Güte gefunden werden
- Bagging = Bootstrap aggregation:
 - Kombiniere mehrere Lernmaschinen
 - Verwende Bootstrap Prinzip
 - Höhere Güte des resultierenden Modells
- Boosting
 - Gezielt schwache Modelle kombinieren um so ein gutes zu bekommen
- AdaBoost - adaptive Boosting
 - Durch unterschiedliche Gewichtung der Daten werden unterschiedliche Modelle trainiert
 - Diese werden dann wie beim Boosting geschickt zusammengeführt

Vapnik-Chervonenkis (VC) Dimension

VC Dimension sind Maß für Datenkomplexität des Lernens und Maß für Kapazität von lernenden Maschinen

Definition: Die VC Dimension $VC(h_{\text{theta}})$ von H ist gleich der maximalen Anzahl von Datenpunkten (aus einer Menge S) die von H beliebig separiert werden können

Abschätzung des Testfehlers

- Nach Vapnik gilt mit einer gewissen Wahrscheinlichkeit η :

$$J(h_\theta) \leq \hat{J}(h_\theta) + \sqrt{\dots \frac{VC(h_\theta)}{N} \dots}$$

■ wobei:

- | | |
|-----------------------|--|
| ■ $VC(h_\theta)$ | – VC-Dimension der lernenden Maschine |
| ■ N | – Anzahl der Lernbeispiele |
| ■ $\hat{J}(h_\theta)$ | – empirischer Fehler abhängig von $VC(h_\theta)$ und N |
| ■ $J(h_\theta)$ | – realer (zu minimierender) Fehler |

- Lernerfolg ist abhängig von:

- Kapazität der lernenden Maschine (so gering wie nötig)
- Optimierungsmethode (so gut wie möglich)
- Lernbeispiele (repräsentativ, so viele wie möglich)

Lösungsansatz: Structural Risk Minimization

Ziel: Finde eine Lösung für:

$$\min_{H^n} \left(\hat{J}(h_\theta) + \sqrt{\dots \frac{VC(h_\theta)}{N} \dots} \right)$$

Nutze dazu Metalgorithmus: Minimiere Summe (nicht Summanden):

- Strukturiere den Hypothesenraum $H^1 \subset H^2 \subset \dots \subset H^n, VC(h_\theta^i) \leq VC(h_\theta^{i+1})$
- Iteriere über H^i
- Suche jeweils Optimum: das Minimum für $J(h_{\text{theta}})$
- Stoppe wenn Summe minimal

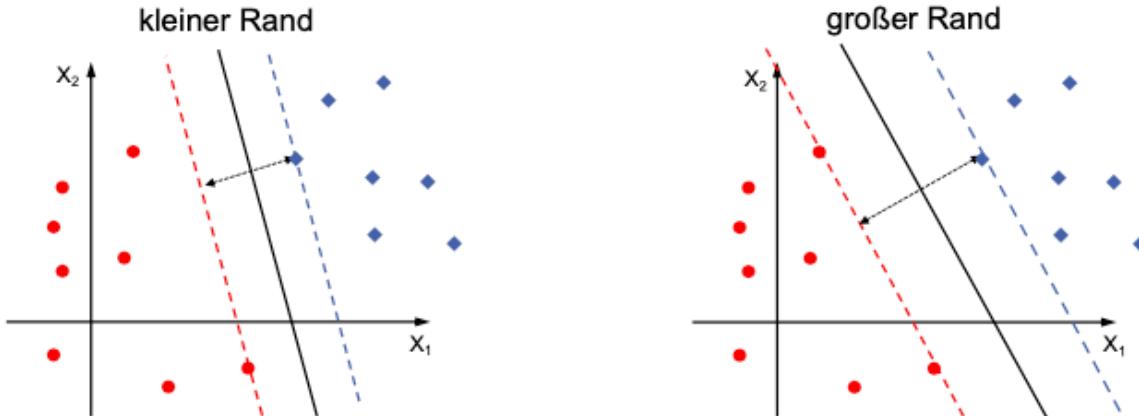
Problem:

- Berechnung der VC Dimension in der Strukturierung schwer, rechenintensiv, für viele Hypothesenräume unmöglich

4 SVM

Montag, 29. Mai 2023 10:22

Lineare Support Vektor Methode



Problem: Klassifikation – Es gibt viele Möglichkeiten die beiden Mengen (rot und blau) zu trennen, aber was ist die optimale Lösung des Problems?

Intuition: Größe des Randes (Margin) bestimmt die Generalisierungsfähigkeit

Lösung: Finde die beste Trenn-Gerade / Hyperebene mit maximalem Rand zu den Klassen

Support Vektoren sind Datenpunkte,

- Die einen direkten Einfluss auf die Position der Hyperebene haben und sie ändern würden, wenn sie entfernt werden
- Die am nächsten an der Hyperebene liegen
- Die ab "schwierigsten" zu klassifizieren sind

Trennhyperebene

■ Finde die **Hyperebene** $\{\vec{x} \in X \mid \vec{w}\vec{x} + b = 0, (\vec{w}, b) \in X \times Y\}$, mit maximalem Rand.

■ **Parameter:** \vec{w}, b

■ **Trainingsdaten:** $\{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$

■ Eingangsdaten $\vec{x}_i \in X$

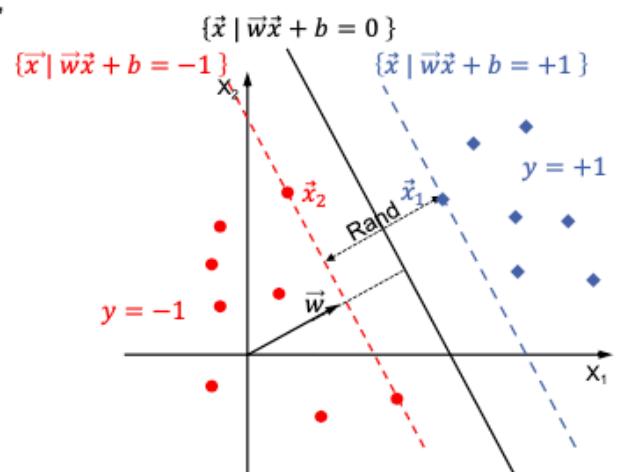
■ Ausgangsdaten $y_i \in Y$

■ **Rand** (= Abstand zw. nächsten Punkten): $\frac{\vec{w}(\vec{x}_1 - \vec{x}_2)}{|\vec{w}|}$

■ **Normierung:** $\vec{w}\vec{x}_1 + b = +1$
 $\vec{w}\vec{x}_2 + b = -1$

■ **Dann gilt:** $\vec{w}(\vec{x}_1 - \vec{x}_2) = 2$

$$\frac{\vec{w}}{|\vec{w}|}(\vec{x}_1 - \vec{x}_2) = \frac{2}{|\vec{w}|}$$



Optimale Hyperebene:

Bedingung für die optimale Hyperebene: $\min_{i=1 \dots n} |\vec{w}\vec{x}_i + b| = 1$

Entscheidungsfunktion:

$$f(\vec{x}) = \vec{w}\vec{x} + b$$

$$n_{\vec{w}, b}(\vec{x}) = \text{sgn}(\vec{w}\vec{x} + b)$$

Optimierungsproblem:

Abstand zwischen 2 Klassen soll maximal sein: $\max\left(\frac{2}{|\vec{w}|}\right) \rightarrow \min(|\vec{w}^2|)$

Unter der Bedingung (umformuliert optimale Hyperebene):

$$y_i(\vec{w}\vec{x}_i + b) \geq 1, i = 1 \dots n$$

Laut Vapnik handelt es sich somit um die Lernmaschine mit der kleinstmöglichen VC-Dimension

Nach Lösen mit Lagrange Minimierung folgt:

Für die meisten Datenpunkte sind die Lagrange Multiplikatoren $a_i = 0$, außer für die wichtigsten Datenpunkte (oben x_1, x_2, x_3), dies sind die Support Vektoren.

Optimale Parameter:

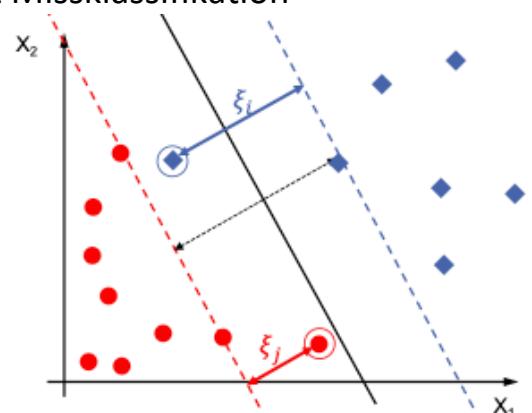
$$\vec{w} = \sum_{i=1}^n \alpha_i y_i \vec{x}_i$$

Generalisierte SVM

Soft Margin SVM

Höhere Generalisierung durch Aufweichung der Randbedingung: Einführung der Slack-Variablen ξ in die Randbedingungen -> erlaubt Missklassifikation

- **Minimiere:** $\min_{\vec{w}, b, \xi_i} \frac{1}{2} |\vec{w}|^2 + C (\sum_{i=1}^n \xi_i)^p$
- **Bedingungen:** $\xi_i \geq 0$ und $y_i(\vec{w}\vec{x}_i + b) \geq 1 - \xi_i$
- **Lösung:** Lagrange - Methode
- **Rolle von C :**
 - Regulierungsparameter
 - C – Groß → Missklassifikationen wird stark bestraft
 - C – Klein → Missklassifikationen wird schwach bestraft



Nichtlineare Kernel-Methoden (Kernel-Trick)

Transformiere Daten in einen anderen (höher-dimensionalen) Raum, wo die Daten linear lösbar sind und löse das Problem dort

Bei Klassifikation: Lineare Trennung im transformierten Raum

Um Rechenzeit und Kompliziertheit zu sparen: **Kernel-Trick**

Statt die komplexe nicht-lineare Transformation und das Skalarprodukt auszurechnen, verwenden wir einen Kernel, der dies implizit tut

$$K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i) \cdot \phi(\vec{x}_j)$$

Es gibt viele unterschiedlich Kernel Funktionen

Versionsraum und strukturelle Risikominimierung bei SVM

Dualität von Merkmal- und Hypothesenraum

Ungleichung bei der Randbedingung kann nach \vec{x}_i im Merkmalsraum und nach \vec{w}_i im Hypothesenraum interpretiert werden

-> Größter Rand heißt gleichzeitig, dass wir nach \vec{w} suchen, das den maximalen Abstand zu allen Hyperebenen der Datenpunkte hat

Somit implementiert die SVM eine Strukturelle Risikominimierung (SRM) implizit in der Lagrange-Optimierung

SVM ähnlich zu neuronalen Netzen?

Gemeinsamkeit: gelernte Gewichte, Kernel, "hidden"-layer

Aber, alles andere ist unterschiedlich: Optimierung, Theoretischer Hintergrund, etc.

Erweiterungen

- Klassifikationen auf k Klassen: Mehrere 2-Klassen-SVMs werden kombiniert (Einer- gegen-Alle, Einer-gegen-Einen, Merfachzugehörigkeit) oder ein gemeinsames Optimierungsverfahren (k-class SVM von Watkins, nicht oft genutzt)
- Gewichtete SVM: verlegt die Hyperebene in Richtung einer Klasse, indem Slack-Variablen unterschiedlich gewichtet werden
- Probabilistische SVM: Abstand der Hyperebene als Wahrscheinlichkeit -> ermöglicht alternatives Training durch Maximierung der Klassifikationswahrscheinlichkeit und des Randes
- Dichte - Träger Schätzung: Sucht eine Funktion, die für eine "kleine" Region, welche die meisten Lernbeispiele enthält, den Wert > 0 und sonst den Wert 0 oder <0 annimt. Ähnlich zu Clustering

Anwendungen

- Kernel Perceptron: nutzt ähnlich wie bei SVM auch Transformierte Räume um dort eine lineare Funktion zu finden
- Nichtlineare Kernel - PCA
- SV- Regression

Konkrete Anwendungen:

Gesichtserkennung, Erkennung von Verkehrszeichen, Segmentierung von Pixeln, Erkennung von Aktionen (-> Gestensteuerung)

Diskussion

Pros:

- Optimale Hyperebene → gute Lernergebnisse
- Finden der optimalen VC-Dimension → korrektes Lernen
- Verarbeitung hochdimensionaler Daten → schnelle Auswertung

- Anwendungsspezifische Kernels (mit Datenverarbeitung)
- Entscheidung wird anhand der Randregionen gefällt
- Viele Anwendungen: Klassifikation, Regression, PCA
- Probabilistische Sicht – wichtig für semi-überwachtes Lernen → siehe ML II
- VS- Beschreibung – gut geeignet für aktives Lernen → siehe ML II

Cons:

- Vorverarbeitung extern (kein „tiefes“ Lernen)
- Finden des optimalen Kernels – „aktuelle“ Forschung
- Parametrisierung des Kernels – „aktuelle“ Forschung
- Speicher und Rechenaufwand (speziell für das Trainieren)
- Anzahl der SV abhängig von Problem und Parameter – aber erweiterte Ansätze möglich

5 Entscheidungsbäume

Dienstag, 30. Mai 2023 12:11

Wofür eignen sich Entscheidungsbäume?

Grundsätzlich:

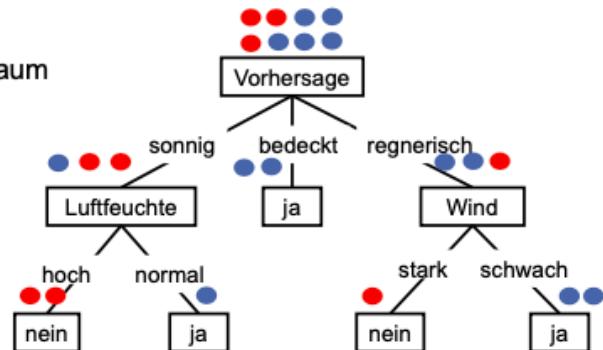
- Instanzen lassen sich als Attribut-Wert Paare beschreiben (kategoriale Daten)
- Zielfunktion besitzt diskrete Ausgabewerte (z.B. Klassen)
- Nicht-parametrisch: Keine Annahme über die zu Grunde liegende Verteilung
- Interpretierbarkeit
- Lernen von Regeln
- Benötigt wenig Rechenressourcen für die Inferenz
- Es ist einfach nicht-lineare Probleme zu lösen
- Daten müssen nicht normalisiert werden und können in verschiedenen Formaten vorliegen

Mit Erweiterungen:

- Anwendbar für Regressionsprobleme (Regressionsbäume)
- Beispieldaten sind möglicherweise verrauscht
- Kontinuierliche Attributwerte möglich (Merkmale)
- Beispieldaten können fehlende Attributwerte enthalten

Aufbau

- **Knoten:** Testet ein Attribut/Merkmal A
 - Z.B. Vorhersage, Luftfeuchte, Wind, ...
- **Wurzelknoten:** Erster Knoten im Entscheidungsbaum
- **Kindknoten:** Nachfolger eines Knotens
- **Elternknoten:** Vorgänger eines Knotens
- **Blatt:** Knoten mit Klassifikationsergebnis Y
 - Z.B. Tennis=ja, Tennis=nein
- **Zweig:** Ergebnis/Attributwert v des Tests von A
 - Z.B. hoch, normal für Luftfeuchte



Attributselektion

Entropie

- Grad an Unsicherheit:
 - Hohe Entropie: Y kommt von einer nahezu gleichverteilten Wahrscheinlichkeitsdichte, die Daten sind also schlecht prädizierbar
 - Niedrige Entropie: Y kommt von einer Wahrscheinlichkeitsdichte mit einer hohen Wahrscheinlichkeit für eine Klasse, relativ zu den anderen. Daten sind gut prädizierbar
- Für Entscheidungsbäume: Maß für die Homogenität (bzgl. ...)

Klassenzugehörigkeit) der aktuellen Datenmenge S

Hier: Diskrete Zufallsvariable $Y \sim p(y)$, wobei $p(y) = \frac{|S_y|}{|S|}$ definiert wird durch die Datenmenge S und die Untermenge S_y der Daten mit der Klasse y

- Entropie für **K Klassen**: $H(S) = -\sum_{i=1}^K p(y_i) \log_2 p(y_i)$
- Entropie für **2 Klassen**(\oplus, \ominus): $H(S) = -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus$

Informationsgewinn

Informationsgewinn $IG(S, A)$: Erwartete Reduzierung der Entropie von S durch die Einsortierung über Attribut A

$$IG(S, A) = H(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} H(S_v)$$

- $V(A)$: Menge aller möglichen Attributwerte v von Attribut A
- S_v : Untermenge von Daten S, für die A den Wert v annimmt

Ziel: sukzessiv die Entropie maximal reduzieren und den Informationsgewinn maximieren -> Baum mit geringer Tiefe

Attributauswahl

Idee: Ein gutes Attribut teilt die Beispiele in Teilmengen auf, die (idealerweise) alle positiv oder alle negativ sind

-> Als Übung wichtig: Beispiel aus Vorlesung rechnen Folien 15-18

Aufbau von Entscheidungsbäumen - ID3 Algorithmus

- Top-Down: Entscheidungsbaum wird vom Wurzelknoten aus aufgebaut
- Greedy: Attribut mit höchstem Informationsgewinn wird gewählt
- Garantiert keine optimale Lösung
- Neigt zur Überanpassung (Overfitting), insb bei verrauschten Daten

ID3(Beispiele, Ziel_Attribut, Attribute)

Erstelle einen Wurzelknoten für den Baum

Wenn (alle Beispiele positiv sind), **Return** (Knoten mit *Label*=+)

Wenn (alle Beispiele negativ sind), **Return** (Knoten mit *Label*=-)

Wenn (Attribute==Leere Menge), **Return** (Knoten mit häufigstem Label in Beispiele)

A = Attribut, welches den höchsten Informationsgewinn auf Beispiele hat

Attribut des Knotens = *A*

Für alle Attributwerte v_i des Attributes *A*:

Füge einen neuen Zweig zum Knoten hinzu, welcher dem Attribut v_i entspricht.

Beispiele(v_i) = Untermenge der Beispiele, welche den Attributwert v_i für *A* haben

Wenn (Beispiele(v_i) == Leere Menge):

Füge einen Blattknoten für diesen Zweig hinzu mit dem häufigsten Label in Beispiele

Dann:

Unter diesem Zweig, füge den Baum **ID3(Beispiele(v_i), Ziel_Attribut, Attribute - {A})** hinzu

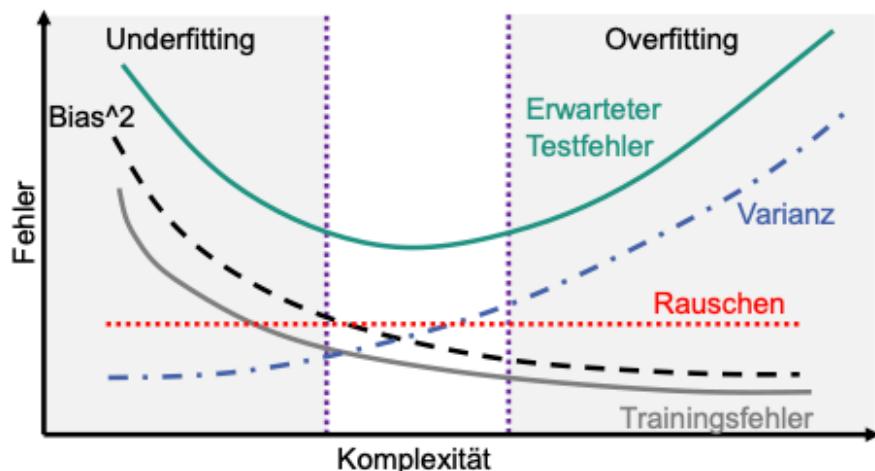
Overfitting

Jeder Zweig wächst solange, bis die Trainingsbeispiele perfekt klassifiziert werden -> oft Overfitting

Es sollte also eine kurze Hypothese vor einer langen bevorzugt werden

Bias-Variance Tradeoff

$$\mathbb{E}_{x,y,D}[(h_D(x) - y)^2] = \underbrace{\mathbb{E}_{x,D}[(h_D(x) - \bar{h}(x))^2]}_{\text{Erwarteter Testfehler}} + \underbrace{\mathbb{E}_x[(\bar{h}(x) - \bar{y}(x))^2]}_{\text{Varianz}} + \underbrace{\mathbb{E}_{x,y}[(\bar{y}(x) - y)^2]}_{\text{Rauschen}}$$



Varianz: Erfasst, wie sehr sich der Klassifikator verändert, wenn mit anderen Trainingsdaten trainiert wird.

- D.h. wie stark lernen wir einen bestimmten Trainingsdatensatz auswendig
- Wie groß ist die Differenz vom aktuellen Klassifikator entfernt vom durchschnittlichen Klassifikator?

Bias: Wie hoch ist der inhärente Fehler, den der Klassifikator selbst bei unendlich vielen Trainingsdaten aufweist?

- Kommt von Modellannahmen, die bestimmte Lösungen präferieren, z.B. ein linearer Klassifikator klassifiziert nur linear trennbare Daten perfekt

Rauschen: Wie groß ist das dateninhärente Rauschen?

- Misst die Mehrdeutigkeit aufgrund der Datenverteilung und Merkmalsrepräsentation
- Dieses Rauschen ist nicht reduzierbar, es ist ein Teil der Daten

Overfitting reduzieren durch:

- Early Stopping
- Pruning
- Bagging
- Random Forests

Early Stopping

Intuition: Stoppe das Baumwachstum bevor

es zu Überanpassung kommt

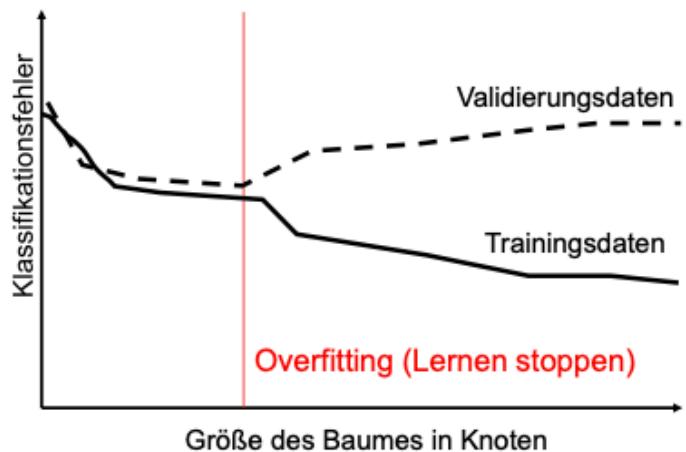
Idee: Validierungsfehler muss um ϵ pro Iteration sinken, ansonsten stoppen wir

Pros:

- Einfach zu implementieren
- In der Praxis oft verwendet

Cons:

- Zu kurzsichtig: Validierungsfehler könnte im aktuellen Schritt weniger als ϵ fallen, aber im nächsten Schritt mehr als ϵ



Pruning 修剪

- **Intuition:** Validierungsdaten werden benutzt um zu entscheiden welche Blätter des Entscheidungsbaums entfernt werden. Wir hören mit dem Beschneiden auf, wenn der Validierungsfehler steigt.
- **Pruning:** Entferne den Teilbaum und ersetze ihn durch einen Blattknoten mit der meistvorhandenen Klasse
- **Resultat:** Liefert die kleinste Variante des akkuratesten Unterbaums

Pruning(Trainierter_Baum, Validierungsdaten)

Schleife (So lange der Validierungsfehler fällt)

Finde den Knoten, der den Klassifikationsfehler auf den Validierungsdaten am meisten reduziert und entferne ihn aus Trainierter_Baum.

Return Trainierter_Baum



Ensemble Learning

Lerne mehrere schwache Modelle (mit unterschiedlichen Maschinen) und kombiniere sie zu einem starken Modell

- + Bessere Prädiktionsgüte
- + Bessere Robustheit (Overfitting)
- Zeit- und Rechenaufwändig
- Modelle schwer erklärbar

Bagging

1. **Bootstrapping:** Erzeuge aus Datensatz D, m Datensätze D_1 bis D_m mit derselben Anzahl an Daten durch Sampling mit Zurücklegen
2. **Training:** Trainiere auf jedem Datensatz ein neues Modell h_{D1} bis h_{Dm}
3. **Aggregation:** Kombiniere die Modelle z.B. mit Mittelwert oder Mehrheitsentscheid

Finale Hypothese ist somit der Durchschnitt über alle m Hypothesen, somit kann die Varianz verringert werden ohne Bias zu erhöhen. Erinnerung:

$$\text{Testfehler} = \text{Bias}^2 + \mathbb{E}_{x,D} \left[(h_D(x) - \bar{h}(x))^2 \right] + \text{Rauschen}$$

Pros

- Reduziert Varianz und hat damit einen starken Effekt auf Modelle mit hoher Varianz
- **Unsicherheit:** Durch Bagging erhält man nicht nur den Erwartungswert, sondern auch die Varianz der Prädiktion
- **Out-of-bag Fehler:** Lerndaten müssen nicht in Trainings- und Testdaten geteilt werden. Der Testfehler kann direkt auf den Trainingsdaten berechnet werden
- **Parallelisierbarkeit**

Cons

- **Ressourcenverbrauch** steigt um Faktor m
- Varianz kann nicht komplett reduziert werden, da **Bootstrapping Samples nicht I.I.D** sind

Random Forests

- Speziell für Entscheidungsbäume
 - Problem bei Bagging: Modelle können korrelieren, Bäume können gleich sein
 - Daher: Wähle zufällig eine Untermenge an k Attributen ($k < d$, $k = \sqrt{d}$) für jeden Split aus ohne Zurücklegen -> somit sind die Bäume unterschiedlich und korrelieren nicht
 - Fehler in den einzelnen Bäumen heben sich durch Mittelwert auf
1. **Bootstrapping:** Erzeuge aus Datensatz D , m Datensätze D_1 bis D_m mit derselben Anzahl an Daten durch Sampling mit Zurücklegen
 2. **Training:** Trainiere auf jedem Datensatz einen neuen Baum h_{D1} bis h_{Dm} . Aber:
 - Vor jedem Split, sample eine Untermenge von $k < d$ Attributen (ohne Zurücklegen) und berücksichtige nur diese
 3. **Aggregation:** Kombiniere die Modelle mit Mehrheitsentscheid

Erweiterungen

Attribute mit vielen Werten

Problem: Attribute mit vielen Werten werden gegenüber Attributen mit wenigen Werten bevorzugt

Lösung: Bestrafung von Attributen

$$\text{GewinnAnteil}(S, A) = \frac{\text{IG}(S, A)}{\text{SplitInformation}(S, A)}$$

$$\text{SplitInformation}(S, A) = - \sum_{v \in V(A)} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$$

Wobei:

- $V(A)$: Menge aller möglichen Attributwerte von A
- S_v : Untermenge von S , für die A den Wert v annimmt

Kontinuierliche Attributwerte

Problem: Wahl eines Schwellenwertes bei kontinuierlichen Werten

Lösung: Auswahl über Informationsgewinn:

- Sortierung der Beispiele gemäß ihrer Werte
- Optimaler Schwellenwert liegt in der Mitte zwischen zwei benachbarten Beispielen mit unterschiedlichen Klassenzugehörigkeiten

Unbekannte Attributwerte

Wenn Attributwerte fehlen:

Sortiere alle Beispiele wie gewohnt, fehlende Attribute bekommen heuristische Werte:

- Häufigster Attributwert der Beispiele oder
- Häufigster Attributwert der Beispiele der gleichen Klasse oder
- Jeder Wert v_i mit Wsh. $p_i \rightarrow$ Verteilung gemäß p_i

Attribute mit Kosten

Finde korrekten Entscheidungsbaum mit niedrigen erwarteten Kosten:

$$\frac{IG(S, A)^2}{\text{Kosten}(A)}$$

■ oder

$$\frac{2^{IG(S, A)} - 1}{(\text{Kosten}(A) + 1)^w}$$

Wobei:

■ die $w \in [0,1]$ Gewichtung (Bedeutung) der Kosten angibt.

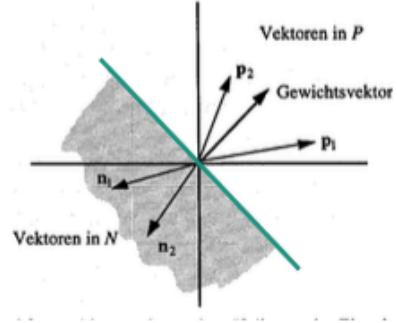
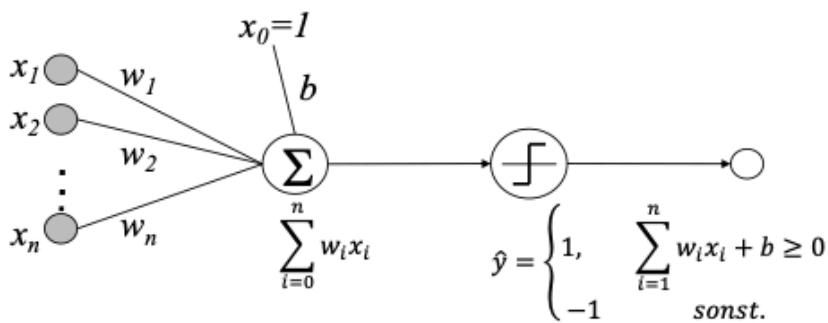
6 Neuronale Netze Grundlagen

Mittwoch, 31. Mai 2023 09:37

- **Ziel:** Eigenschaften von Gehirn mit denen des Computers kombinieren
- **Im Gehirn:** große Anzahl Neuronen mit vielen Verbindungen
- **Dadurch:** hochparallele Berechnung, verteilte Repräsentation von Wissen, Guter Umgang mit Fehlern, niedriger Energieverbrauch
- **Gesuchte, gemeinsame Eigenschaften eines NN:** Parallel Informationsverarbeitung, Lernfähigkeit, Generalisierungsfähigkeit, Fehlertoleranz

Perzeptron

Anlehnung an das Funktionsprinzip aus der Natur



Ein Perzeptron realisiert eine Trennhyperebene (in \mathbb{R}^2 eine Gerade)

Die Gewichte definieren diese Trennhyperebenen (sie stellen die Normale dar)

Gegeben Positive und Negative Daten (P, N) erfolgt eine Entscheidung durch gewichtete Summe → Skalarprodukt mit dem Nomalenvektor

Erweiterung der Dimension durch x_0

Lernen = Anpassen der Gewichte → Gesucht wird die beste Trennhyperebene

Probleme:

- Sehr langsame Anpassung, Lösung: Normierung
- Antiparallele Vektoren, Lösung: Gradientenabstieg -> benötigt differenzierbare Aktivierungsfunktion
- Niedrige Kapazität von einem Perzeptron (nur lineare Funktionen möglich), Lösung: Kombination von "Perzeptronen" (Neuronen)

Erinnerung: Kernel-Trick aus VL 3

- Lineare Trennung im transformierten Raum führt zu komplexer Trennung im Ursprungsraum
- Um allerdings keine Berechnungen im transformierten Raum machen zu müssen, lässt sich idealerweise alles auf ein Skalarprodukt reduzieren und

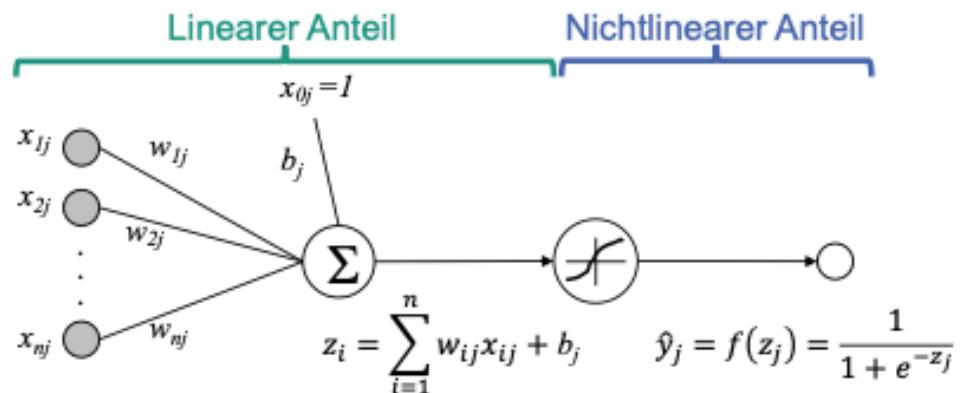
durch einen Kernel ersetzen

- Mit einer Kernel-Funktion muss die Transformation nicht explizit bekannt sein

Multi-Layer Neural Networks (MLNN)

- Mehrere versteckte (innere) Schichten
- Lernen mit Backpropagation Algorithmus
- Neuronen mit nichtlinearer Aktivierungsfunktion

Neuronen



- x_{ij} = i -te Eingabe des Neurons j
- w_{ij} = Gewicht zwischen Neuron i und Neuron j
- z_i = $\sum_i w_{ij}x_{ij}$ Propagierungsfunktion
- $f(x)$ = Aktivierungsfunktion
- \hat{y}_j = Ausgabe des Neurons j
- y_j = Zielausgabe des Ausgabeneurons j

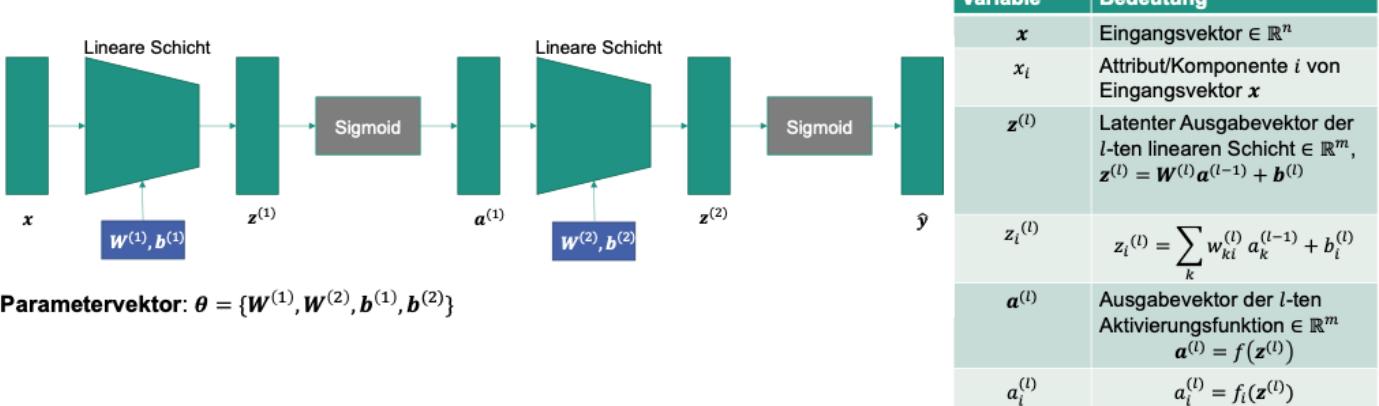
Warum wird der nichtlineare Teil benötigt?

-> Ohne (nichtlineare) Aktivierungsfunktion kollabieren neuronale Netze aus mehreren Schichten, immer zu linearen Modellen

Nichtlineare Aktivierungsfunktionen:

Name	Funktion	Ableitung	Vorteile	Nachteile
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$\frac{\partial f}{\partial x} = f(x)(1 - f(x))$	<ul style="list-style-type: none"> - Nichtlineares Funktionsverhalten - Approximiert eine Wahrscheinlichkeit 	<ul style="list-style-type: none"> - Nicht um 0 zentriert - Sättigungsverhalten - Ableitung verschwindet in Sättigungsbe

				reichen - Rechenintensiv
Tangens Hyperbolicus	$f(x) = \tanh(x)$	$\frac{\partial f}{\partial x} = (1 + f(x))(1 - f(x))$	- Zentriert um 0 - Normalisiert um [-1,1]	- Sättigungsverhalten - Ableitung verschwindet in Sättigungsbe reichen - Rechenintensiv
ReLU	$f(x) = \max(0, x)$	$\frac{\partial f}{\partial x} = \begin{cases} 1, & x > 0 \\ 0, & \text{sonst} \end{cases}$	- Meistbenutzt - Einfach zu berechnen - Kein Sättigungsver halten für $x > 0$	- Nicht um 0 zentriert - Ableitung verschwindet für $x < 0$
LeakyReLU	$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$	$\frac{\partial f}{\partial x} = \begin{cases} 1, & x > 0 \\ \alpha, & \text{sonst} \end{cases}$	- Kein Sättigungsver halten für $\alpha > 0$ - Einfach zu berechnen - Nahe an einer um 0 zentrierten Funktion	- Zusätzlicher Hyperparameter α
Softmax	$f(\mathbf{x}) = \begin{bmatrix} e^{x_1} \\ \vdots \\ e^{x_k} \\ \vdots \\ e^{x_n} \end{bmatrix} / \sum_{j=0}^k e^{x_j}$ $\therefore f([x_0]) = \begin{bmatrix} e^x \\ e^x + e^0 \\ \vdots \\ e^0 \\ \vdots \\ e^x + e^0 \end{bmatrix} = \begin{bmatrix} f_\sigma(x) \\ 1 - f_\sigma(x) \end{bmatrix}$	$\frac{\partial f_i(\mathbf{x})}{\partial x_j} = \begin{cases} f_i(\mathbf{x})(1 - f_i(\mathbf{x})), & i = j \\ -f_i(\mathbf{x})f_j(\mathbf{x}), & i \neq j \end{cases}$	- Multi-dimensionale Verallgemeinerung der Sigmoidfunkt. - Diskrete Wahrscheinlichkeitsdichte - Einfache Ableitung	- Sättigungsverhalten - Ableitung verschwindet in Sättigungsbe reichen - Rechenintensiv



Gradientenabstieg

Ziel: Minimum der Lossfunction finden.

Problem: Berechnung und Multiplikation der Ableitungsmatrizen extrem rechenintensiv

Lösung: Backpropagation

Backpropagation

Effiziente Berechnung der Ableitungen durch Rückpropagierung

Vorgaben:

- Menge D von N Trainingsbeispielen als Eingabe-/Ausgabevektor
- Lernrate α (alpha)
- Netztopologie

Ziel: Finde eine Parameterbelegung (θ), die D korrekt wiedergibt

Vorgehen:

- Initialisieren der Gewichte mit kleinen zufälligen Werten (siehe später)
- Wiederhole
 - Bestimmen der Netzausgaben \hat{y} für D
 - Bestimmen des Ausgabefehlers/Loss (bzgl. Sollausgaben y)
 - Sukzessives Rückpropagieren des Fehlers auf die einzelnen Neuronen
 - Anpassung der Gewichtsbelegungen um z.B.

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_{\theta} J(\theta_k)$$
- solange ein gewähltes Abbruchkriterien nicht erfüllt ist

Rückpropagieren:

Wie kann $\frac{\partial J(\theta)}{\partial W^{(l)}}$ effizient berechnet werden?

$$\frac{\partial J(\theta)}{\partial W^{(l)}} = \frac{\partial z^{(l)}}{\partial W^{(l)}} \underbrace{\frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial z^{(l+1)}}{\partial a^{(l)}} \cdots \frac{\partial z^{(end)}}{\partial a^{(l)}}}_{\text{Backpropagation}}$$

Backward Pass:

- Initialisiere: $\delta = \frac{\partial J(\theta)}{\partial z^{(end)}}$
- Für jede Schicht l gibt es mehrere Funktionen f linear bzw. z.B. Sigmoid Aktivierung mit den Parametern $\theta_f^{(l)}$ und input x_f , daher sukzessive rückwärts Berechnung durch:

$$\frac{\partial J(\theta)}{\partial \theta_f^{(l)}} \leftarrow \frac{\partial f}{\partial \theta_f^{(l)}} \delta$$
$$\delta \leftarrow \frac{\partial f}{\partial x_f^{(l)}} \delta$$

Stochastischer Gradientenabstieg

Epochen Lernen beim Gradientenabstieg:

- Mittelung der Gewichtsänderung über alle Beispiele
- Anpassung nachdem alle Beispiele propagiert wurden
- Teils sehr große Datensätze (ImageNet: 1,5 Millionen annotierte Bilder)

Nachteil:

- Berechnung sehr langsam
- Speicherintensiv
- Häufig Overfitting

Lösung:

- Stochastische Auswahl der Lerndaten (aktuell mit "mini batch learning" (Lernen auf Teilmengen) alternativ "pattern learning" (Lernen auf Einzeldaten))

7 Neuronale Netze - Hyperparameter

Donnerstag, 1. Juni 2023 09:23

Parameter: Gewichte und Biase der Neuronen

Hyperparameter: Einstellungen des Trainingsalgorithmus, bspw. Lernrate, Anzahl Neuronen

-> Gut gewählte Hyperparameter sind wichtig für Erfolg des Modells

Optimierungsverfahren

Um das globale Minimum zu finden, können Optimierungsverfahren helfen

Newton Methode

- Informationen der zweiten Ableitung werden berücksichtigt
- Rechenintensiv, daher nicht ideal (Berechnung Hesse Matrix $O(n^2)$)

Momentum

- Heuristische Methode um Information über Gradient zu berücksichtigen
- Vorherige Richtung des Gradienten wird berücksichtigt
- Intuition: Wenn aufeinanderfolgende Gradientenschritte...
 - o In unterschiedliche Richtungen zeigen, soll die Richtung, die nicht übereinstimmt gestrichen werden
 - o In ähnliche Richtung zeigen, soll schneller in die Richtung gegangen werden

Aktualisierungsregel (GA) In Gradientenabstieg: Mit Momentum:

$$\theta_{k+1} \leftarrow \theta_k + v_k \quad v_k = -\alpha \nabla_{\theta} J(\theta_k) \quad v_k = -\alpha \nabla_{\theta} J(\theta_k) + \beta v_{k-1}$$



"Koeffizient des Momentums" "vorherige Richtung"

RMS Prop (Root Mean Square Propogation)

- Angepasste Lernraten für jeden Parameter
- Intuition: Bei kleinen Gradienten, wird Lernrate größer -> hilft bei Plateaus; bei steilen Abstiegen, wird Lernrate kleiner
- Für einen Parameter wird ein gleitender Schätzer für quadrierten Gradienten mitgeführt

Adam (Adaptive Moment Estimation)

- Angepasste Lernraten für jeden Parameter
- Für jeden Parameter werden gleitende Schätzer für Mittelwert und Varianz

- für jeden Parameter werden gleichermaßen Schätzungen für Mittelwert und Varianz mitgeführt

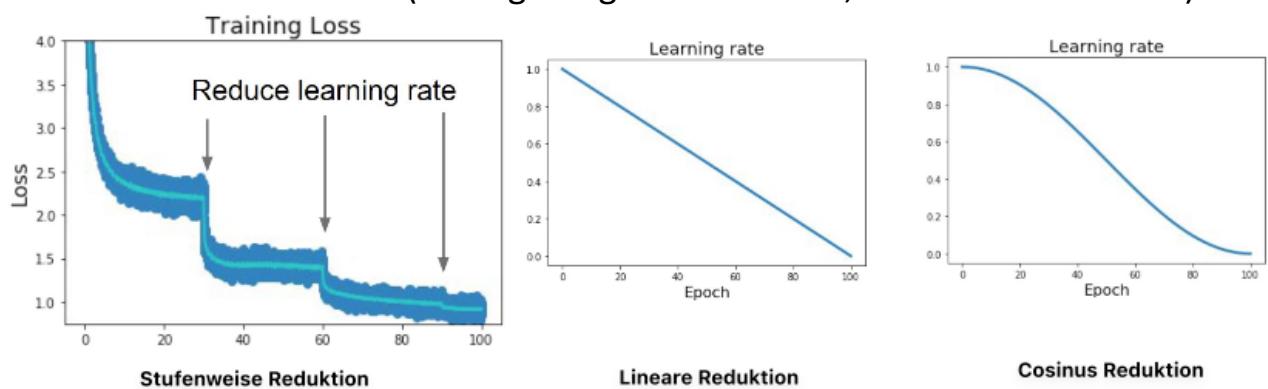
Lernrate

Größe der Lernrate ist entscheidender Faktor für das Training

Häufig: unterschiedliche Werte für die Lernrate testen mit Werten zwischen 0.0001 und 0.1

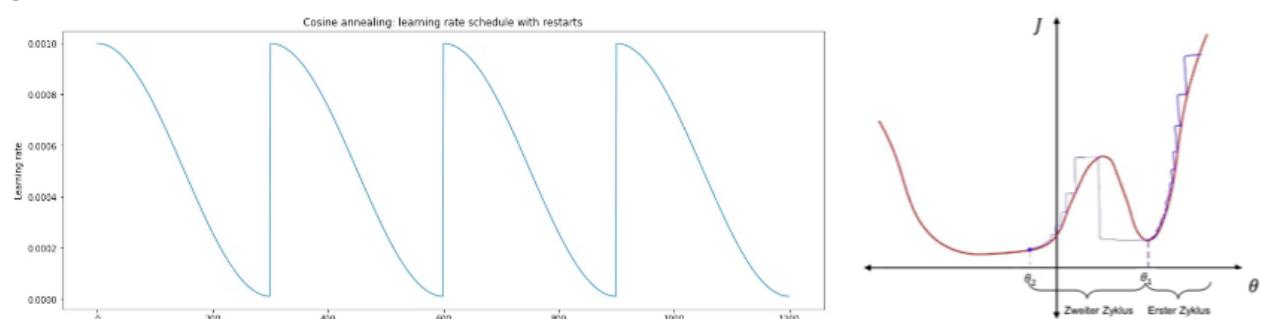
Dynamische Lernraten

- Idee: Reduzierung der Lernrate (learning rate decay)
- Nutzung der Vorteile von hohen und niedrigen Lernraten
 - o Hohe Lernrate (anfangs): schnell gute Parameter lernen
 - o Niedrige Lernrate (zum Ende): Minimum finden (nicht überspringen)
- Optionen:
 - Stufenweise Reduktion der Lernrate nach n Epochen
 - Lineare Reduktion
 - Cosinus Reduktion (anfangs langer hoch lassen, zum Ende abflachen)



Zyklische Lernraten

- Idee: um lokalen Minima zu entkommen, zyklisch Lernrate wieder erhöhen
- Verwende mehrere Zyklen von Decay
- Praxis: Am Beginn jedes Zyklus verschlechtert sich der Loss, da aus Minimum gesprungen wird, Langfristig allerdings Verbesserung, da bessere Minima gefunden werden



=> Auch bei Adam erzeugen dynamische Lernraten bspw. mit einer Cosinus

Reaktion bessere Ergebnisse

Regularisierung

- Zentrales Problem: Overfitting vermeiden -> Modelle trainieren, die auf ungesesehenen Daten gute Ergebnisse liefern
- Regularisierung: Methoden mit dem Ziel Generalisierungsfehler zu reduzieren (evtl. höherer Trainingsfehler)

Early Stopping

- Training stoppen, wenn Validierungsfehler steigt
- Problem: es gibt keine "glatten" Fehlerkurven
- 1. Lösung: Training stoppen, wenn in Intervallen von n Trainingszyklen der Validierungsfehler p mal aufeinanderfolgend größer wird
- 2. Lösung: Training bis Ende laufen lassen, anschließend Parameter mit geringsten Validierungsfehler nutzen

Parameter Regularisierung (Weight Decay)

- Große Parameterwerte, können Anzeichen für Überanpassung sein
- Daher: bestrafte große Parameterwerte durch zusätzliche Parameternorm
- Normen:
 - L^1 Regularisierung:
 - Aufsummierter Betrag der w_i
 - Tendenz die Koeffizienten w_i auf 0 zu setzen
 - + Unwichtige Merkmale werden ignoriert
 - Repräsentation verliert Redundanzen
 - L^2 Regularisierung:
 - L_2 -Norm über alle Gewichte
 - Koeffizienten werden gleichmäßig reduziert
 - + Reduziert Multikollinearität
 - + Redundanzen bleiben erhalten
 - Koeffizienten w_i werden nie 0

Dropout

- Zufällige Deaktivierung von Neuronen während des Trainings
- Idee: Dadurch wird auf Subnetzen trainiert, welche zu einem großen Netz gehören -> Bagging wird simuliert (sonst zu aufwendig k Modelle zu trainieren und zu aggregieren)
- Verhindert Angleichung der Neuronen und fördert eine redundante Repräsentation
- Problem: Dropout macht Ausgabe zufällig. Es muss der Erwartungswert berechnet werden (was sehr komplex ist)

- Stattdessen: Multipliziere während der Testzeit mit der Dropoutwahrscheinlichkeit p

Datenaugmentierung

- Methode um Datensatz künstlich, aber realistisch zu erhöhen
- Durch größeren Datensatz ist Generalisierung wahrscheinlicher
- Datenaugmentierung muss aber auf jeweilige Domäne abgestimmt werden und Sinn der Daten erhalten

Label Smoothing

- Netzwerke die Eingabedaten auswendig lernen, sind oft sehr selbstsicher
- Generalisierte Netzwerke sind meist weniger sicher
- Lösung: Bestrafte Netzwerk wenn es sich zu sicher ist, indem Label fehlerhafte Klasse hinzugefügt wird (Bsp.: Statt "Hund", jetzt 95% "Hund" und 5% "Katze" -> Netzwerk erhält höheren Loss wenn es zu 100% "Hund" prädiziert)

Parameterinitialisierung

- Training kann stark von der Initialisierung der Startwerte abhängen
- Problem:
 - Exploding Gradient: Bei zu großen initialen Gewichten, können die Werte von $a^{(l)}$ exponentiell ansteigen und zu großen Gradienten und somit instabilem Training führen
 - Vanishing Gradient: Bei zu kleinen initialen Gewichten, können die Werten von $a^{(l)}$ exponentiell sinken und zu kleinen Gradienten und somit zum Stoppen des Trainings führen
 - Initialisierung ist abhängig von Aktivierungsfunktion und der Netzarchitektur
- Ziel: Bestimmte Eigenschaften der Aktivierungsverteilung sollen die ganze Zeit beibehalten werden
- Gewünschte Eigenschaften:
 - Erinnerung - Schicht: $a^{(l)} = f(z^{(l)})$, $z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$
 - Symmetrien brechen zwischen Neuronen
 - Aktivierungsfunktionen in "Arbeitsbereich" halten
 - $E[a^{[l-1]}] = E[a^{[l]}] \approx 0$
 - $Var(a^{[l-1]}) = Var(a^{[l]}) > 0$

Parameterinitialisierung - Xavier

- Bias mit 0 initialisiert
- Gewichte werden normalverteilt initialisiert: $w_{ij}^{(l)} \sim \mathcal{N}\left(0, \frac{1}{n_{in}^{(l)}}\right)$
- Nicht für alle Aktivierungsfunktionen geeignet (-> tanh() ist ok)

- NICHT FÜR ALLE AKTIVIERUNGSFUNKTIONEN GEGENÜBERSTELLBAR

Parameterinitialisierung - Kaiming He

- Berücksichtigt explizit Eigenschaften der ReLU
- Unterschied:

$$w_{ij}^{(l)} \sim \mathcal{N}\left(0, \frac{2}{n_{in}^{(l)}}\right)$$

8 CNN

Sonntag, 4. Juni 2023 19:11

Motivation

Probleme:

- Bilder sind eine numerische Repräsentation von Farbintensitäten
- Es können zur Bilderkennung keine Regeln vom Menschen geschrieben werden
- Extrem hohe Anzahl an Gewichten schon für kleine Bilder nötig (bei normalen NN)
- Neuronale Netze sind Translation-invariant

Lösung:

- Nutze kleineres Neuronale Netz und lasse es über Bild gleiten
- > Convolutional Neural Networks

Convolution

Convolution = Faltung, in CNNs: Diskrete Faltungen

Vom (Graustufen-) Bild zur Feature Map mithilfe von Kernel:

■ Image $I(x, y) \in \mathbb{R}^{m_1 \times m_2}$



■ Kernel $K(x, y) \in \mathbb{R}^{n \times n}$

0	1	0
1	-4	1
0	1	0

Convolutional Kernel
 $n = 3$

■ Convolution $K * I(x, y)$

$$= \sum_{u=1}^n \sum_{v=1}^n K(u, v) I(x - u + a, y - v + a)$$

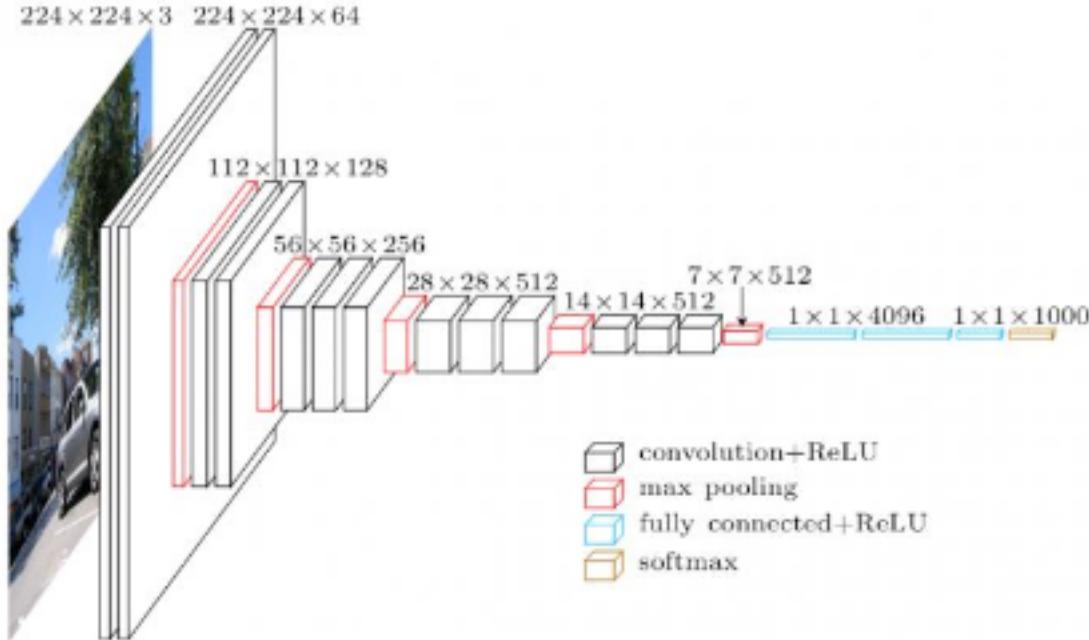


-> Um RGB-Bilder zu lernen werden Convolutional Filter genutzt:

Ein Convolutional Filter beinhaltet einen Bias und mehrere Kernel (für RGB: 3) und erstellt daraus eine eindimensionale Featuremap. Convolutional Filter sind

also Feature Detektoren

Convolutional Neural Networks



Convolutional layer

- Besteht aus mehreren Convolutional Filtern, wobei k Filter k Feature Maps erstellen
- Input: Vorherige Feature Map: $input\ size = m * m * c$
- Output: Neue Feature Map:
$$output\ size = m_{new} * m_{new} * k, \text{ mit } k = \text{Anzahl Filter und}$$
$$m_{new} = \frac{m - n}{s} + 1, \quad s = stride$$

Stride

- Schrittweite des Kernel, in beide Richtungen
- Größer Stride verringert Auflösung der Feature Map

Padding

- Wenn Kernel größer 1×1 , werden Pixel am Rand nicht so oft vom Kernel berücksichtigt -> Auflösung der Feature Map reduziert sich
- Lösung: Auflösung der Input Feature Map erhöhen:
 - Zero Padding: Oen am Rand ergänzen
 - Reflect Padding
 - Circular Padding

Pooling

- $p \times p$ Bildbereiche werden in Feature Maps zusammengefasst
- Strategien: Max Pooling, Average Pooling
 - + lokale Translationsinvarianz
 - + Datenreduzierung (Reduzierung der Auflösung)
 - + Kein lernbarer Parameter

Aktivierungsfunktion

- Erzeugen Nichtlinearität
- Meist verwendet: RELU: $f(x) = \max(0, x)$, neuerdings auch: SiLU: $f(x) = x * \text{sgn}(x)$

Klassifikation

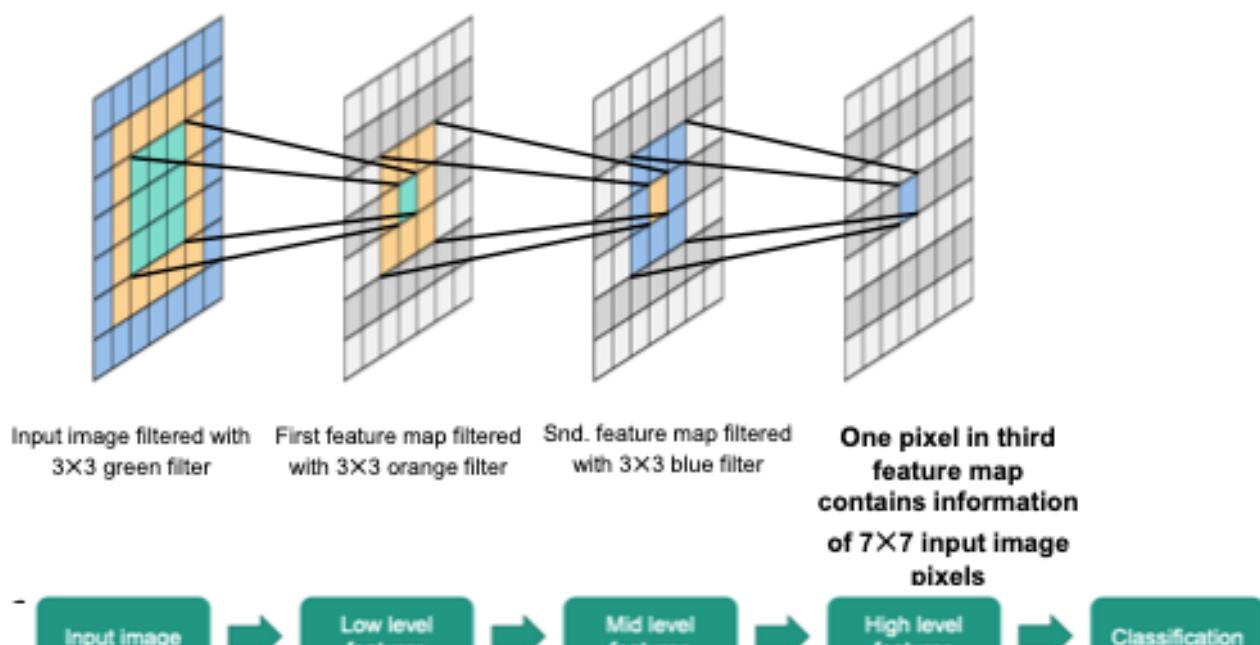
- Am Ende eines CNNs: Fully connected layer
- Anzahl Klassen = Anzahl Neuronen
- Um unterschiedliche Bildauflösungen verwenden zu können, wird mittlerweile die "Channelwise Global Average Pool"-Methode verwendet
- Danach wird mit Softmax als letzte Aktivierungsfunktion klassifiziert

Initialisierung der Gewichte

1. Durch He/Kaiming oder Xavier Gewichte initialisieren
2. Oder Gewichte von bereits trainierten Modellen verwenden (Transfer Learning)

Rezeptives Feld von Convolutions

- Wieviel Informationen des originalen Bildes in einem Feature Map Pixel sind





- Features in einem Level basieren auf Features des vorherigen Levels
- Es haben sich in der Praxis 3x3 conv. Filter durchgesetzt

Wichtige Architekturen

1989: LeNet

Warum haben sich CNNs jetzt erst durchgesetzt?

- Große gelabelte Datensätze
- Training auf GPU mit Parallelisation (GPU optimiert für Matrixoperationen)

Neue Architekturen:

- VGG 16 (2014), SENET (2018), EfficientNet (2019)
- ResNet(2015): Durch Skip connections wird nun die Änderung der vorhandenen Feature Maps gelernt. Deutlich besser, da Verlustfunktion (loss landscape) geglättet ist. Und es können mehrere Layer hintereinander geschaltet werden. Tiefere CNNs -> Bessere Performance

9 Unüberwachtes Lernen

Montag, 5. Juni 2023 20:19

Motivation

- Erkennung von Zusammenhängen/Mustern in Daten
- Semi-supervised learning: Ergänzung von Labels durch unüberwachtes Lernen
- Unterstützung anderer Lernansätze bspw Deep Reinforcement Learning

Grundidee

- Ausnutzen von Ähnlichkeiten in Trainingsdaten, um
 - Klassen / Ballungen zu erschließen
 - Wesentliche Charakteristika zu verwenden

Clustering

K-means-Clustering

- Teilt Datenmenge in eine a-priori festgelegte Anzahl von Ballungen ein
- **Idee:**
 - Definieren eines Mittelpunkts für jedes Cluster
 - Iterative Anpassung/Verbesserung bzgl. Daten die zu dem Cluster gehören und Cluster-Mittelpunkt
 - Optimalitätskriterium: Minimierung der Abstände aller Datenpunkte von ihrem Cluster-Mittelpunkt
- Gegeben: k (Anzahl gesuchte Cluster), x (unklassifizierte Trainingsbeispiele mit d Attributen)
- **Gesucht:** Einteilung der Trainingsmenge in Cluster X_1, \dots, X_k , s.d. Minimum des Abstandes der Daten von den Zentren erreicht ist
- **Algorithmus:**
 - Platziere k Punkte c_j im d -dimensionalen Raum als initiale Mittelpunkte der Ballungen
 - Wiederhole bis keine Änderungen mehr:
 - Klassifiziere jedes x_i gemäß dem nächsten c_j
 - Berechne Mittelpunkte c_j der Cluster neu
- **Bewertung:**
 - Ergebnis stark von initialer Belegung abhängig (Lage und Anzahl Cluster)
 - Resultate hängen von Metrik ab (Curse of dimensionality)
 - Nicht deterministisch, Ergebnis hängt von Reihenfolge der Abarbeitung ab

Fuzzy-k-means Clustering

- Jeder Datenpunkt hat eine Wahrscheinlichkeit für die Zugehörigkeit eines Clusters
- Längere Laufzeit als k-means Clustering -> selten verwendet

DBSCAN - Density Based Clustering

- Dichtebasierter Clustering Algorithmus
- Teil Datenpunkte iterativ in Kernpunkte, dicht-erreichbare Nachbarn und Rauschpunkte ein
- Zwei Punkte sind dicht-verbunden wenn es eine Kette dichter Punkte gibt (Kernpunkte), von den aus sie dicht-erreichbar sind. Andere: Rauschen
- Es entsteht ein Cluster wenn Punkte dicht-verbunden sind
- Bewertung
 - Robust gegen Rauschen
 - Kann verschiedene Daten-Dichten clustern: Herausforderung Dichte- und Bereichsanfrage "richtig" definieren
 - Braucht kein a-priori Wissen über Anzahl der Cluster
 - Nicht deterministisch, Ergebnis hängt von Reihenfolge der Abarbeitung ab
 - Abhängigkeit zur Distanzfunktion -> Curse of dimensionality

OPTICS - Ordering Points To Identify the Clustering Structure

- Erweiterung DBSCAN durch:
 - Berechnung der Distanz, ab der ein Punkt ein Kernpunkt wäre
 - Iterative Sortierung der Nachbar-Punkte nach ihrer Erreichbarkeit
 - Abarbeitung und Zuordnung zu Cluster in Sortierreihenfolge
- Folge:
 - Ausprägung der Cluster im Erreichbarkeitsgraphen sichtbar
 - Liefert auch Subcluster

Dimensionsreduzierung

Unüberwachtes Lernen mit neuronalen Netzen:

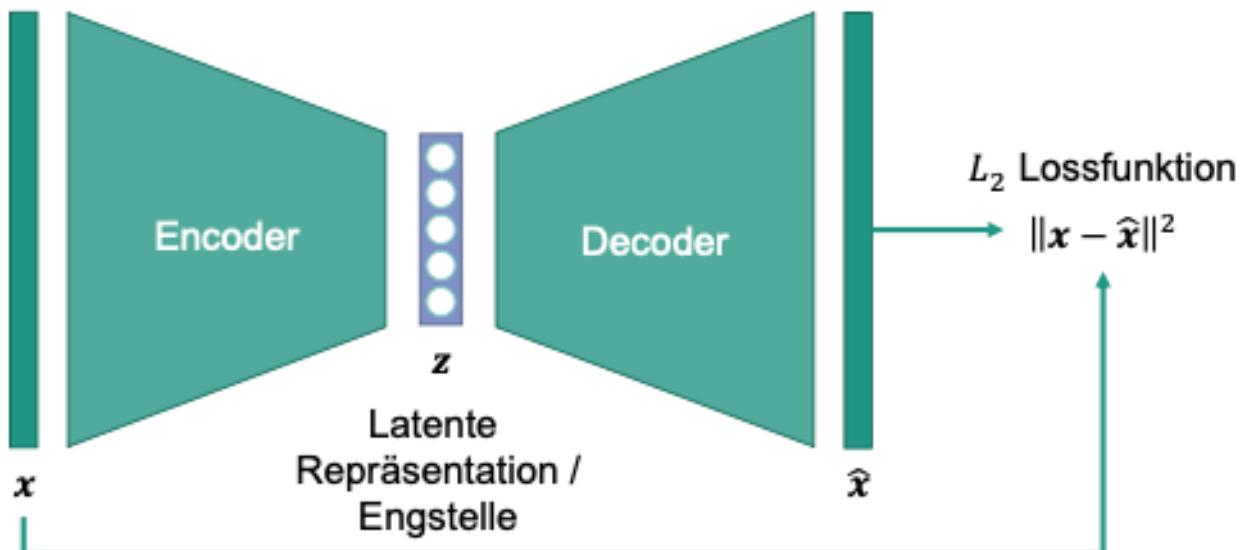
- Netz lernt die Eingabe zu rekonstruieren, Struktur des Netzes erzeugt dabei Einschränkungen
- Netz lernt so Merkmale aus Datensatz

Autoencoder

- Encoder: Komprimierung von Daten, Decoder: Generator
- Tunneln von Informationen durch Engstelle (latente Repräsentation)
-> Dimensionsreduzierung

- Dimensionenreduzierung

- Unüberwachtes Trainieren über den Rekonstruktionsfehler



- Benötigt keine Labels, genutzt z.B. für Entfernen von Rauschen

Restricted Boltzmann Maschine (RBM)

Daten werden zwischen sichtbarer und verdeckter Schicht hin und her geschickt.
Gewichte der Verbindungen interessant -> Dimensionsreduzierung

Deep Belief Networks

- Generatives graphisches Modell
- Gestapelte RBM
- Schichten werden sequentiell trainiert
- Anwendung als Merkmalsdetektoren/Clustering/Klassifikation

10 Reinforcement Learning

Dienstag, 6. Juni 2023 10:55

- Menschen und Tiere lernen durch Interaktion mit der Umgebung
- Unterschiede zu supervised und unsupervised learning:
 - o Aktiv statt passiv
 - o Zukünftige Interaktion kann von früheren abhängig sein
- Zielgerichtet
- Optimales Verhalten ohne Beispiele lernen
- Belohnungssignal wird optimiert

Einführung

Es gilt für jeden Zeitschritt t :

- Agent führt eine Aktion A_t aus
- Umgebung aktualisiert ihren Zustand S_{t+1} entsprechend, sendet die Beobachtung O_{t+1} und Belohnung R_{t+1}
- Agent erhält Beobachtung O_{t+1} und Belohnung R_{t+1}

Belohnung R_t

- Skalares Feedbacksignal
- Beschreibt Güte der Aktion zum Zeitpunkt t
- Agent versucht kumulierte Belohnung G_t zu maximieren
- Basis für RL ist die Belohnungshypothese (welche Belohnung wird gegeben)

Aktionen

- Unterschiedliche Umgebung ermöglichen unterschiedliche Aktionen
- Aktionsraum: Menge aller gültigen Aktionen in der Umgebung
- Aktionsräume können diskret oder kontinuierlich sein

Zustände und Beobachtungen

- Zustand s ist eine **vollständige** Beschreibung des Zustands der Welt
- Eine Beobachtung o ist eine **partielle** Beschreibung des Zustands der Welt

Annahme in der VL: Beobachtung $O_t = \text{Zustand } S_t$ (Umgebung vollständig beobachtbar)

Markov'scher Entscheidungsprozess (MDP)

- Mathematische Formulierung für Interaktion zwischen Agent und

Umgebung

- Annahme $O_t = S_t$

Formalisierung:

- S : endlicher Satz von Zuständen
- A : endlicher Satz von Aktionen
- P : Transitionswahrscheinlichkeit
- R : Belohnungsfunktion
- γ : Diskontierungsfaktor zwischen [0,1]

Transitionswshk.: $P(S_{t+1} = s' | S_t = s, A_t = a)$

Belohnungfkt.: $r = R(S_t = s, A_t = a)$

Notation

S_t, R_t, A_t, S_{t+1} sind Zufallsvariablen,

die jeweils das Ereignis s, a, r, s' annehmen können

Markov Eigenschaft

Definition

Ein Zustand erfüllt die Markov Eigenschaft wenn und nur wenn:

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, \dots, S_t)$$

- Die Zukunft ist unabhängig von der Vergangenheit, wenn man die Gegenwart betrachtet

$$H_{0:t} \rightarrow S_t \rightarrow H_{t+1:\infty}$$

- Der gegenwärtige Zustand S_t fasst alle relevanten Informationen aus der Vergangenheit zusammen.
- Somit kann die Historie $H_{0:t}$ verworfen werden, wenn der aktuelle Zustand bekannt ist.

Hauptkomponenten

Ein Agent kann eine oder mehrere der folgenden Komponenten enthalten:

- Bewertungsbasiert (Value-based): Keine Policy, aber Bewertungsfunktion
- Strategiebasiert (Policy-based): Policy, aber keine Bewertungsfunktion
- Actor Critic: Policy und Bewertungsfunktion

Policy (Strategie)

- Policy π : Verhaltensfunktion des Agenten
- Abhängig vom aktuellen Zustand
- Deterministisch $a = \pi(s)$ oder stochastisch $a \sim \pi(\cdot | s)$

Modelle

- Darstellung der Umgebung durch den Agenten
 \mathcal{P} prognostiziert den nächsten Zustand:

$$\mathcal{P}(s', s, a) \approx P(S_{t+1} = s' | S_t = s, A_t = a)$$

\mathcal{R} prognostiziert die nächste (unmittelbare) Belohnung:

$$\hat{r} = \mathcal{R}(s, a) \approx R(S_t = s, A_t = a)$$

Notation: \hat{r} ist eine Approximation

Bewertungsfunktion (Value Function)

- Vorhersage über erwartete, zukünftige Belohnungen gegeben Policy π
- Bewertete Güte des Zustands:

$$V_\pi(S_t = s) = \mathbb{E}_\pi(R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots | s_t = s)$$

- Diskontierungsfaktor wägt zwischen sofortigen und zukünftigen Belohnungen ab
- Warum der Diskontierungsfaktor?
 - Vermeidet unendliche Belohnungen in zyklischen Markovprozessen
 - Impliziert Unsicherheit über die Zukunft
 - Natürliches Verhalten zeigt Präferenzen für sofortige Belohnungen

Formalisierung des Ziels

Aktionen in einem MDP führen zu unmittelbaren Belohnungen r_t , die zu Gewinnen G_t (kumulierte Belohnung) führen:

$$G_t = R_t + R_{t+1} + R_{t+2} + \dots + R_{H-1} + R_H = \sum_{k=0}^{H-t} R_{t+k}$$

G_t kann rekursiv definiert werden:

$$G_H = R_H \quad G_t = R_t + R_{t+1} + R_{t+2} + \dots + R_{H-1} + G_H$$

$$G_{H-1} = R_{H-1} + G_H \quad G_t = R_t + R_{t+1} + R_{t+2} + \dots + G_{H-1}$$

⋮

$$G_{t+1} = R_{t+1} + G_{t+2} \quad G_t = R_t + G_{t+1}$$

- Diskontierter Gewinn (endlicher Zeithorizont)

$$\begin{aligned} G_t &= R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots + \gamma^{H-t} R_H = \sum_{k=0}^{H-t} \gamma^k R_{t+k} \\ &= R_t + \gamma G_{t+1} \end{aligned}$$

Anmerkung

Die Gewinne sind Zufallsvariablen, die vom MDP und der Policy abhängen

=> Ziel des Agenten ist es eine Policy zu finden, die den erwarteten Gewinn G_t maximiert

Zustandswertfunktion (State Value Function)

- Bewertet Güte eines Zustands
- Erwartete kumulierte Belohnung ausgehend vom Zustand s wenn im folgenden Policy π verfolgt wird:

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

Aktionswertfunktion (Action Value Function)

- Bewertet die Güte einer Aktion in einem Zustand
- Erwartete kumulierte Belohnung ausgehend vom Zustand s und der Aktion a wenn im folgenden die Policy π verfolgt wird:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

=> Wenn der Agent eine der Funktionen maximieren kann, hat er die optimale Strategie

Für jeden Markov-Entscheidungsprozess

- Gibt es eine optimale Policy π^* , die besser oder gleich allen anderen Policies ist
- Alle optimalen Policies erreichen die optimale Zustandswertfunktion $V^*(s)$
- Alle optimalen Policies erreichen die optimale Aktionswertfunktion $Q^*(s, a)$

Bellman Gleichung

Zustandswertfunktion

V-Funktion $\rightarrow V(s)$

- Die Bewertungsfunktion ist eine Vorhersage der zukünftigen Belohnung ausgehend vom Zustand s

$$V_\pi(s) = \mathbb{E}[G_t | S_t = s, \pi]$$

- $V(s)$ kann rekursiv definiert werden:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}[R_t + \gamma G_{t+1} | S_t = s, \pi] \\ &= \mathbb{E}[R_t + \gamma V_\pi(S_{t+1}) | S_t = s, A_t \sim \pi(s)] \\ &= \sum_{a \in A} \pi(a|s) \left(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V_\pi(s') \right) \end{aligned}$$

Aktionswertfunktion

Q-Funktion $\rightarrow Q(s, a)$

- Die Bewertungsfunktion ist eine Vorhersage der zukünftigen Belohnung ausgehend vom Zustand s und der Aktion a

$$Q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a, \pi]$$

- $Q(s, a)$ kann rekursiv definiert werden:

$$\begin{aligned}
 Q_\pi(s, a) &= \mathbb{E}[R_t + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a] \\
 &= \mathbb{E}[R_t + \gamma Q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a, A_{t+1} \sim \pi(S_{t+1})] \\
 &= R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \sum_{a' \in A} \pi(a'|s') Q_\pi(s', a')
 \end{aligned}$$

Optimalitätsprinzip von Bellman

Wenn der Weg $a \rightarrow b \rightarrow d$ die Verbindung von a nach d mit maximaler Belohnung $R^* = R_1 + R_2$ ist, dann muss $b \rightarrow d$ die Verbindung von b nach d mit maximaler Belohnung sein.

Bellman Optimalitätsgleichung für die Zustandswertfunktion

$$V^*(s) = \max_a \left(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s') \right)$$

Bellman Optimalitätsgleichung für die Aktionswertfunktion

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a'} Q^*(s', a')$$

11 Reinforcement Learning 2

Mittwoch, 7. Juni 2023 11:35

Anmerkung: Um die Formeln und Inhalte von der vorherigen VL angewendet zu sehen, lohnt es sich die Wiederholung dieser in der VL RL2 anzugucken.

MDP lösen

Ziel: optimale Policy π^* finden, dann ist MDP gelöst

Problem:

- Formeln aufzustellen und direkt mathematisch zu lösen ist extrem aufwendig: $O(|S|^3)$
- Bellman Optimalitätsgleichung ist nichtlinear, und mit LGS nicht lösbar

Lösung: iterative Methoden:

1. Modellbasierte Ansätze (Dynamische Programmierung)
2. Modelfreie Ansätze (Reinforcement Learning)

Dynamische Programmierung (DP)

- Modell der Umwelt ist gegeben
- Es werden Algorithmen genutzt, die die optimale Policy für MDP berechnen
- Alle Algorithmen beinhalten die Policy Evaluation und Policy Improvement

Policy Evaluation (Strategieevaluation)

- Bellman Gleichung für die Zustandswertfunktion V_π
$$V_\pi(s) = \mathbb{E}[R_t + \gamma V_\pi(S_{t+1}) | S_t = s, \pi]$$

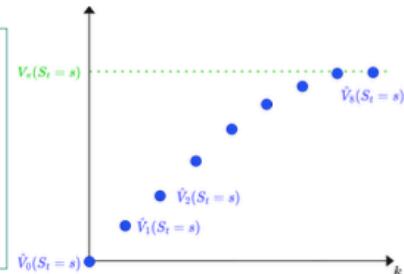
Notation

$\hat{V}_{k+1}(s)$ = ist eine Approximation für $V_\pi(s)$

- Idee: $V_\pi(s)$ iterativ berechnen.

Algorithmus

- Zuerst wird \hat{V}_0 initialisiert, z.B. auf Null.
- Wiederhole:
 - $\hat{V}_{k+1}(s) \leftarrow \mathbb{E}[R_t + \gamma \hat{V}_k(S_{t+1}) | S_t = s, \pi]$
 - Stoppen: Wenn $\hat{V}_{k+1}(s) = \hat{V}_k(s)$, $\forall s$
 - → Wir haben $V_\pi(s)$ gefunden



Policy Improvement (Strategieverbesserung)

Algorithmus

■ Iteriere

$$\begin{aligned}\forall s: \pi_{new}(s) &= \arg \max_a Q_\pi(s, a) \\ &= \arg \max_a \mathbb{E}[R_t(s, a) + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a]\end{aligned}$$

■ Dann bewerte π_{new} und wiederhole

■ Anspruch: Es lässt sich zeigen, dass

$$V_{\pi_{new}}(s) \geq V_\pi(s), \forall s$$

Policy Iteration

Wiederhole Evaluation und improvement so lange, bis es zum Optimum konvergiert

Dabei reicht es meist die Evaluation, nur 3 Iterationen durchzuführen

Reinforcement Learning

- Trial-and-Error-Lernen
- Roboter kann Erfahrungen sammeln: Erfahrung (Transition) bezieht sich auf die Zustandsänderung, die erfolgt, wenn der Agent eine Aktion in einer Umgebung ausführt.
- Trajektorie (Episode): bezieht sich auf die Abfolge von Zuständen, Aktionen und Belohnungen, die ein Agent durchläuft, während er in einer Umgebung interagiert

Policy Evaluation (Modellfrei)

Algorithmus

■ Wiederhole:

- Generiere Trajektorie τ mit Policy π von zufälligem Zustand s_0
- Für alle Erfahrungen s, a der Episode (von s_k, s_{k-1}, \dots, s_0):
 - $G \leftarrow$ Berechne den Gewinn der auf s_t, a_t folgt
$$G = r_t + \gamma r_{t+1} + \dots + \gamma^k r_k$$
 - Füge G zu Gewinne(s_t, a_t) hinzu
 - Schätze die Aktionswertfunktion
$$\hat{Q}_\pi(s_t, a_t) \leftarrow \text{Durchschnitt}(\text{Gewinne}(s_t, a_t))$$

Policy Improvement (Modellfrei)

- Agent muss Balance zwischen Exploration und Exploitation finden
- Exploration (Wissen erweitern): Such nach neuen und unbekannten Aktionen oder Zuständen, die zu einer höheren Belohnung führen können

探索：寻找新的未知的可能导致更高奖励的动作或者状态

~~Aktionen und Zustände, die zu einer höheren Belohnung führen können~~

- Exploitation (Wissen ausnutzen): Die Verwendung des bereits erworbenen Wissens, um die höchstmögliche Belohnung zu erzielen
- Nutze Epsilon-Greedy Policy: 利用已获得的知识，来达到最高可能的回报
 - o Wähle eine greedy Aktion (Exploitation) mit Wahrscheinlichkeit $1-\epsilon$
 - o Wähle eine zufällige Aktion (Exploration) mit Wahrscheinlichkeit ϵ
 - o Beispiel ϵ -greedy (decreasing):
 - Epsilon konvergiert im Laufe des Trainings gegen 0, somit geht die Suchstrategie von global zu lokal während des Lernprozesses

Deep Reinforcement Learning

- Problem: Reinforcement Learning wir häufig auf komplexe Probleme angewandt, dann können nicht mehr alle Werte aus der Exploration gesammelt/gespeichert werden
- Daher: Nutze Neuronal Netze um V- oder Q-Funktion zu approximieren

12 Lernen nach Bayes

Donnerstag, 8. Juni 2023 10:56

Statistische Lernverfahren:

- Kombinieren vorhandenes Wissen (a priori Wahrscheinlichkeiten) mit beobachteten Daten

Theorem von Bayes

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Diagramm zur Herleitung des Bayes-Theorems:

- Ein grüner Pfeil führt von "likelihood" zu $P(D|h)$.
- Ein blauer Pfeil führt von "a priori" zu $P(h)$.
- Ein roter Pfeil führt von "a posteriori" zu $P(h|D)$.

- $P(h)$: A priori Wahrscheinlichkeit, dass h aus H gültig ist (d.h. vor Beobachtung von D).
- $P(D)$: Wahrscheinlichkeit, dass D als Ereignisdatensatz auftritt (ohne Wissen über gültige Hypothese).
- $P(D|h)$: Likelihood, Wahrscheinlichkeit des Auftretens von D in einer Welt, in der h gilt.
- $P(h|D)$: A posteriori Wahrscheinlichkeit, dass h gilt gegeben den beobachteten Daten D .

Bedingte Unabhängigkeit

$$P(X|Y, Z) = P(X|Z)$$

$$\text{Bsp.: } P(\text{Donner}|\text{Regen}, \text{Blitz}) = P(\text{Donner}|\text{Blitz})$$

-> Donner ist bedingt unabhängig von Regen gegeben Blitz

MAP-/ML-Hypothesen

Maximum a posteriori (MAP) Hypothese: Hypothese h aus H mit der größten Wahrscheinlichkeit gegeben der beobachteten Daten D :

$$\begin{aligned} h_{MAP} &= \arg \max_{h \in H} P(h|D) \\ &= \arg \max_{h \in H} \frac{P(D|h)P(h)}{P(D)} \quad \text{Bayes} \\ &= \arg \max_{h \in H} P(D|h)P(h) \quad P(D) = \text{const.} \end{aligned}$$

Unter der Annahme $P(h_i) = P(h_j)$ gilt:

$$h_{ML} = \operatorname{argmax}_{h_i \in H} P(D|h) \rightarrow \text{Maximum Likelihood Hypothese}$$

Konzeptlernen

- Frage: Liefert Suche vom speziellen zum Allgemeinen eine MAP Hypothese?
- Konsistenter Lerner: liefert Hypothese, die keine Fehler auf den Trainingsdaten macht
 - > jeder konsistente Lerner gibt eine MAP-Hypothese aus
 - > Entwicklung der a posteriori Wahrscheinlichkeit mit wachsender Anzahl

-> Entwicklung bei a posteriori Wahrscheinlichkeit mit wachsender Anzahl

von Trainingsdaten: für inkonsistente Hypothesen $P(h)=0$

Optimaler Bayes-Klassifikator

- Sucht wahrscheinlichste Klassifikation $v_j \in V$ einer Instanz $x \in X$

$$v_{OB} = \arg \max_{v_j \in V} \sum_{h_i \in H} P(v_j | h_i) P(h_i | D)$$

- Vorsicht: Ausgabe der $h_{MAP}(x)$ ist nicht immer die wahrscheinlichste Klassifikation
- Vorteil: Kein anderes Klassifikationsverfahren (bei gleichem Hypothesenraum und Vorwissen) schneidet im Durchschnitt besser ab
- Nachteil: Sehr kostenintensiv bei großen H
- Vereinfacht: Gibbs-Algorithmus (wähle h zufällig, nutze h(x) als Klassifikation) oder erweitert Voting Gibbs Algorithmus

Naiver Bayes-Klassifikator

- Sucht wahrscheinlichste Klassifikation $v_{MAP} \in V$ für eine neue Instanz (a_1, a_2, \dots, a_n)

Naiver Bayes-Klassifikator:

$$v_{NB} = \arg \max_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$$

-> vereinfachende Annahme: a_i bedingt unabhängig (Attribute sind unabhängig)

- Vorgehen: $P(v_j)$ und $P(a_i | v_j)$ werden basierend auf Häufigkeiten aus Daten geschätzt -> Wahrscheinlichkeiten sind die Parameter der gelernten Hypothese -> neue Instanzen werden mit MAP Regel klassifiziert -> Wenn Annahme erfüllt, ist v_{NB} äquivalent zu MAP-Klassifikation
- Problem: Was wenn ein Attribut in Daten gar nicht vorkommt? -> Schätzung mit m-Laplace Schätzer

Bayessche Netze

Problem: Annahme der bedingten Unabhängigkeit bei Navier Bayes oft zu restriktiv

Bayessche Netze: beschreiben bedingte Abhängigkeiten (und Unabhängigkeiten) bzgl. Untermengen von (Zufalls-)Variablen

-> erlauben so Kombination von a priori Wissen über bedingte (Un-)Abhängigkeiten von Variablen mir beobachteten Trainingsdaten

Aufbau:

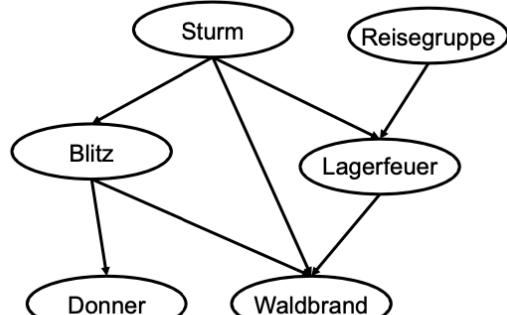
- X ist Nachfolger von Y wenn Pfad existiert (-->)
- Jede Zufallsvariable ist ein Knoten, Abhängigkeiten sind Kanten
- ...

Es gilt:

$$P(y_1, \dots, y_n) = \prod_{i=1}^n P(y_i | \text{Vorgänger}(Y_i))$$

■ Wobei:

- $\text{Vorgänger}(Y_i)$: die Menge der direkten Vorgänger von Y_i ist.



$$P(\text{Lagerfeuer} | \text{Sturm}, \text{Reisegruppe})$$

	S, R	$S, \neg R$	$\neg S, R$	$\neg S, \neg R$
L	0.4	0.1	0.8	0.2
$\neg L$	0.6	0.9	0.2	0.8

Wertbestimmung von Variablen, gegeben der beobachteten Werte:

- Netz enthält alle benötigten Informationen
- Ableitung einer einzigen Variablen einfach
- Aber: Der allgemeine Fall ist NP-vollständig

In der Praxis:

- Berechnung von approximierten Lösungen mit Monte Carlo Zufallssimulationen

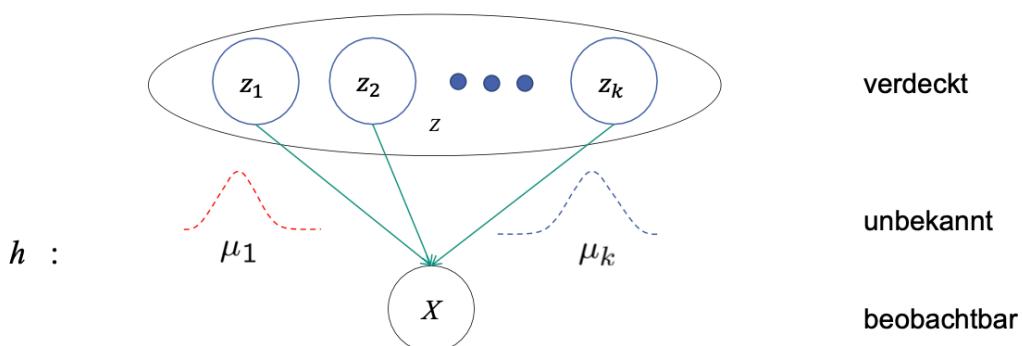
Lernen:

- Struktur bekannt, alle Variablen beobachtbar -> lernen wie Navier Bayes Klassifikator
- Struktur bekannt, nur einige Variablen beobachtbar -> Gradientenabstieg, EM-Methode
- Struktur unbekannt -> heuristische Verfahren

Expectation Maximization (EM)-Algorithmus

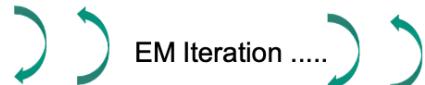
Problem: Daten nur partiell beobachtbar und Parameter einer Hypothese sollen geschätzt werden

Idee: Iterativer Ansatz -> Schätzen der nicht beobachtbaren Werte (E) und Anpassung der Parameter (M)



E -Schritt: Wenn x und h bekannt $\rightarrow z$ bzw. $P(z)$

M - Schritt: Wenn z und x bekannt $\rightarrow h_{\mu_1, \dots, \mu_k}$



13 HMM - Hidden Markov Modelle

Montag, 26. Juni 2023 10:55

Motivation: Prozesse und Signale in der realen Welt oft nichtdeterministisch und verrauscht

-> Nutzung von stochastischen Prozess- und Signalmodellen z.B. zur Gestenerkennung, Ampelzustandsschätzung

-> Nutzung überall dort, wo zugrunde liegende stochastische Prozesse nur indirekt beobachtbar sind

Diskreter Markov Prozess

N diskrete Zustände: $S = \{S_1, S_2, \dots, S_N\}$

Zeitpunkte der Zustandsübergänge: $t = 1, 2, \dots, T$, aktueller Zustand: q_t

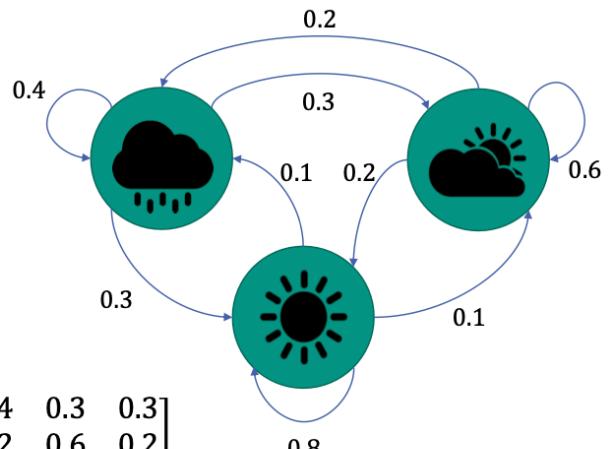
Parameter:

- $\pi_i = P(q_1 = S_i)$: Anfangszustandswahrscheinlichkeit
- $a_{ij} = P(q_t = S_j | q_{t-1} = S_i) = P(S_j | S_i)$: Übergangswahrscheinlichkeiten

Beispiel: Wetter I

- Jeden Mittag wird das Wetter beobachtet:

- S_1 : Regen (oder Schnee)
- S_2 : bewölkt
- S_3 : sonnig
- $S = \{S_1, S_2, S_3\}$

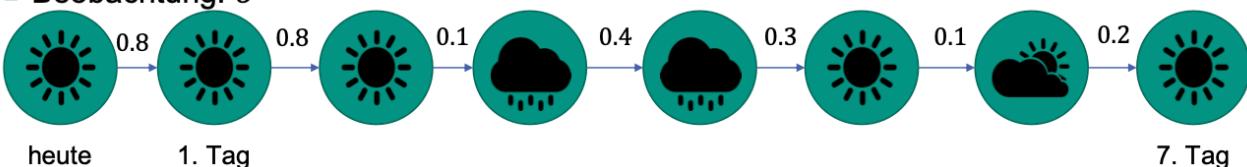


- Übergangswahrscheinlichkeiten:

$$A = (a_{ij}) = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$

- **Frage:** Wenn es heute sonnig ist, wie wahrscheinlich ist es, dass das Wetter der nächsten 7 Tage {sonnig, sonnig, Regen, Regen, sonnig, bewölkt, sonnig} ist?

- Beobachtung: O



- Lösung:

$$\begin{aligned} P(O|\text{Modell}) &= P(S_3, S_3, S_3, S_1, S_1, S_3, S_2, S_3 | \text{Modell}) \\ &= P(S_3) \cdot P(S_3 | S_3) \cdot P(S_3 | S_3) \cdot P(S_1 | S_3) \end{aligned}$$

$$\begin{aligned}
& \cdot P(S_1|S_1) \cdot P(S_3|S_1) \cdot P(S_2|S_3) \cdot P(S_3|S_2) \\
& = 1 \cdot 0.8 \cdot 0.8 \cdot 0.1 \cdot 0.4 \cdot 0.3 \cdot 0.1 \cdot 0.2 \\
& = 1.536 \times 10^{-4}
\end{aligned}$$

Markov-Bedingung

Beschränkter Horizont: Die Wahrscheinlichkeit einen Zustand zu erreichen ist nur von seinem direkten Vorgängerzustand abhängig

$$P(q_{t+1} = S_j | q_t = S_i, q_{t-1} = S_k, \dots) = P(q_{t+1} = S_j | q_t = S_i)$$

HMM

Beschreibt einen doppelt stochastischen Prozess: Beobachtungen kommen als stochastische Funktion des Zustands hinzu -> Zustände können nur indirekt beobachtet werden

- Ein HMM ist ein Fünf-Tupel: $\lambda = \{S, V, A, B, \Pi\}$ mit:
 - S : Menge der Zustände, $S = \{S_1, S_2, \dots, S_N\}$
 - V : Menge der Ausgaben/Beobachtungen, $V = \{V_1, V_2, \dots, V_M\}$
 - $A = (a_{ij})$: Matrix der Übergangswahrscheinlichkeiten $a_{ij} = p(S_j|S_i)$, $A \in [0, 1]^{N \times N}$
 - $B = (b_{ik})$: Matrix der Emissions(Beobachtungs)-Wahrscheinlichkeiten, $B \in [0, 1]^{N \times M}$
 - Π : Verteilung des Anfangszustands $\Pi = \{\pi_i | \pi_i = p(q_1 = S_i)\}$ wobei q_1 - Startzustand
- Es gilt:
 - Wahrscheinlichkeit für einen Zustand: $p(q_{t+1} = S_j) = \sum_{S_i \in S} a_{ij} \cdot p(q_t = S_i)$
 - Wahrscheinlichkeit für eine Beobachtung: $p(o_t) = \sum_{S_i \in S} b_{S_i}(o_t) \cdot p(q_t = S_i)$
- Hinweise/Notation:
 - q_t bezeichnet den Zustand, $o_t \in V$ die (bekannte) Ausgabe/Beobachtung zum Zeitpunkt t
 - $b_{S_i}(V_k) = b_{ik}$ bezeichnet Wahrscheinlichkeit der Ausgabe/Beobachtung V_k im Zustand S_i

Evaluationsproblem

Wie gut erklärt ein Modell eine Beobachtungssequenz?

Gegeben: Modell $\lambda = \{S, V, A, B, \Pi\}$

Gesucht: Wahrscheinlichkeit $P(O|\lambda)$ für die Ausgabe $O = o_1 o_2 \dots o_T$

| Annahme: Eine feste Zustandsfolge q_1, q_2, \dots, q_T sei gegeben

- Wahrscheinlichkeit für die Ausgabe O :

$$P(O|Q, \lambda) = \prod_{t=1}^T P(o_t|q_t, \lambda) = b_{q_1}(o_1) \cdot b_{q_2}(o_2) \cdot \dots \cdot b_{q_T}(o_T)$$

- Aber wir wissen nicht, was die verdeckte Zustandsfolge war.
- Wir müssen daher alle möglichen Zustandsfolgen betrachten

1. Lösungsmethode: Naiver Ansatz

- Summierung aller möglichen Zustandsfolgen -> hoher Aufwand $O(2T * N^T)$

2. Lösungsmethoden: vorwärts-/rückwärtsalgorithmus

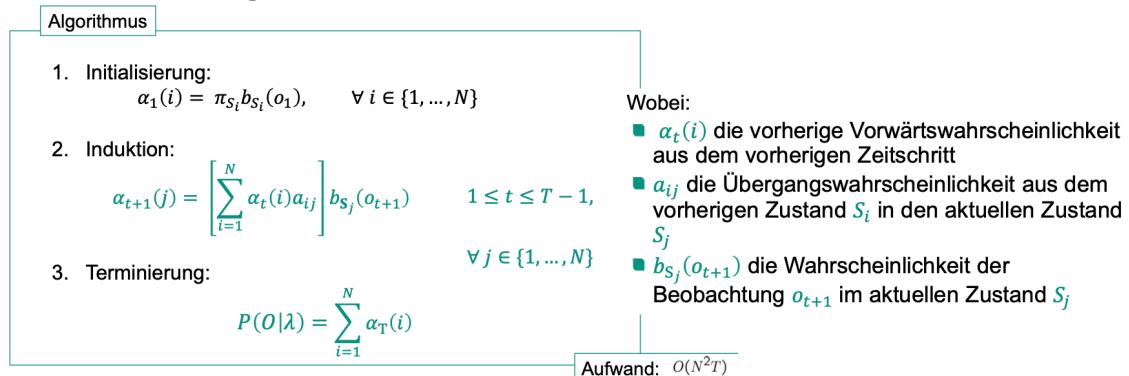
- Idee: Berechne Teilresultate und speichere sie in Tabelle

Umsetzung:

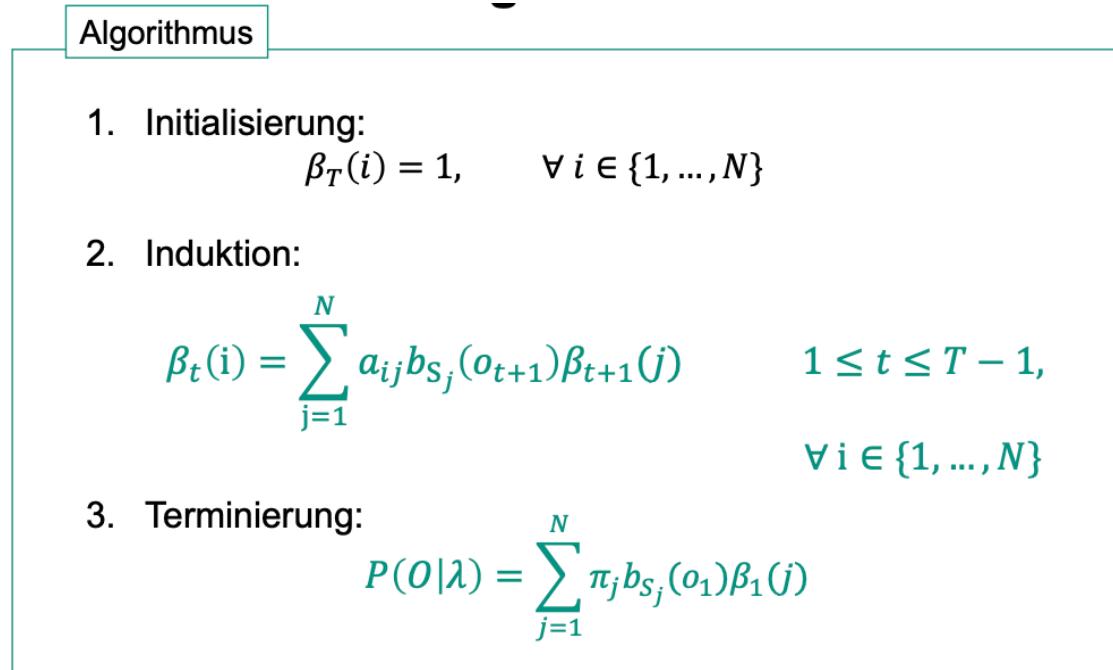
- Definiere z.B. vorwärts die Wahrscheinlichkeit, dass $o_1 o_2 \dots o_t$ beobachtet wurde und S_j = Zustand zum Zeitpunkt t (rückwärts analog s. später)
- $\alpha_t(j) = P(o_1, o_2, \dots, o_t, q_t = S_j | \lambda)$

- Vollständige Berechnung durch Induktion

Vorwärts-Algorithmus:



Rückwärts-Algorithmus



Dekodierungsproblem

Gesucht: wahrscheinlichste Zustandsfolge Q, die Ausgabesequenz O erklärt

1. Lösungsmethode:

- Wahl der Zustände q_t , die unabhängig voneinander am wahrscheinlichsten sind
- Problem: Bei nicht vollständig vernetztem HMM u.U. ungültiger Pfad
- Besser: Wahl der insgesamt besten Zustandsfolge über Maximierung von $P(Q, O | \lambda)$
- Problem: sehr aufwändig

2. Lösungsmethode: Viterbi-Algorithmus

- rekursive Maximierung entlang möglicher Zustandstrassen (Pfade)
- Prozesskette -> höchste Wsh. Für Zustandswert -> iterativ alle v_{tsj} Werte bestimmen -> Pfad speichern -> berechne alle nachfolgezustände S_j von S_i
- Unterschied zum Vorwärtsalgorithmus:

$$v_{t+1}(S_j) = b_{S_j}(o_{t+1}) \cdot \max_{1 \leq i \leq N} [v_t(S_i) \cdot a_{ij}]$$

Unterschied zum Vorwärts-Algorithmus:
Maximierung statt Summation

Lern- oder Optimierungsproblem

Optimierung der Modellparameter (Training), gesucht: Anpassung der Parameter, sodass O gut erklärt wird

Kein analytischer Lösungsweg bekannt -> Lokale Maximierung von $P(O|\lambda)$ mit iterativem Ansatz

Baum-Welch-Algorithmus (Spezialfall von EM-Algorithmus):

Gegeben:

- Trainingssequenz $O_{training}$ und
- Hypothesenraum für Modelle $\lambda = \{S, V, A, B, \Pi\}$

Gesucht:

- Modell, das die Daten am besten erklärt:

$$\bar{\lambda} = \arg \max_{\lambda} P(O|\lambda)$$

Ansatz:

- Hypothesenraum wird so gewählt, dass die Anzahl der Zustände vorgegeben wird.
- Lediglich die stochastischen Modellparameter werden angepasst, d.h.:

$$\bar{\lambda} = \{\bar{A}, \bar{B}, \bar{\Pi}\}$$

Algorithmus

1. Beginne mit zufälligem Modell λ , berechne $P(O_{training}|\lambda)$
2. **Schleife:**
3. **E-Schritt:** Bestimme Erwartungswert von Zustandsübergängen (aus und zwischen Zuständen) und Symbolausgaben
4. **M-Schritt:** Neuschätzung der Übergangs- und Emissionswahrscheinlichkeiten → Berechnung eines neuen Modells
5. **Wenn** (lokales) Maximum erreicht ist

Es ist kein globales Maximum garantiert -> Training mit unterschiedlichen Startwerten

Arten von HMM

- Ergodisches Modell: Jeder Zustand kann von jedem anderen Zustand in endlichen Schritten erreicht werden
- Links-nach-rechts-Modell (Bakis-Modell): Zustandsindex wird mit der Zeit größer oder bleibt gleich