Statistische und neuronale Lernverfahren

Martin Stetter

WS 06/07, 2 SWS

Bereich:
prüfbare Vorlesung im Bereich praktischer und theoretischer Informatik:
Künstliche Intelligenz / Maschinelles Lernen

Zeit: Dienstag 8.30 (s.t.) - 10.00

Ort: 03.09.014

Beginn: 24.10.2006

Behandelte Themen

0. „Motivation“: Lernen in Statistik und Biologie

1. Überblick über statistische Datenmodellierungs-Verfahren

2. Das lineare Modell (Regression)

3. Perceptron und Multilagen-Perceptron (Funktionsapproximation)

4. Selbstorganisierende Merkmalskarten (Dichteschätzung)

5. Lernen von Datenmodellen

- Approximation: Bayes‘sches Schließen, MAP, ML
- Maximum Likelihood Schätzung und Fehlerminimierung
- Generalisierung und Regularisierung
- Optimierungsverfahren

6. Kern-Trick und Support Vector Machine

7. Bayesianische Netze (Dichteschätzung und Funktionsapproximation)

Literatur

Statistische Lernverfahren:

- B. Schölkopf, A. Smola: *Learning with Kernels.* MIT Press, Cambridge, MA (2002)
- S**.** Lauritzen. *Graphical Models*, Oxford Univ. Press (1996)
- Kevin Murphy´s tutorial. *A Brief Introduction to Graphical Models and Bayesian Networks* http://www.ai.mit.edu/~murphyk/Bayes/bayes.html

Statistische und neuronale Verfahren:

- C. M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press Oxford (1995)

Neuronale Netze:

- J. Hertz, A. Krogh, R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley, Redwood City CA (1991)

Computational Neuroscience und Bioinformatik

- M. Stetter, *Exploration of Cortical Function*, Kluwer Academic Publishers, Dordrecht (2002)
- P. Baldi, G. W. Hatfield, *DNA Microarrays and Gene Expression*, Cambridge University Press Cambridge, MA (2002)

Parallelveranstaltung zu unüberwachten Lernverfahren

PD Dr. Thomas Runkler: Data Mining und Knowledge Discovery

Bereich: prüfbare Vorlesung im Bereich 1.4 Künstliche Intelligenz / Maschinelles Lernen

Zeit: Montag 8.30 - 10.00
Ort: Raum 00.13.009A
Beginn: 23.10.06

Inhalt:

1. Einführung: Ziel, Definitionen, Schritte der Knowledge Discovery (KDD)
2. Datenquellen, -charakteristika und Fehlerquellen,
3. Datenvorverarbeitung und -filterung
4. Datenvisualisierung: Projektionen, Hauptachsentrafo., mehrdim. Skalierung, Sammon-Methode, selbstorg. Karten
5. Datentransformationen und Merkmalsgenerierung
6. Datenanalyse: Korrelationsanalyse und Scheinkorrelationen, Regression, Klassifikation, Clustering
7. Anwendungsbeispiele

Einführung: Lernen in Statistik und Biologie

Biologisches Lernen	Maschinelles Lernen
Erkennen von Zusammenhängen im Lebensraum	Erkennen von statistischer Struktur in Datensätzen
„Synaptische Plastizität“ im Gehirn	Einstellung der Modellparameter
Erlernen von Fähigkeiten aus Beispielen (prozedurales Lernen)	Überwachtes Lernen
Erlernen von Fähigkeiten durch Probieren	Unüberwachtes Lernen
Finden einfacher Lösungen	Regularisierung
Schlußfolgern	Bayes‘sches Schließen (Inferenz)

Statistisches Lernen: Beispiel

Statistisches / Maschinelles Lernen: „Entdeckung von Struktur in Daten“

<u>Beispiel: 2D-Klassifikation:</u>

M **Fiktive Daten eines Geldinstituts:**

x_1 **= Mittl. abgehobener Geldbetrag**

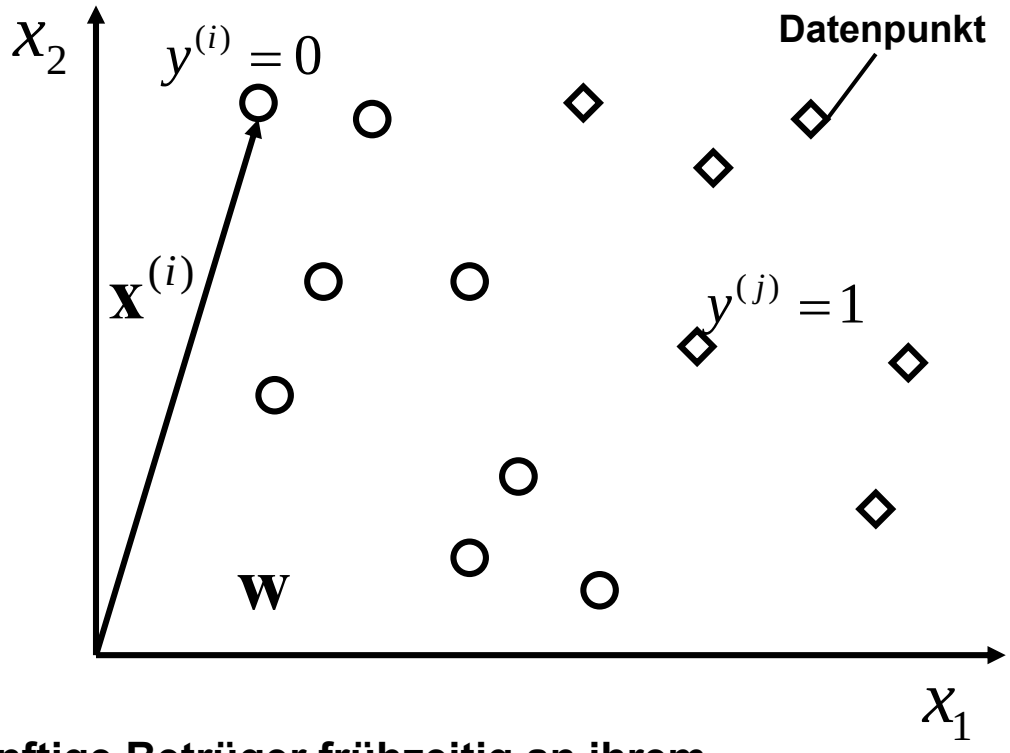
x_2 **= Häufigkeit Abhebevorgänge**

$\mathbf{x} = (x_1, x_2)$ **= „Muster“**

y **= Scheckkartenbetrug (1=ja, 0=nein)**

y **= „Klassenlabel“, Soll, Output**

$(\mathbf{x}^{(1)}, y^{(1)}) \ldots (\mathbf{x}^{(M)}, y^{(M)})$



<u>Ziel:</u> Lerne Klassifikationsregel, um künftige Betrüger frühzeitig an ihrem Verhalten zu erkennen.

Lernen eines Klassifikators:

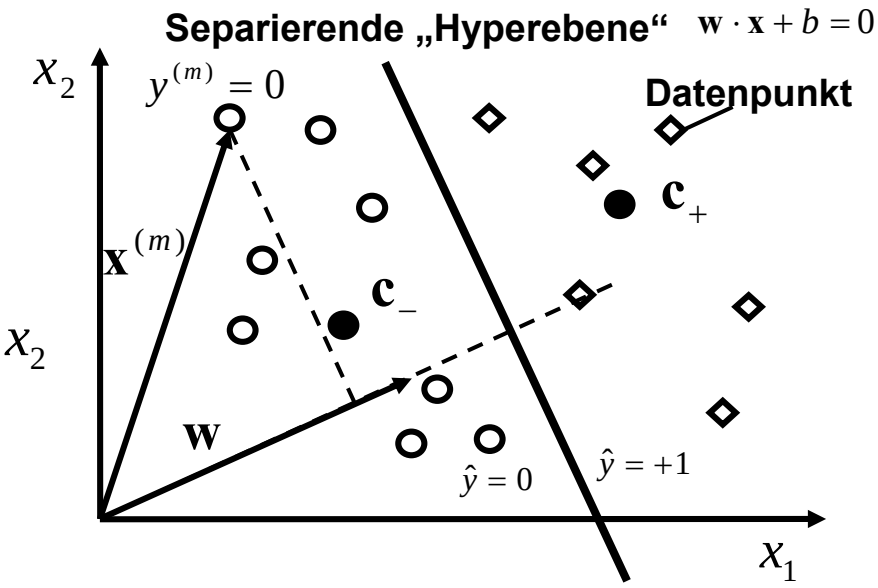
Linearer Klassifikator:

$$\hat{y} = \Theta(\mathbf{w} \cdot \mathbf{x} + b)$$

$$\Theta(x) = 1, x \geq 0$$

$$\Theta(x) = 0, x < 0$$

$$\mathbf{w} \cdot \mathbf{x} := w_1 x_1 + w_2 x_2$$



$(\mathbf{w}, b)$ **= Parametersatz, „Modell“, „Hypothese“**

„Lernen“: Finde besten Parametersatz

Offline-Lernen:

$$\mathbf{c}_{+/-} = \frac{1}{M_{+/-}} \sum_{\{m | y^{(m)} = \pm 1\}} \mathbf{x}^{(m)}$$ **Klassenzentren**

$$\hat{\mathbf{w}} = \mathbf{c}_+ - \mathbf{c}_-$$

$$\hat{b} = -\mathbf{w} \cdot (\mathbf{c}_+ + \mathbf{c}_-)/2$$

Online-Lernen (Beispiel für *b=0*):

Für jeden Datenpunkt ändere w gemäß

$$\Delta \mathbf{w} = \eta (y^{(m)} - \hat{y}^{(m)}) \mathbf{x}^{(m)}$$

(Perceptron-Lernregel, siehe später)

Warum „statistisches“ Lernen?

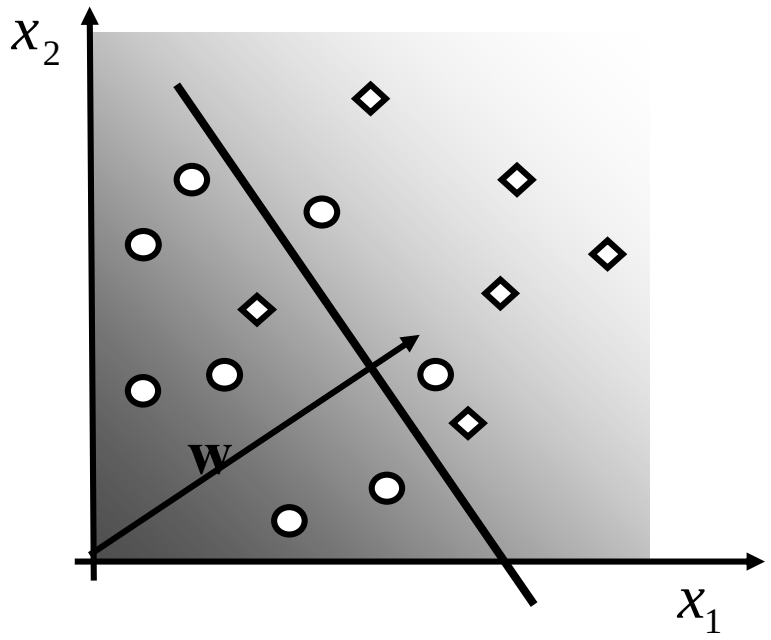
Daten sind unsicher:

-- Datenpunkte x könnten versetzt sein

-- Gemessene Klassen können falsch sein

Klassifikationsgesetz ist unsicher

-- Mitglieder unterschiedlicher Klassen könnten dasselbe Muster x aufweisen



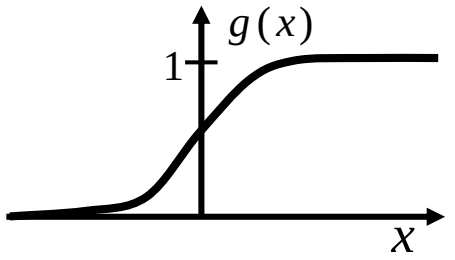
Lösung: Probabilistischer Klassifikator („soft classifier“)

Spezifiziere Wahrscheinlichkeit für Klassenmitgliedschaft

$$\Pr(y(\mathbf{x})=1) = g(\mathbf{w}\cdot\mathbf{x}+b)$$

Beispiel: Logistische Transferfunktion :

$$g(x) = \frac{1}{1+\exp(-x)}$$



Neuronales Lernen: Gehirn und Nervenzelle

Das menschliche Gehirn

Besteht aus Nervenzellen (Neuronen) und Hilfszellen

Hochstrukturiert (Kerne, Areale der Grosshirnrinde)

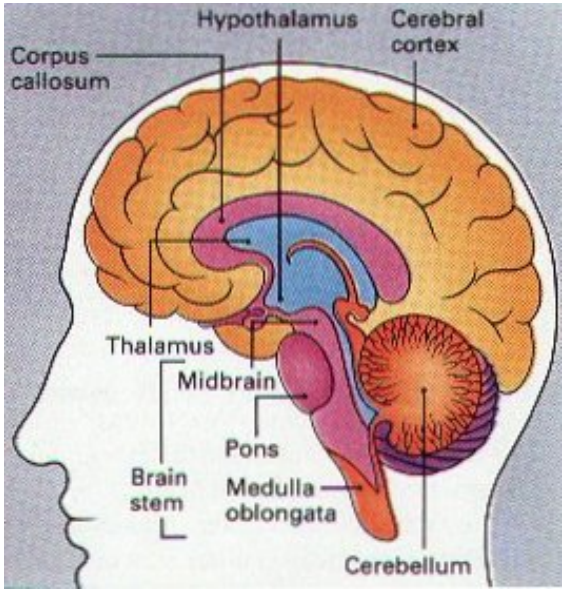
Gigantisches Netz aus Neuronen:

-- **100 000 000 000 Nervenzellen**

-- **Jede Zelle erhält synaptischen Input von ca. 10 000 anderen Nervenzellen (Konvergenz)**

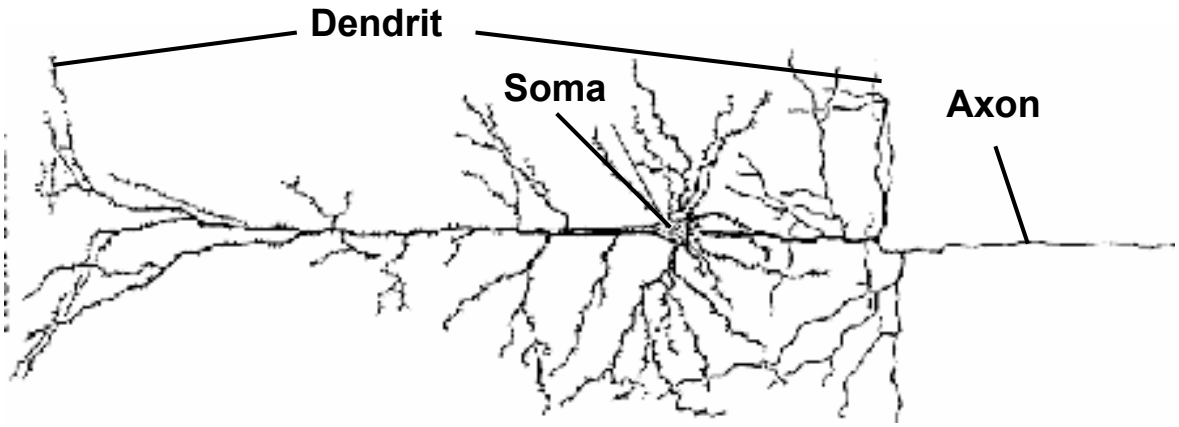
-- **Jede Zelle sendet ca. 10 000 Outputs (Divergenz)**

-- **Gesamte Leitungslänge: 750 000 km !!!**



Nervenzelle

Besteht aus Dendrit,
Soma (Zellkörper)
Axon



Neuronales Lernen: Reizleitung in Neuronen

Funktionsweise des Neurons

Signal: Aktionspotential, „Spike“

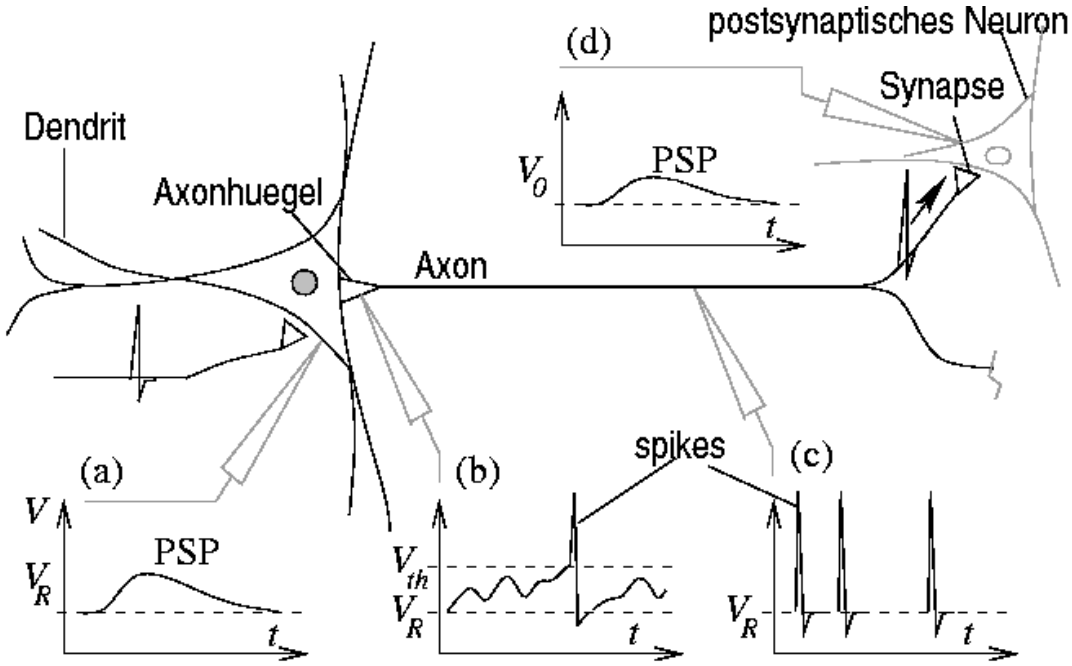
Signalfluss:
Dendrit --> Soma --> Axon-->
Synapse--> Dendrit ...

(a) Spike kommt an;
Synapse injiziert Strom I
Membranspannung steigt (PSP)

(b) Viele PSPs summieren sich
Bei Schwellenspannung: Spike

(c) Spike läuft Axon entlang
verzweigt sich mit dem Axon

(d) Spike kommt an.....



Biologisches Lernen (Hypothese):

Synaptischer Strom I ändert sich in Abhängigkeit von der Zeit und der Hirnaktivität („LTP, LTD“.....)

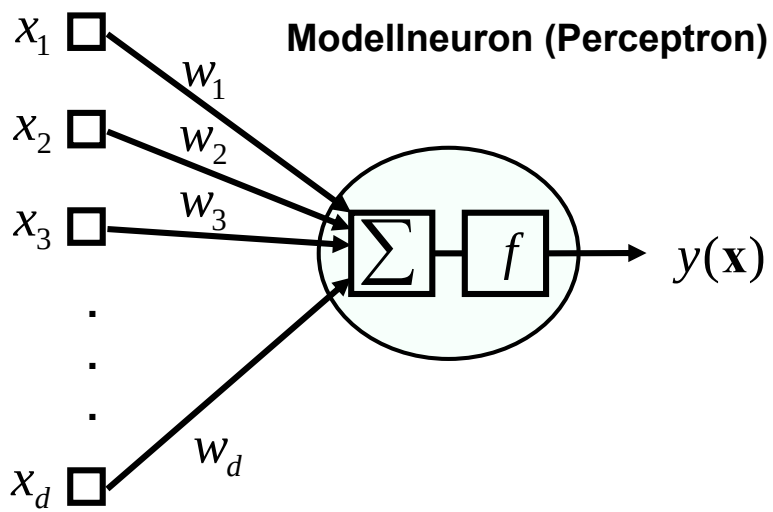
Ratenmodell des Neurons

- **Neuron erhält Signale von d Synapsen**
- **An Synapse i kommen Spikes mit der Rate x_i an**
- **Synapse i induziert Spannung $U_i = w_i x_i$ w_i heißt synaptisches Gewicht**
- **Das Soma summiert die Spannungs-änderungen:**

$$U = \sum_{i=1}^{d} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

- **Die Spikerate y am Axon ist eine sigmoide Funktion der Summenspannung**

$$y(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x} - \theta)$$



Lernen im Modellneuron:

Synaptische Gewichte $\mathbf{w}$ ändern sich abhängig von der Aktivität

Biologisch motivierte Lernregeln... (zB. „Hebb-Regel“)

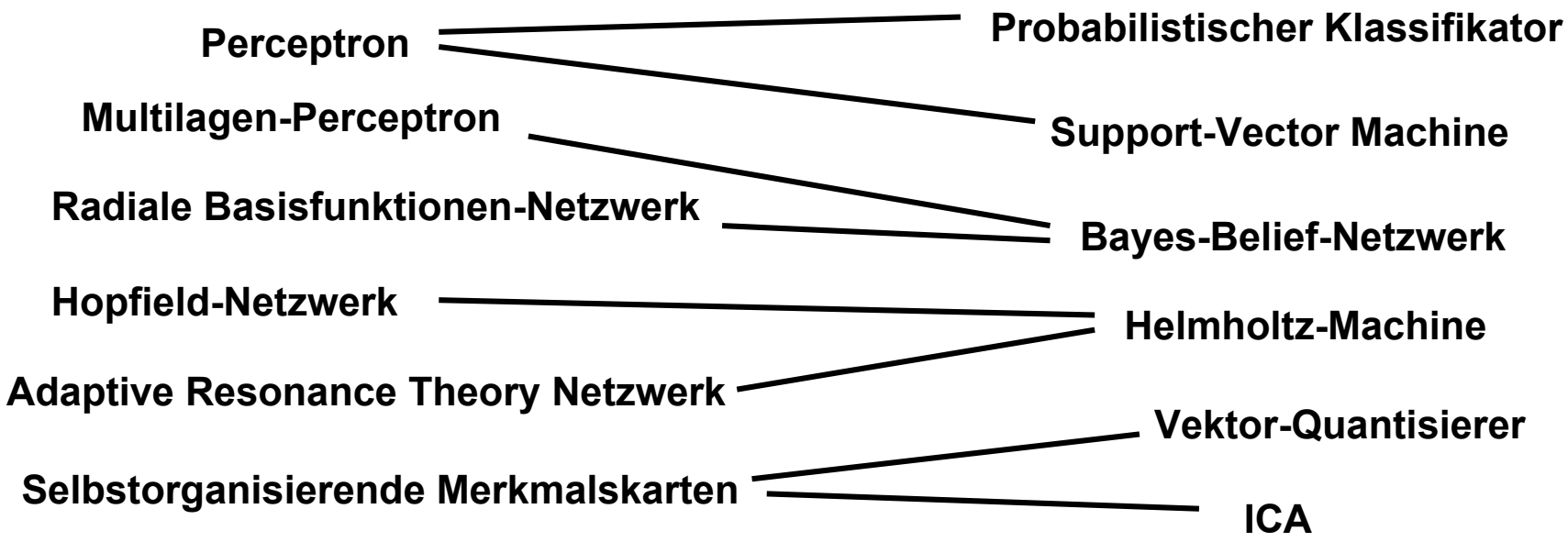
Statistische und Neuronale Lernverfahren: Beispiele

Künstliche Neuronale Netze

Modellneuronen können zu künstlichen neuronalen Netzen zusammengeschaltet werden, zB.

Statistische Datenmodellierung

Modellneuronen können auch als statistische Datenmodelle interpretiert werden, z.B.



Zusammenfassung

- **Ein biologisch motiviertes Modellneuron läßt sich mit einem linearen probabilistischen Klassifikator identifizieren**
- **Viele künstliche Neuronale Netze lassen sich mit statistischen Lernverfahren identifizieren**
- **Viele Verfahren, viele Aufgaben**

„Road Map“

- **Bayes‘sche Inferenz als genereller Rahmen für statistische Lernverfahren**
- **Statistische Datenmodellierung durch Optimierung und Regularisierung**
- **Spezielle maschinelle Lernverfahren und Neuronale Netzwerk-Typen**

Wahrscheinlichkeitstheorie: Grundlagen und Definitionen

- **Wahrscheinlichkeiten und Dichten**
- **Interpretation von Wahrscheinlichkeiten**
- **Einige Definitionen und Gesetze**

Zufallsvektoren

Multivariate Daten: Mehrere Zufallsvariablen nehmen ihre Werte in gegenseitiger Abhängigkeit voneinander an, müssen also zusammen betrachtet werden.

Bsp: Körpergröße und Körpergewicht

- $\mathbf{X} = (X_1, X_2, ..., X_d)$ **= Zufallsvektor; Wertebereich:** $\mathbf{x} \in \mathrm{R}^d$
- $p(\mathbf{x}) = p(x_1, ..., x_d)$ **= zusammengesetzte („multivariate“) Wahrscheinlichkeitsdichte**

$$p(\mathbf{x})d\mathbf{x} = p(x_1, ..., x_d)dx_1 dx_2 ... dx_d$$

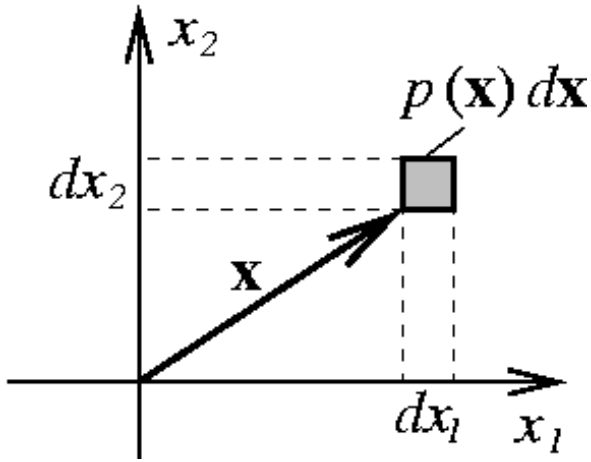
= Wahrscheinlichkeit, bei der Messung eines Vektors

$X_1 \in [x_1, x_1 + dx_1]$ **und in derselben Messung**

$X_2 \in [x_2, x_2 + dx_2]$ **und in derselben Messung ....**

. . .

$X_d \in [x_d, x_d + dx_d]$ **zu finden**



Nomenklatur: Messung = Ziehen aus der Verteilung $p(\mathbf{x})$

Beispiel: Multivariate Gaussverteilung

$$\varphi(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} (\det \boldsymbol{\Sigma})^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right)$$

$\mu =$ **Mittelwert-Vektor**

$\Sigma =$ **Kovarianz-Matrix**

Einschub: Bilinearform: $\mathbf{x}^T \mathbf{A} \mathbf{x} = \sum_{i,j=1}^{d} x_i a_{ij} x_j$

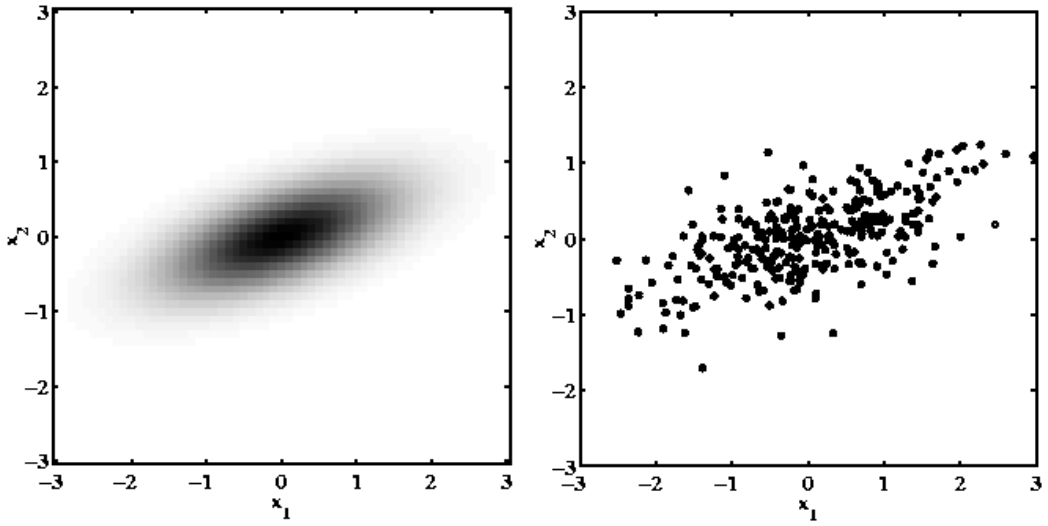
Z.B. 2D: $\mathbf{x}^T \mathbf{A} \mathbf{x} = (x_1, x_2) \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = x_1^2 a_{11} + x_1 x_2 (a_{12} + a_{21}) + x_2^2 a_{22}$

2D Gaussverteilung mit

$$\boldsymbol{\Sigma} = \begin{pmatrix} 2.0 & 0.6 \\ 0.6 & 0.5 \end{pmatrix}$$

Links: Grauwertbild der pdf

Rechts: Stichprobe von 300 Vektoren



Interpretation von Wahrscheinlichkeiten

Betrachte den Wert x_0 einer Zufallsvariable X:

<u>Frequentistische Philosophie:</u>

- **Der Datenpunkt ist eine Stichprobe aus einer existierenden „wahren“ Verteilung $p(x)$**
- **=> $p(x)$ kann als Grenzwert der relativen Häufigkeit interpretiert werden**
- **„Wahrscheinlichkeit als Prozentsatz“**
- **Typisches Beispiel: Wahrscheinlichkeit, 6 zu würfeln**

<u>Bayesianische Philosophie:</u>

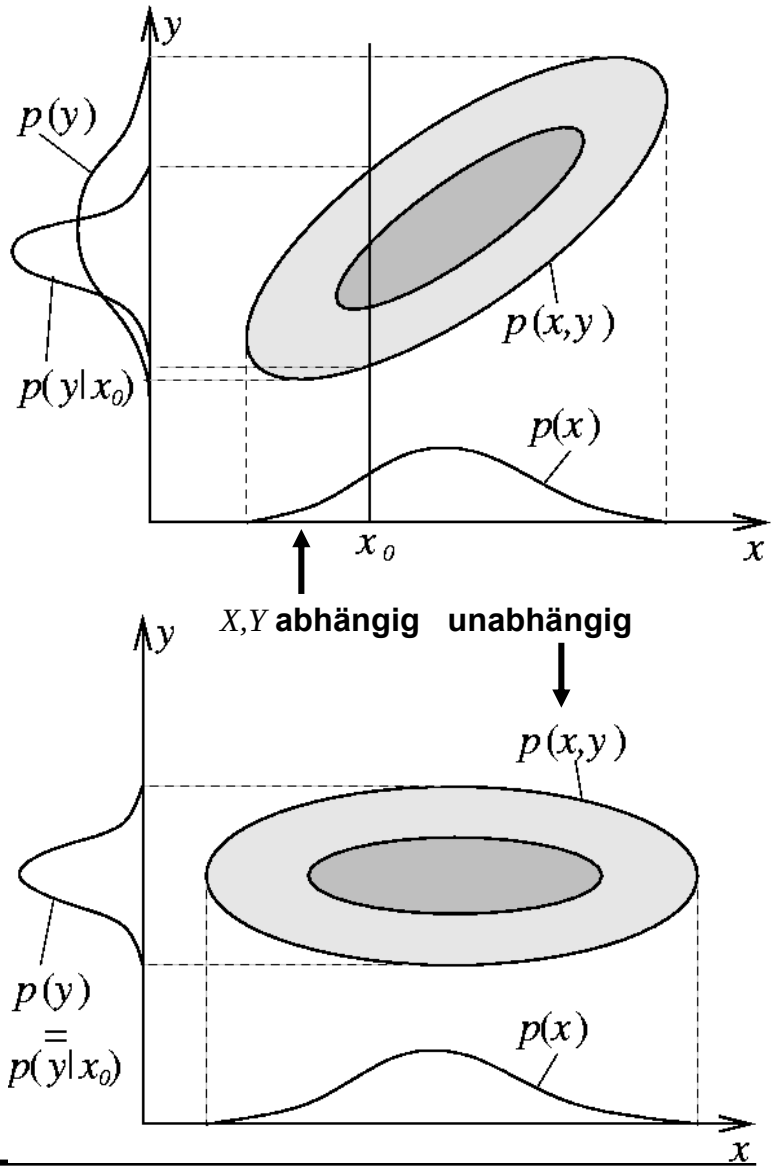
- **Nur der Datenpunkt x_0 existiert, es gibt keine zugrundeliegende Verteilung**
- **„Wahrscheinlichkeit als Glaube (Belief) des Eintretens eines Sachverhalts“**
- **Basiert allein auf Vorwissen und auf den beobachteten Daten**
- **Typisches Beispiel: Wahrscheinlichkeit, den nächsten Marathon zu gewinnen**

Einige Definitionen und Gesetze

Betrachte zwei Zufallsvariablen X, Y
(hier oBdA kontinuierlich):

<u>Bedingte Wahrscheinlichkeit</u> $p(y \mid x)dxdy$

- **Wahrscheinlichkeit,** $Y \in [y, y+dy]$ **zu finden**
  wenn $X \in [x, x+dx]$ **bekannt ist**
- **Zusammengesetze Dichte:**
  $$p(y,x) = p(y \mid x)\, p(x)$$
- **Marginalisierung:**
  $$p(y) = \int p(y,x)dx = \int p(y \mid x)\, p(x)dx$$
- **Statistische Unabhängigkeit:**
  $$\Leftrightarrow p(y,x) = p(y)\, p(x) \Leftrightarrow p(y \mid x) = p(y)$$



Betrachte mehrdimensionale Daten: $\mathbf{X} = (X_1, X_2, ..., X_d)$

Dekomposition von zusammengesetzten Dichten:

$$p(x_d, x_{d-1}, ..., x_2, x_1) =$$

$$= p_d(x_d \mid x_{d-1}, ..., x_1) p_{d-1}(x_{d-1} \mid x_{d-2}, ..., x_1) ... p_2(x_2 \mid x_1) p_1(x_1)$$

Spezialfall statistische Unabhängigkeit:

$$p(x_d, x_{d-1}, ..., x_2, x_1) = p_d(x_d) p_{d-1}(x_{d-1}) ... p_1(x_1)$$

Spezialfall Markov-Kette 1. Ordnung:

$$p(x_d, x_{d-1}, ..., x_2, x_1) = p_d(x_d \mid x_{d-1}) p_{d-1}(x_{d-1} \mid x_{d-2}) ... p_2(x_2 \mid x_1) p_1(x_1)$$

Satz von Bayes:

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

Denn: $p(y \mid x) p(x) = p(y, x) = p(x, y) = p(x \mid y) p(y)$

Überblick über statistische Datenmodellierungs-Verfahren

- **Techniken der Datenmodellierung**
- **Dichteschätzung**
- **Regression**
- **Klassifikation**

Techniken der Datenmodellierung

Ziel der statistischen Datenmodellierung:

- **Realistische Situation: Datensatz** $D = \left\{ \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(M)} \right\}$
  gezogen aus der unbekannten Verteilung $p(\mathbf{x})$

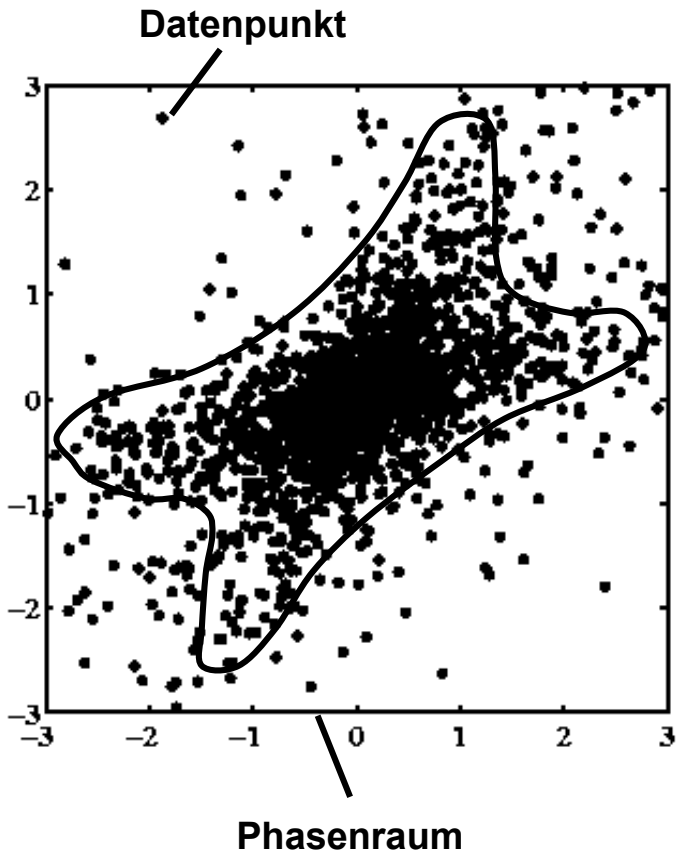
- **Datenvektoren** $\mathbf{x}$ **werden auch als „Datenpunkte“ oder „Muster“ bezeichnet**
- **Datenvektoren existieren in einem „Phasenraum“, „Zustandsraum“**
  (z. B. abstrakter Datenraum, Vektorraum, Hilbertraum ...)

- **Ziel: Extrahiere statistische Struktur aus den Daten...**
  und zwar durch Erstellung eines Modells der zugrundeliegenden Struktur.
- **Problem: Finden eines guten Modells... => Maschinelles Lernen (später)**
- **Wichtige Techniken: Dichteschätzung, Funktionsapproximation**

Dichteschätzung

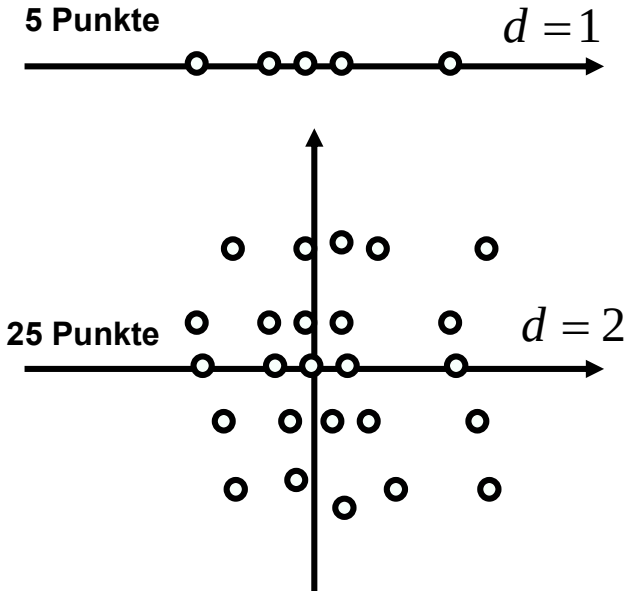
Bemerkungen

- **Schätzung der unterliegenden zusammengesetzten Wahrscheinlichkeitsdichte $p(\mathbf{x})$ aus den Daten**
- **Kenntnis der Dichte bedeutet vollständige statistische Charakterisierung!**
- **Charakterisierung der Struktur darin (z.B. Form Abhängigkeiten, Trends....)**



Bemerkungen (Fortsetzung)

- **Unüberwachtes Lernverfahren (Struktur wird nur aus den Daten extrahiert)**
- **Ein Dichteschätzer stellt ein „Generatives Modell“ dar (kann zur Datengenerierung benutzt werden)**
- **„Fluch der Dimensionen“: Im allgemeinen Fall steigt die Anzahl der benötigten Datenpunkte exponentiell mit der Dimension d des Problems an**
- **Man unterscheidet nichtparametrische und parametrische Dichteschätzung**



Fluch der Dimensionen

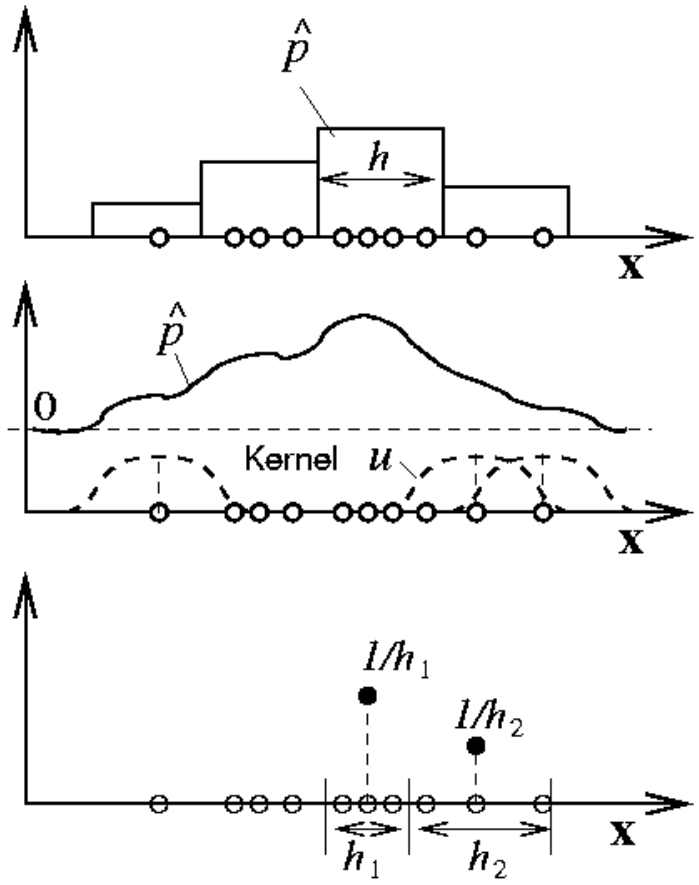
Nichtparametrische Dichteschätzung:

Es wird kein expliziter Funktionsverlauf für die Dichte $p(\mathbf{x})$ angenommen. Beispiele:

- ***Histogramm-Methode:***
  Teile Datenraum in Parzellen des Volumens h^d ein. Berechne darin die relativen Häufigkeiten

- ***Kernel-Dichteschätzer:***
  „Glättung der Datenwolke“
  $\hat{p}(\mathbf{x}) = 1/M \sum_{m=1}^{M} u((\mathbf{x}-\mathbf{x}^{(m)})/h)$
  $u(\mathbf{x}) \geq 0, \ \int u(\mathbf{x}) d\mathbf{x} = 1$ **Kernel**

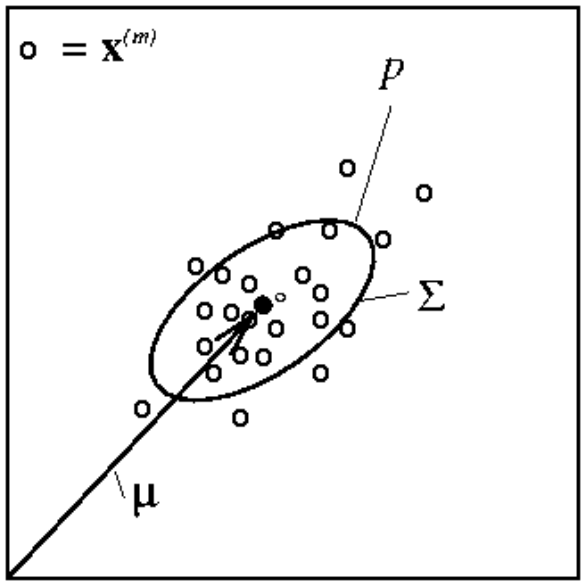
- ***K-nächste Nachbarn***
  Mittelung über K benachbarte Datenpunkte, kein festes h.



- **Bem: -- Größen h,K müssen geeignet gewählt werden (-> Regularisierung)**
  -- h, K, Histogrammeinträge ... können als Parameter aufgefasst werden

Parametrische Dichteschätzung:

- **Die Dichte wird als Dichtefunktion formuliert:** $p = p(\mathbf{x} \mid \mathbf{w})$
  -- Der Verlauf der Dichtefunktion wird durch einen Parametersatz $\mathbf{w}$ beschrieben
  -- $\mathbf{w}$ heißt Parametervektor, Modell, manchmal Gewichtsvektor (-> synapt: Gewichte)
- **Lernen = Optimierung des Parametervektors so, daß die Daten durch** $p(\mathbf{x} \mid \mathbf{w})$ **am besten beschrieben werden.**
- **Beachte Schreibweise als bedingte Wahrscheinlichkeit**
- **Vorteil: Wenige Parameter zu bestimmen**
- **Nachteil: Das Modell kann falsch/ungeeignet sein**
- **Beispiele:**
  -- Hauptkomponentenanalyse (PCA)
  -- Independent Component Analysis (ICA)
  -- Clustering
  -- Bayes-Belief Netze (frequent. Interpretation)
  -- Graphische Modelle



Beispiel: Gaußverteilung

$\mathbf{w} = (\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad p(\mathbf{x} \mid \mathbf{w}) = \varphi(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$

Funktionsapproximation

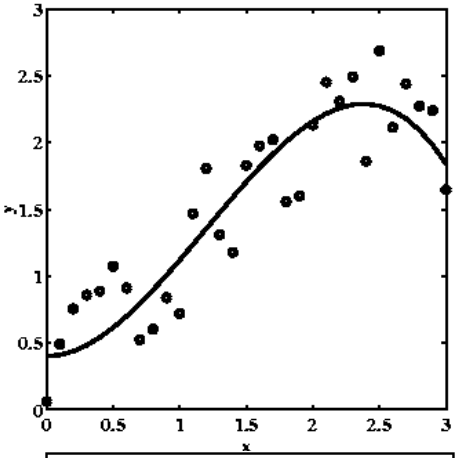
Oft lassen sich die Komponenten der Datenpunkte als Input und Output auffassen:

$\mathbf{x} \longrightarrow (x_1, x_2, ..., x_d, y) = \mathbf{x}, y$ **$\mathbf{x}$ = Input (Daten),** y **= Output (z.B. menschl. Bewertung)**

Bsp: Fertigungsparameter, Produktqualität. Man unterscheidet:

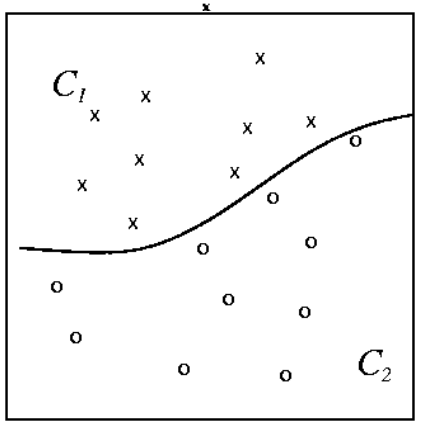
Regression

- **Oft reellwertiger/vektorwertiger Output**
- **Charakterisierung eines zugrundeliegenden funktionellen Zusammenhangs**



Klassifikation

- **Binärer oder kategorialer output (Klassenmitgliedschaft)**
- **Approximation der Klassenmitgliedschaft, oder**
- **Approximation der Wahrsch. für die Klassenmitgliedschaft**



Funktionsapproximation = überwachte Lernverfahren

Regression

Bemerkung

- **Implizit wird ein kausaler Zusammenhang zwischen In- und Output angenommen**

Formulierung

- **Erstelle Modell des zugrundeliegenden deterministischen (kausalen) Zusammenhangs zwischen input-output Paaren. Lerne bestes Modell**

$$\mathbf{y} = \mathbf{f}(\mathbf{x} \mid \mathbf{w}) + \mathbf{n}$$

f = Regressionsfunktion, parametrisiert durch w

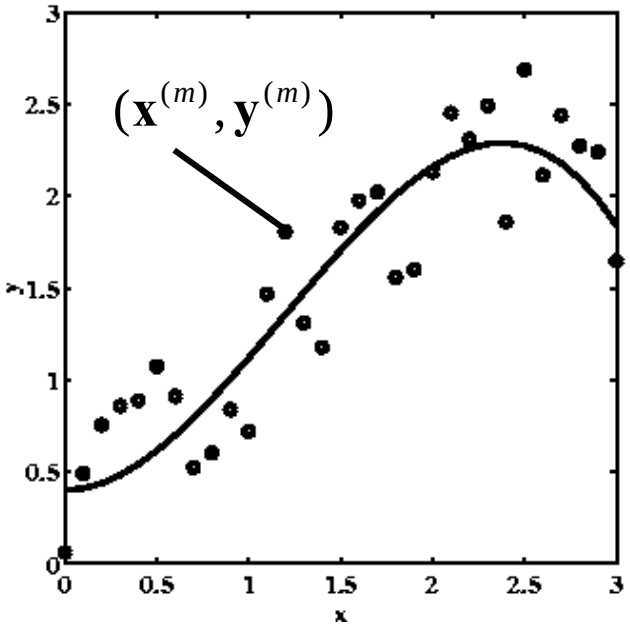
n beschreibt stochastisches Rauschen (Rauschvektor)

Beispiele:

-- Polynomfit, Lineares Modell... aber auch

-- Multilagen-Perceptron

-- Radiale Basisfunktionen-Netzwerke



Beispiel:

$$f(x) = 0.4 + x^2 - 0.28x^3$$

n **gleichverteilt in** [-0.5,0.5]

w = Vorfaktoren des Polynoms

Wahres Modell: (0.4, 0, 1, -0.28)

Verbindung zur parametrischen Dichteschätzung:

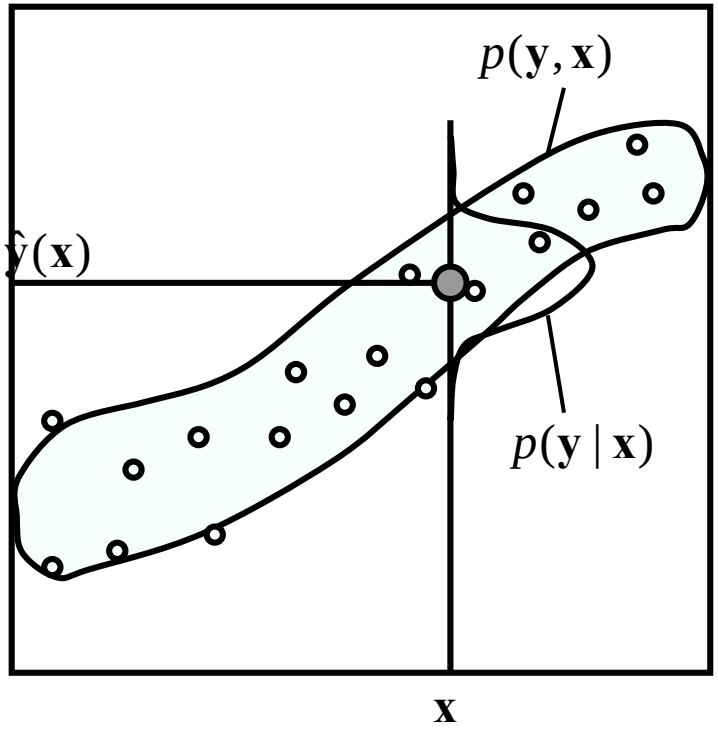
- **Schätze die zusammengesetzte Dichte** $p(\mathbf{y}, \mathbf{x} \mid \mathbf{w})$
- **Ansatz ($p_{\mathbf{n}}$=Verteilung des Rauschens):**

$$p(\mathbf{y}, \mathbf{x} \mid \mathbf{w}) = p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})\, p(\mathbf{x}) = p_{\mathbf{n}}(\mathbf{y} - \mathbf{f}(\mathbf{x} \mid \mathbf{w}))\, p(\mathbf{x})$$

- **Wenn der Mittelwert des Rauschens verschwindet, gilt:**

$$\hat{\mathbf{f}}(x) = \hat{\mathbf{y}}(\mathbf{x}) = \int p(\mathbf{y} \mid \mathbf{x})\mathbf{y}d\mathbf{y} = \int \frac{p(\mathbf{y}, \mathbf{x})}{p(\mathbf{x})}\mathbf{y}d\mathbf{y}$$

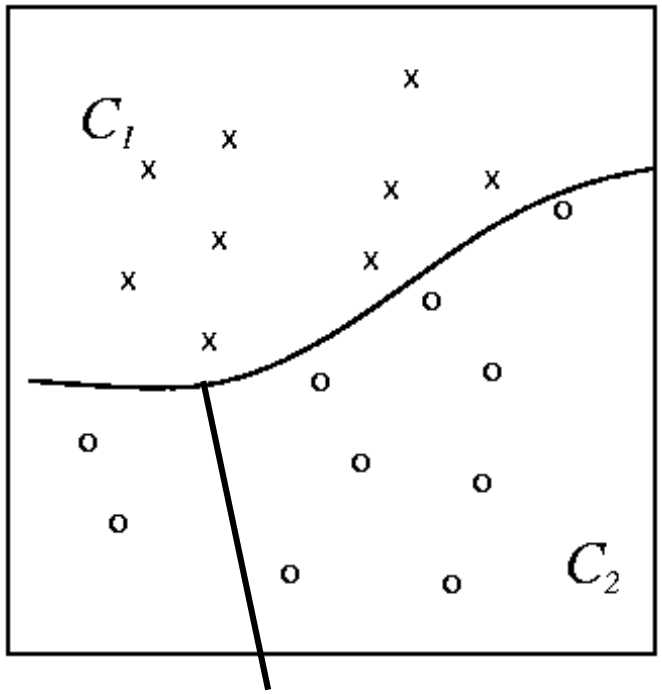
NB: Letzteres kann auch bei nichtparametrischer Dichteschätzung verwendet werden



Klassifikation

Bemerkungen

- **Datendimensionen werden in Input und (kategorialer) Klassenmitgliedschaft aufgespalten**
- **Output heißt auch Klassenlabel**



Separierende Hyperfläche

Formulierung

- **Erstelle Modell für die Wahrscheinlichkeiten der Klassenmitgliedschaft. Lerne bestes Modell**

$$\Pr(y(\mathbf{x}) = 1) = f(\mathbf{x} \mid \mathbf{w})$$

- f **= Klassifikator**
- **Separierende Hyperfläche definiert durch** $\Pr(y(\mathbf{x}) = 1) > \theta, \quad$ z.B. $\theta = 1/2$
- **Deterministischer Klassifikator entspricht „Regression“ einer binären Funktion**

Lineare Regression: Das lineare Modell

- **Problemstellung am Beispiel funktioneller Kernspintomographie (fMRI)**
- **Lineares Perceptron und Lineare Regression**
- **Parameterschätzung**
- **Varianzschätzung und Konfidenzintervalle**
- **Anwendungsbeispiel fMRI-Analyse:**
  Hirnaktivierung als Statistische Parameterkarten (SPM)

Anwendungsbeispiel: Analyse von fMRI-Daten

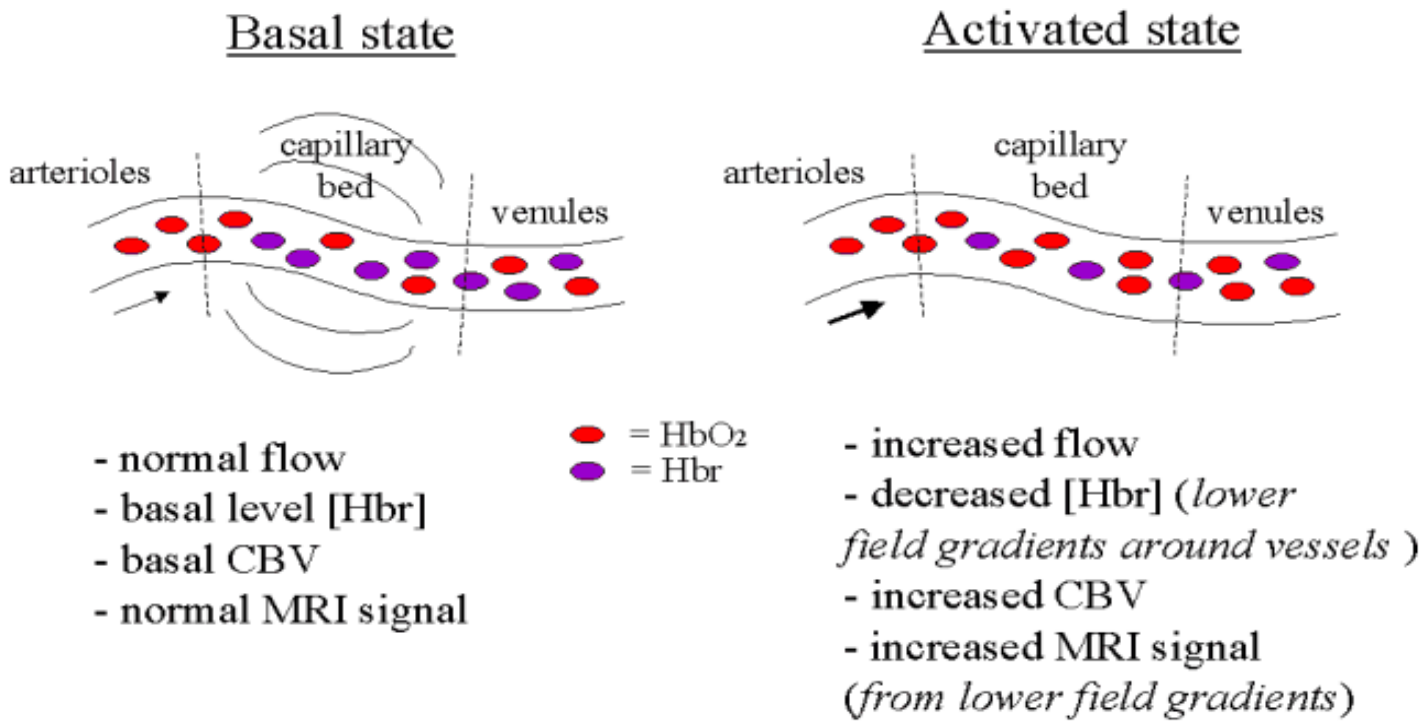
Technik: funktionelle Kernspintomographie

- **„functional Magnetic Resonance Imaging“ (fMRI)**
- **Magnetmomente der Atome werden in einem starken Magnetfeld angeregt (Magnetresonanz)**
- **Sie relaxieren in den Grundzustand zurück**
- **Die Relaxation wird durch Gradientenfelder ortsabhängig gemacht (=> Imaging)**
- **Die Relaxation ist abhängig von der molekularen Umgebung**
  -- Wasserstoffkonzentration: strukturelle MRI
  -- Paramagnetische Substanzen (Hbr): funktionelle MRI

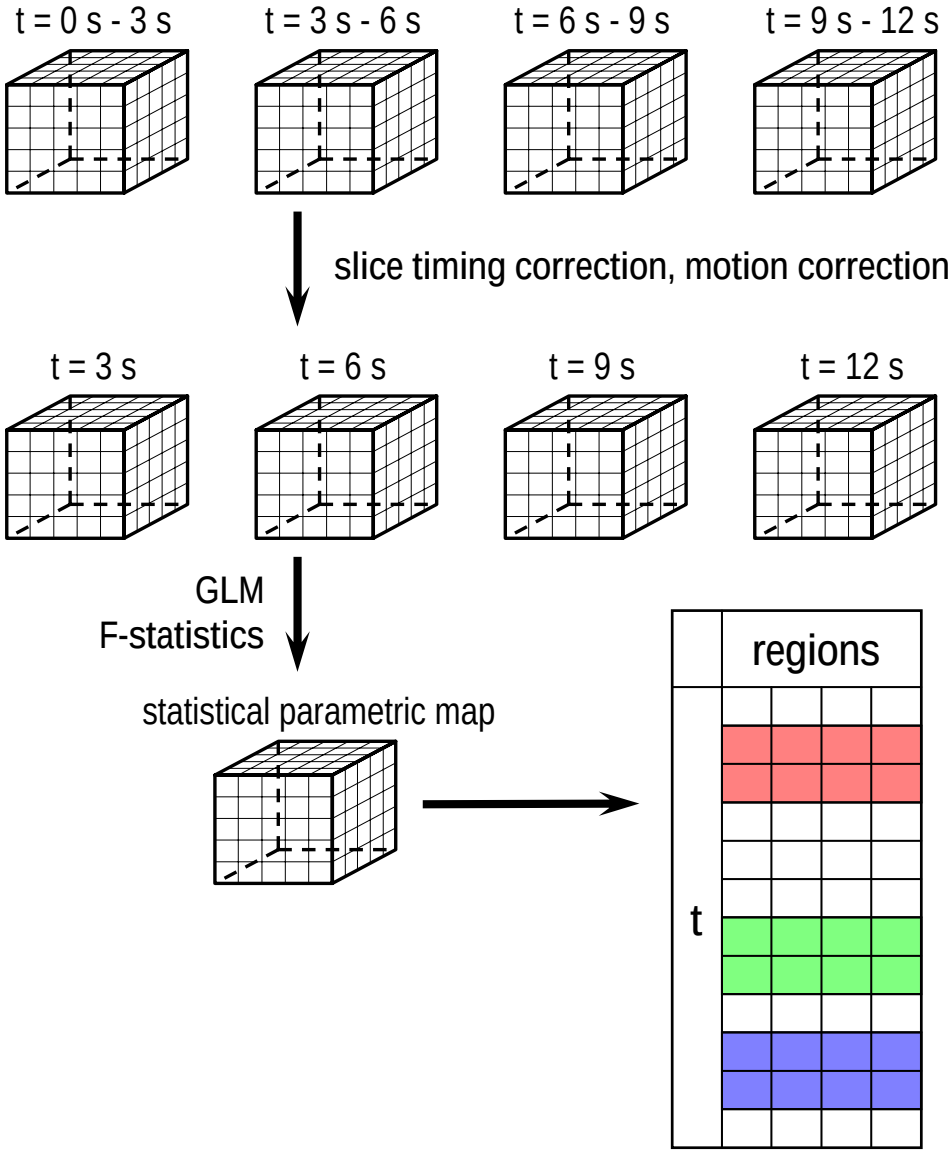
BOLD Signal

in fMRI experiments the BOLD signal (blood oxygen level dependent) is measured:

- **At least two components: local de-oxygenation and local increase in flow**



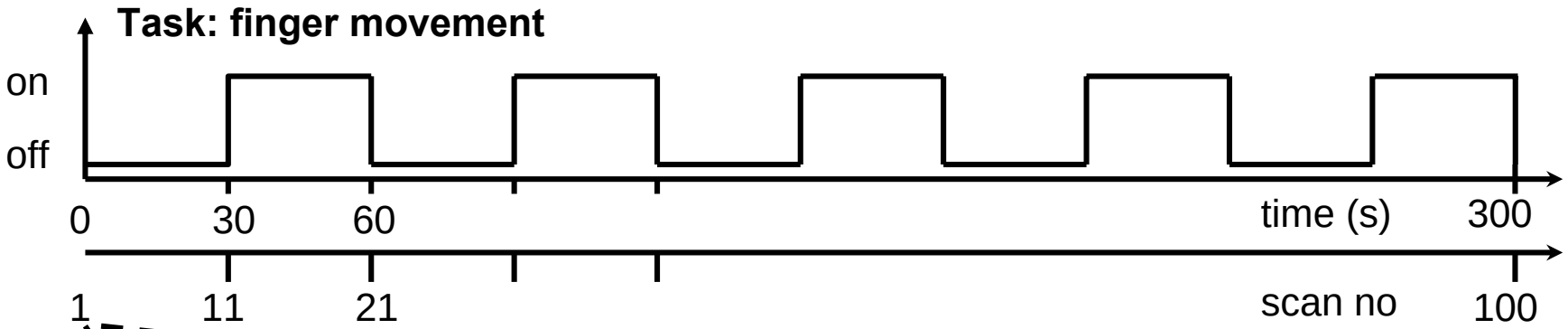
fMRI Data Format

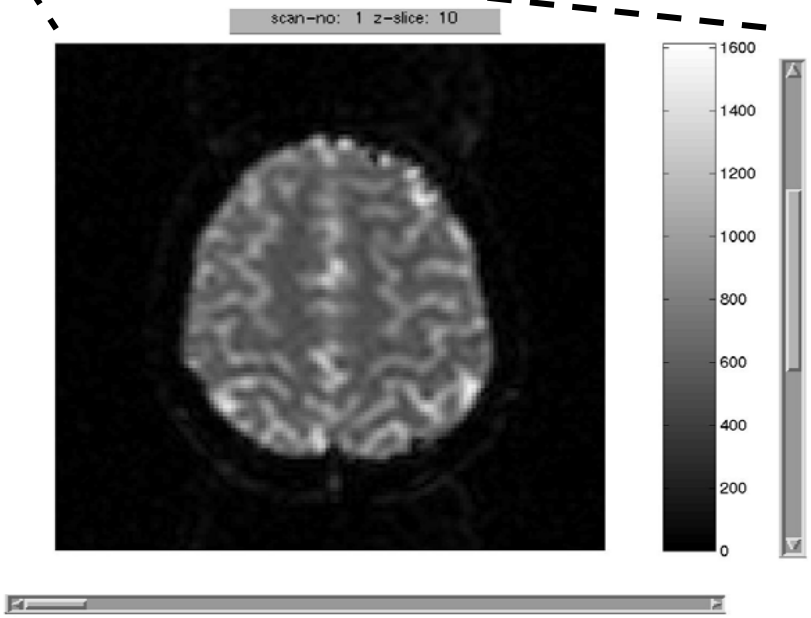


Task:

- **Extract small signal**
- **Separate signal components (BOLD from structural signal, background)**
- **Separate signal from noise**

Typical Experimental Task and Data Format



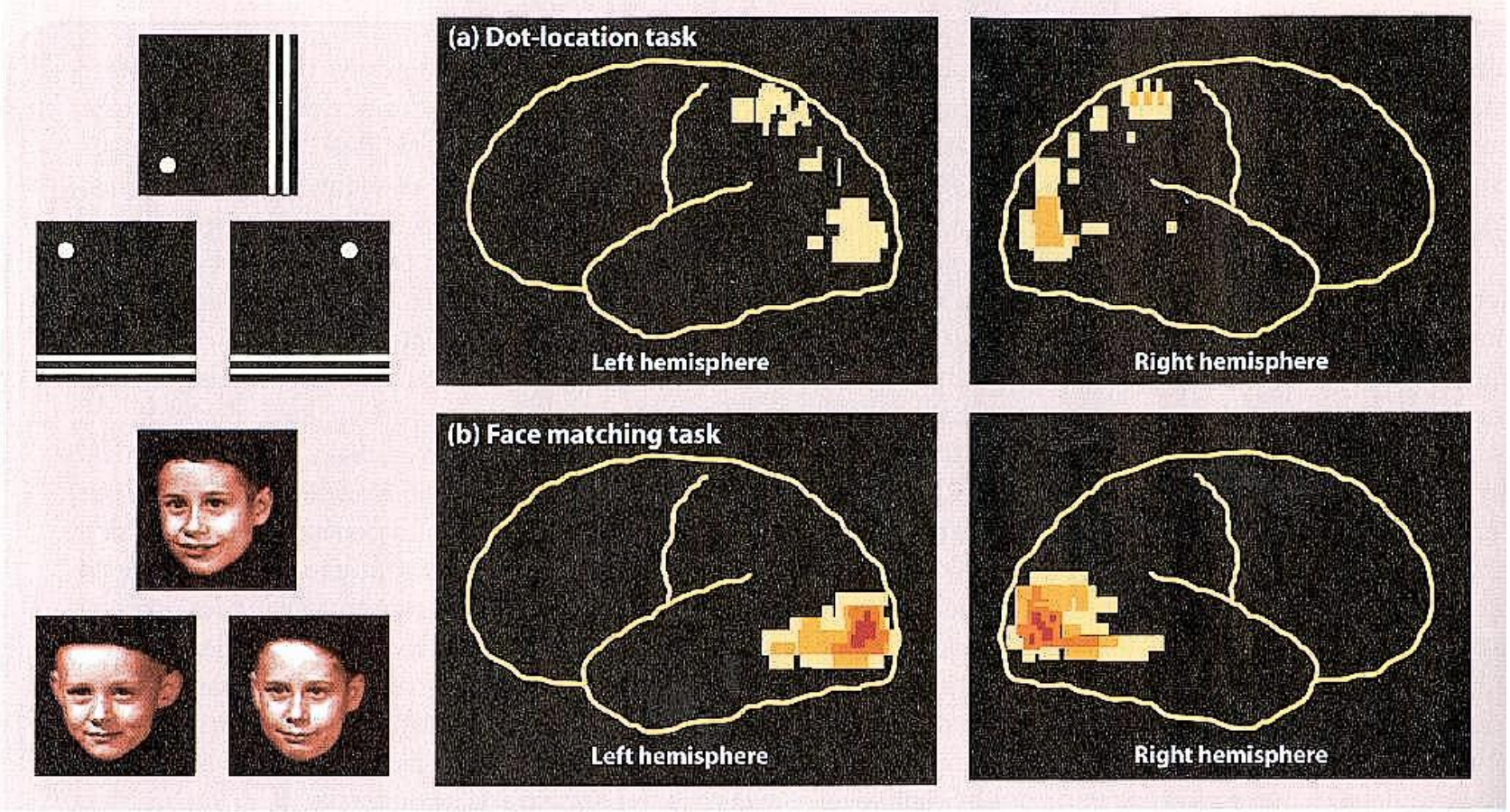


Raw data:

- **EPI scans**
- **100 scans,**
- **128x128 slice size, 16 slices**
- **Scan time: 1.6 sec**
- **Scan interval: 3 sec**

Raw data: scan 1, slice 10

Funktionelle Kernspintomographie: Ein Beispiel



(Haxby et al., 1994)

Das lineare Modell

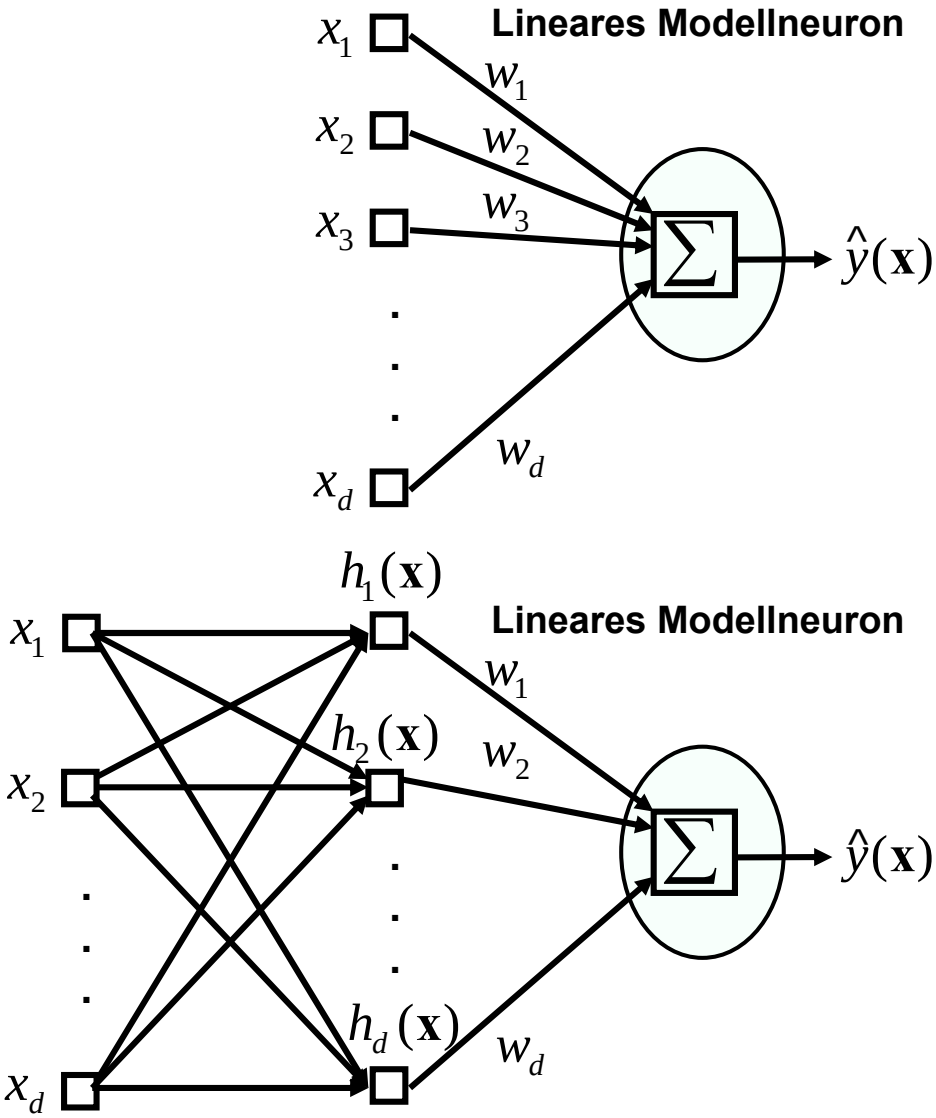
- **<u>Ausgangspunkt:</u> Lineares Perceptron**

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^{d} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

- **Fittet Ebene (mit Gradient $\mathbf{w}$)**
  Interessanter:
- **<u>Lineares Perceptron mit Vorverarbeitung</u>**

$$\hat{y}(\mathbf{x}) = h_1(\mathbf{x})w_1 + h_2(\mathbf{x})w_2 + \ldots + h_d(\mathbf{x})w_d$$
$$= \sum_{i=1}^{d} h_i(\mathbf{x})w_i$$

- **Perceptron mit VV ist linear in $\mathbf{w}$**
- **Aber: Nichtlinear in $\mathbf{x}$**
- **Funktionen $h(\mathbf{x})$ sind von Hand vorgegeben, werden nicht gelernt**



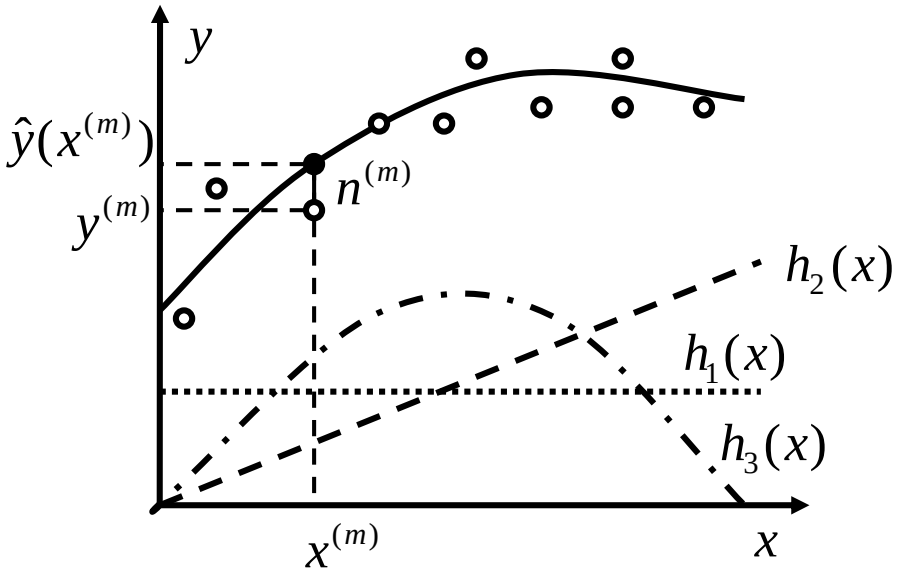
- **Geschrieben als Regressionsmodell**

Allgemein: $y = \hat{y} + n = f(\mathbf{x} \mid \mathbf{w}) + n$

Lineares Modell: $y = \sum_{i=1}^{d} h_i(\mathbf{x})\, w_i + n$

Bsp: Polynom-Fit:

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + \ldots\ w_n x^n$$



- **Empirische Formulierung, Design-Matrix:**

Betrachte Datensatz $D = \{(\mathbf{x}^{(m)}, y^{(m)}), m = 1,\ldots,M\}$

$$\begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(M)} \end{pmatrix} = \begin{pmatrix} h_1(\mathbf{x}^{(1)}) & \cdots & h_d(\mathbf{x}^{(1)}) \\ h_1(\mathbf{x}^{(2)}) & \ddots & h_d(\mathbf{x}^{(2)}) \\ \vdots & & \vdots \\ h_1(\mathbf{x}^{(M)}) & \cdots & h_d(\mathbf{x}^{(M)}) \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{pmatrix} + \begin{pmatrix} n^{(1)} \\ n^{(2)} \\ \vdots \\ n^{(M)} \end{pmatrix}$$

$$\mathbf{y} = \mathbf{H}\mathbf{w} + \mathbf{n}, \quad (\mathbf{H})_{mi} = h_i(\mathbf{x}^{(m)})$$

Output **Design-Matrix** **Parameter** **Rauschen**

- **„Lernen“: ML-Parameterschätzung des besten lin. Modells**

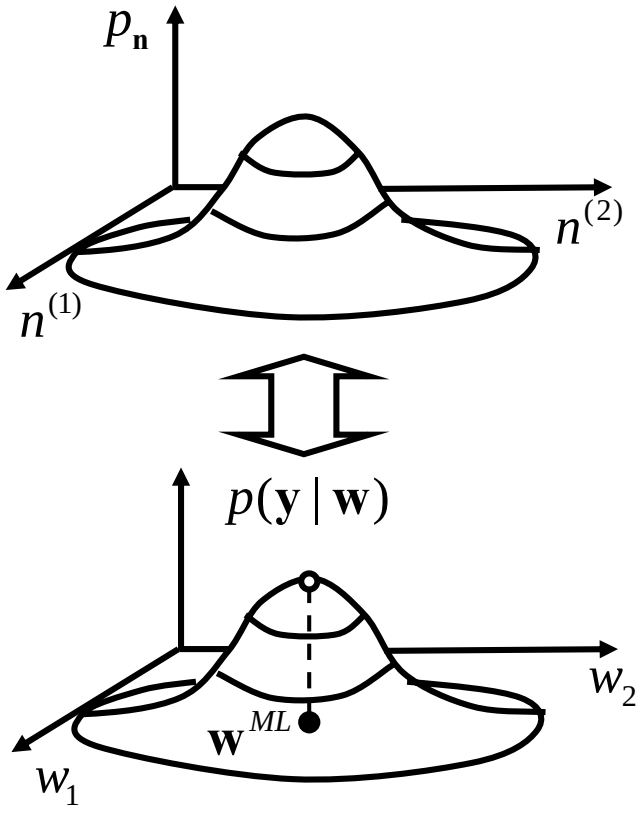
Likelihood: $p(D \mid \mathbf{w}) \equiv p(\mathbf{y} \mid \mathbf{w}) = p_{\mathbf{n}}(\mathbf{y} - \mathbf{H}\mathbf{w})$

Annahme: Gaussisches weißes Rauschen

$$p_{\mathbf{n}}(\mathbf{n}) = \prod_m p_n(n^{(m)}) = \prod_m \frac{1}{(2\pi\sigma_n^2)^{1/2}} \exp\left(-\frac{n^{(m)2}}{2\sigma_n^2}\right)$$

$$= \frac{1}{(2\pi\sigma_n^2)^{M/2}} \exp\left(-\frac{\mathbf{n}^T\mathbf{n}}{2\sigma_n^2}\right)$$

$$p(\mathbf{y} \mid \mathbf{w}) \propto \exp\left(-\frac{(\mathbf{y} - \mathbf{H}\mathbf{w})^T(\mathbf{y} - \mathbf{H}\mathbf{w})}{2\sigma_n^2}\right)$$



- **Maximum-Likelihood Parameter**

$$0! = \nabla_{\mathbf{w}} \ln p(\mathbf{y}|\mathbf{w}) = -\frac{1}{2\sigma_n^2} \nabla_{\mathbf{w}} (\mathbf{y}^T\mathbf{y} - 2\mathbf{w}^T\mathbf{H}^T\mathbf{y} + \mathbf{w}^T\mathbf{H}^T\mathbf{H}\mathbf{w})$$

$$= \frac{1}{\sigma_n^2}(\mathbf{H}^T\mathbf{y} - \mathbf{H}^T\mathbf{H}\mathbf{w})$$

$$= \frac{(\mathbf{H}^T\mathbf{H})}{\sigma_n^2}((\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y} - \mathbf{w})$$

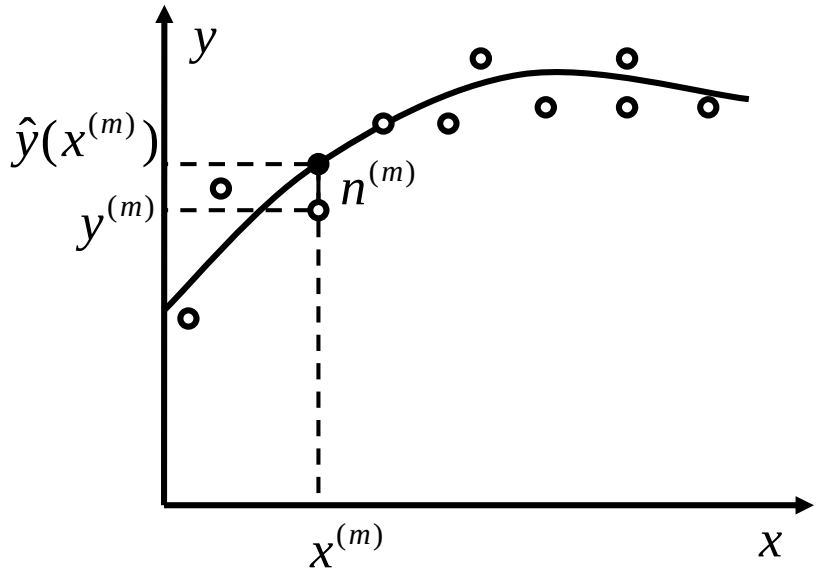
$$\Rightarrow \hat{\mathbf{w}} = \mathbf{w}^{ML} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$$

analytisch berechenbar!

- **Schätzung der Rauschvarianz :**

Schätzer des Rauschvektors:

$$\begin{aligned}\hat{\mathbf{n}} &= \mathbf{y} - \hat{\mathbf{y}} \\ &= \mathbf{y} - \mathbf{H}\hat{\mathbf{w}} \\ &= \mathbf{y} - \mathbf{H}(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}\mathbf{y} \\ &=: \mathbf{R}\mathbf{y}\end{aligned}$$



Residuum-erzeugende Matrix: $\mathbf{R} = \mathbf{I} - \mathbf{H}(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}$

Schätzer Rauschvarianz: $\hat{\sigma}_n^2 = \dfrac{\sum_{m=1}^{M} \hat{n}^{(m)2}}{M - (\approx \text{Anz. geschätzter Parameter})}$

$$\boxed{\hat{\sigma}_n^2 = \frac{\hat{\mathbf{n}}^T\hat{\mathbf{n}}}{tr(\mathbf{R})}}$$

- **Schätzung der Parametervarianz**

Beobachtung:

$$\nabla_{\mathbf{w}} \ln p(\mathbf{y}\,|\,\mathbf{w}) = \frac{(\mathbf{H}^T\mathbf{H})}{\sigma_n^2}((\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y} - \mathbf{w}) = \frac{(\mathbf{H}^T\mathbf{H})}{\sigma_n^2}(\hat{\mathbf{w}} - \mathbf{w}) = -\frac{(\mathbf{H}^T\mathbf{H})}{\sigma_n^2}(\mathbf{w} - \hat{\mathbf{w}})$$

$$\Rightarrow \ln p(\mathbf{y}\,|\,\mathbf{w}) = -\frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^T \frac{(\mathbf{H}^T\mathbf{H})}{\sigma_n^2}(\mathbf{w} - \hat{\mathbf{w}}) \equiv -\frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^T \Sigma_{\mathbf{w}}^{-1}(\mathbf{w} - \hat{\mathbf{w}})$$

$$\Rightarrow p(\mathbf{y}\,|\,\mathbf{w}) \propto \exp\left(-\tfrac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^T \Sigma_{\mathbf{w}}^{-1}(\mathbf{w} - \hat{\mathbf{w}})\right)$$

Parametervektor folgt einer multivariaten Gaußverteilung

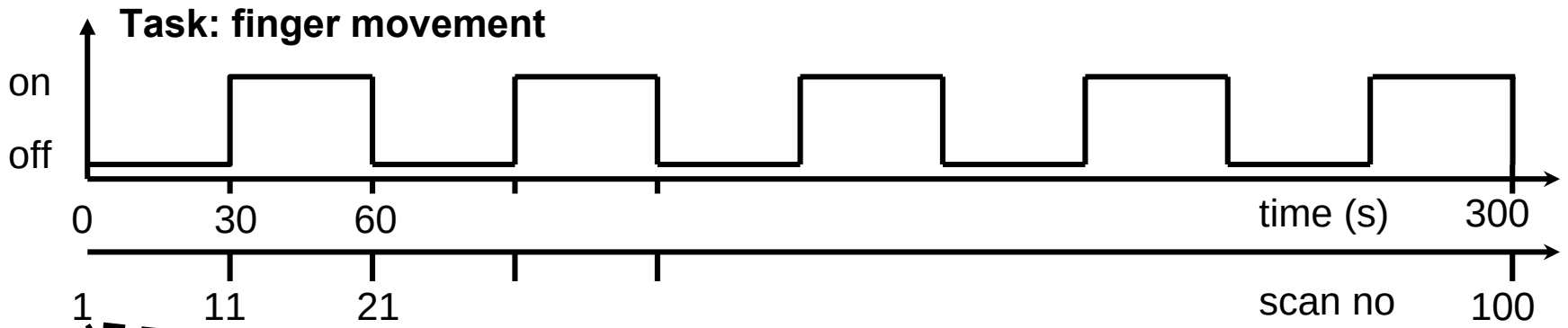
- **Kovarianzmatrix der Parameter:** $\Sigma_{\mathbf{w}} = \sigma_n^2(\mathbf{H}^T\mathbf{H})^{-1}$

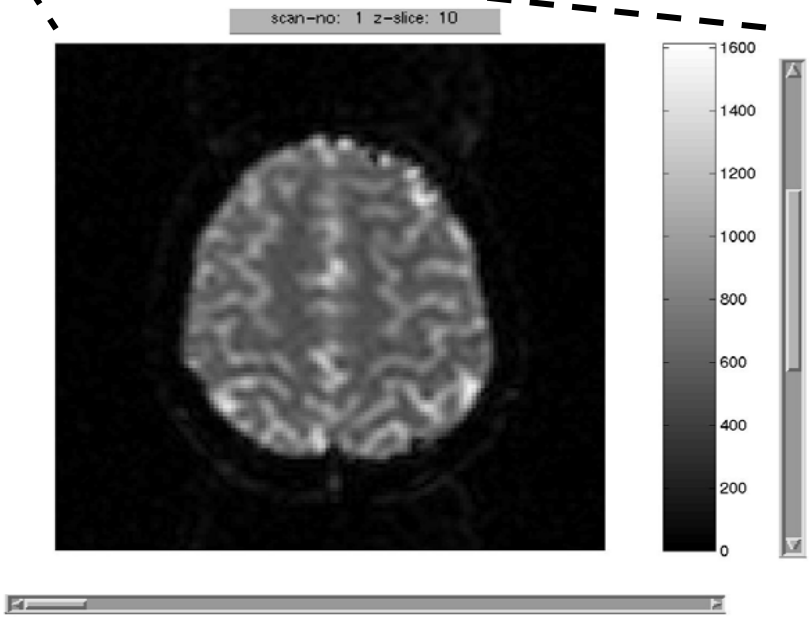
- **Varianz des i-ten Parameters:** $\sigma_i^2 = \left(\sigma_n^2(\mathbf{H}^T\mathbf{H})^{-1}\right)_{ii}$

- **Geschätztes Signal-Rausch-Verhältmis: Der Z-score**

$$Z_i =: \frac{\hat{w}_i}{\hat{\sigma}_i}^T = \frac{w_i}{\left(\Sigma_{\mathbf{w}}\right)_{ii}^{1/2}} = \frac{w_i}{\left(\hat{\sigma}_n^2\left(\mathbf{H}^T\mathbf{H}\right)^{-1}\right)_{ii}^{1/2}}$$

Linear Model Analysis of fMRI Time Series





Raw data:

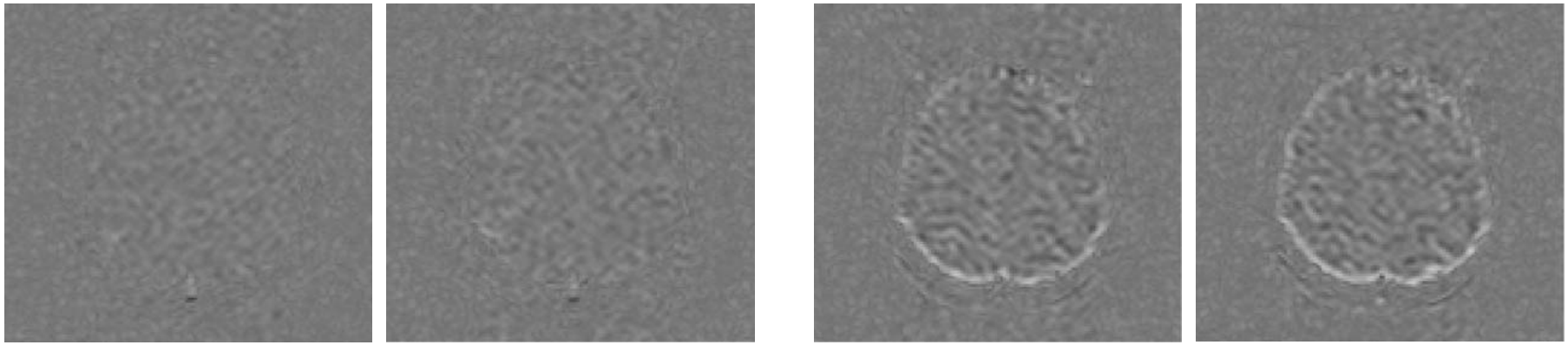
- **EPI scans**
- **100 scans,**
- **128x128 slice size, 16 slices**
- **Scan time: 1.6 sec**
- **Scan interval: 3 sec**

Raw data: scan 1, slice 10

fMRI Raw Images: Visual Inspection

After subtraction of second scan from the others (first frame has higher intensity)

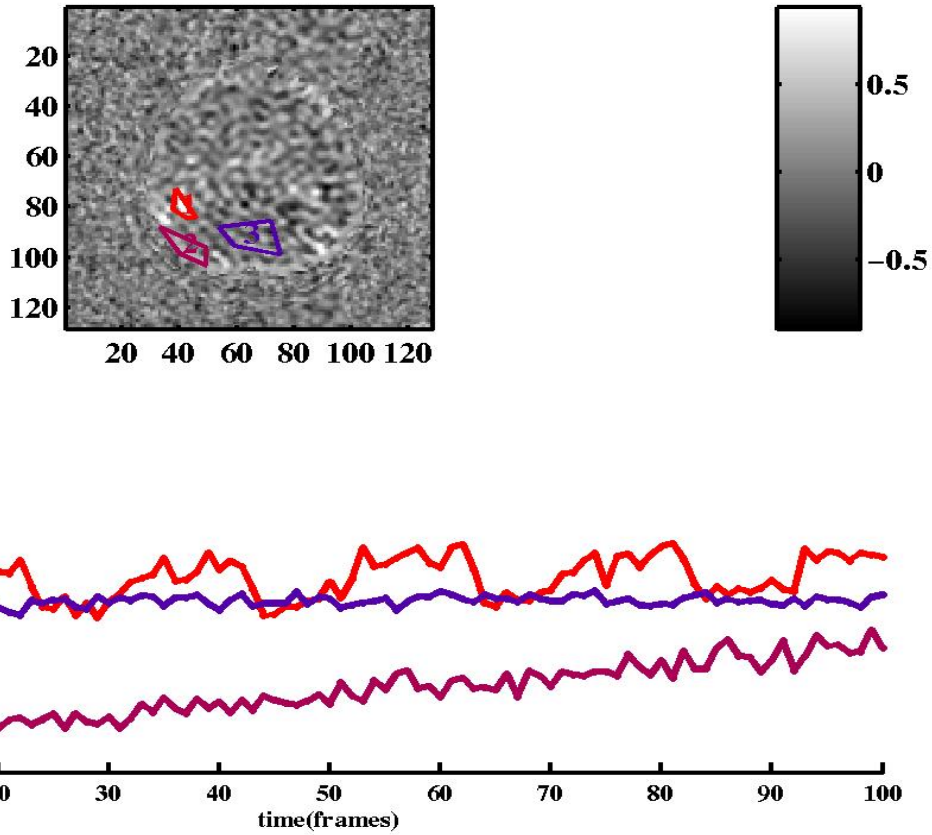
Scan 3, slice 10 **Scan 13, slice 10** **Scan 83, slice** **Scan 93, slice**



- **Strong correlated noise; correlations extend over several voxels**
- **From comparison of single scans with / without task no signal visible**
- **Slow drift in the head position (movement artifact in later scans)**

fMRI Time Series: Visual Inspection

Mean time series over three regions: Signal, drift, and background



- **Task-related response**
- **linear drift present**
- **First frame has higher values (magnetic transient?)**

General Linear Model: Design Matrix

Principle: $\mathbf{X} = \mathbf{Hw} + \mathbf{n}$

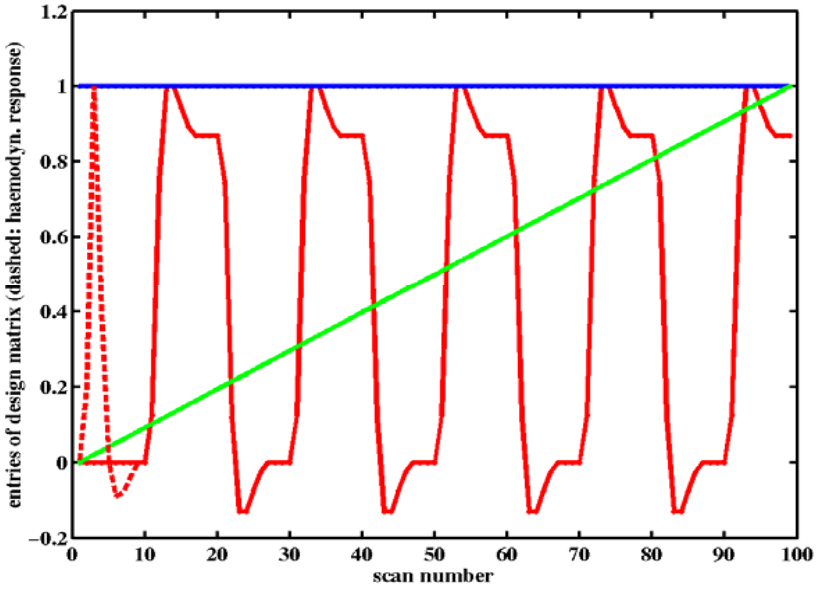
X = **matrix of time series**

H = **Design matrix**

w = **parameter (estimate used for t-test)**

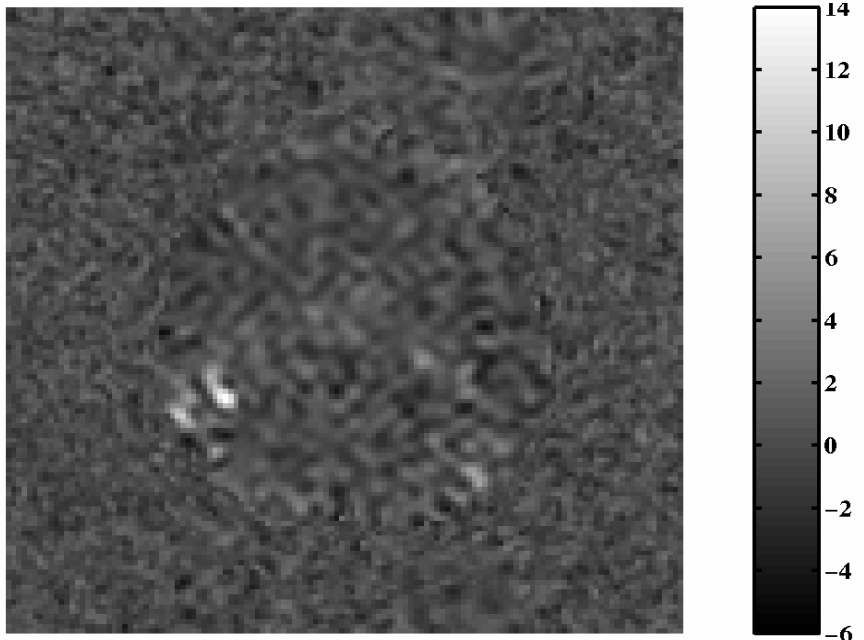
n = **noise (estimate used for t-test)**

Entries of design matrix used:



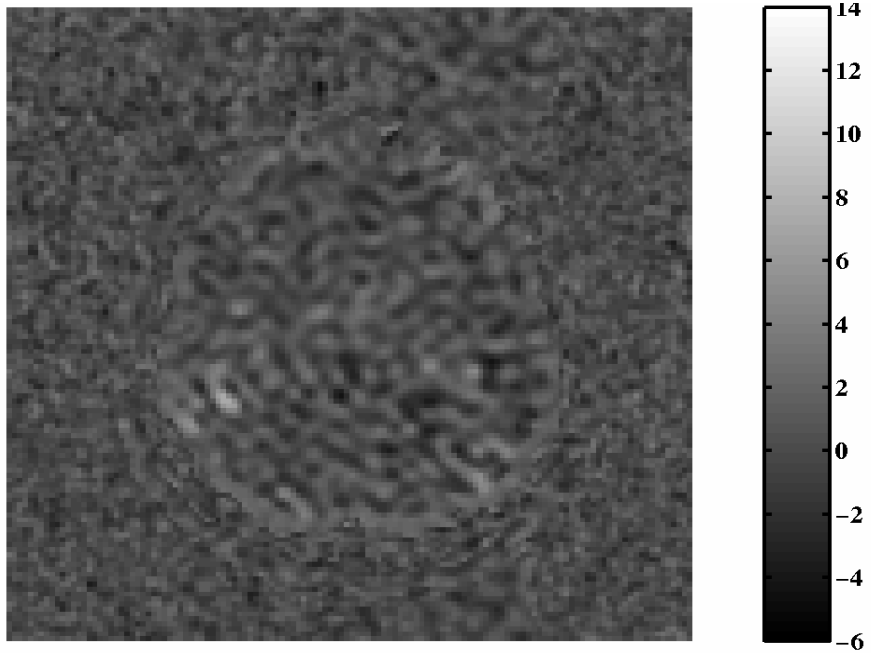
GLM: Statistical Parametric Maps (SPM)

3 component GLM on raw data



$w_1(x, y)$

Raw t-test on raw data



$t(x, y)$

- **Higher absolute t-value in GLM (due to convolution with hemodynamic response)**
- **t-values along head border are suppressed in GLM (because drift is modeled)**

Neuronale Verfahren zur Funktionsapproximation

- **Klassifikation mit dem Perceptron von Rosenblatt**
- **Vom Perceptron zum Multilagen-Perceptron**
- **Error-Backpropagation Lernregel**
- **Radiale Basisfunktionen-Netze**

Das Rosenblatt-Perceptron (1962)

Vorher: Regression. Jetzt: Binäre Klassifikation

- **Neuronale Struktur:**
  Das binäre Modell-Neuron

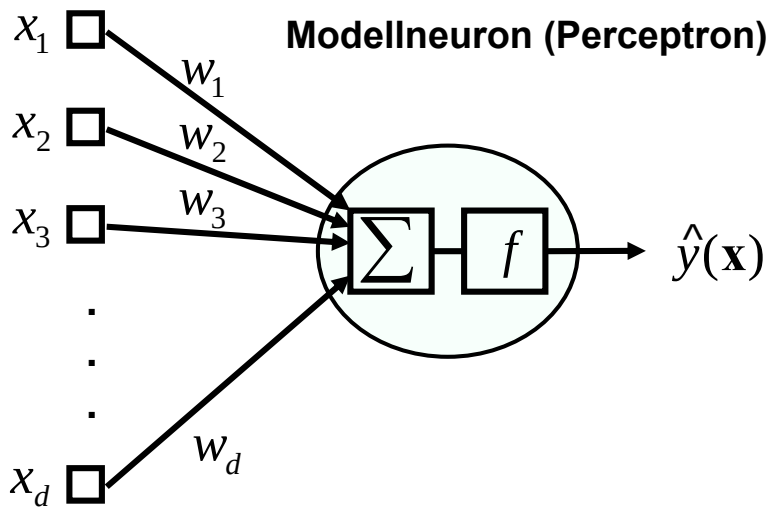
$$\hat{y}(\mathbf{x}) = \Theta(\mathbf{w}^T\mathbf{x} + b), \quad y = \hat{y} + n$$

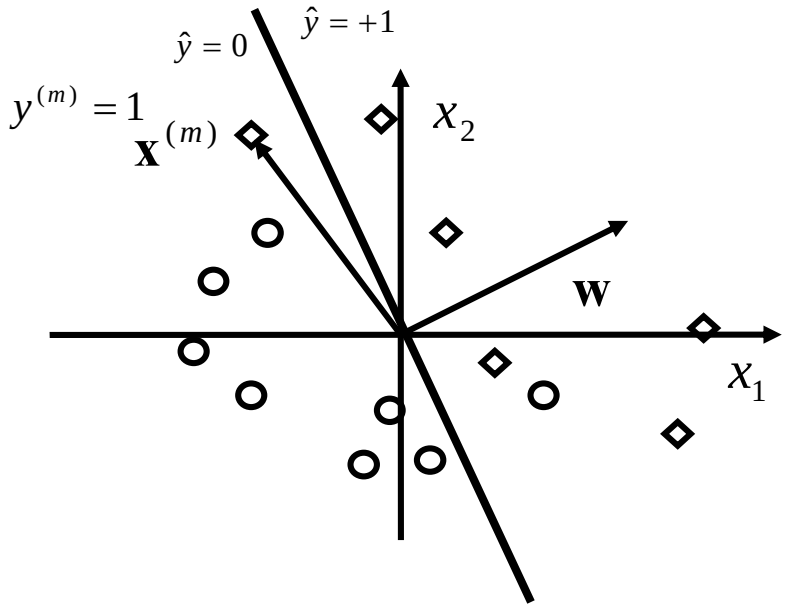
- **Wiederholung:**
  Das Perceptron kann als binärer Klassifikator dienen
  -- Geg: Daten in zwei Klassen, $y = 1, 0$
  -- Ges: Klassifikationsregel als Fkt. von $\mathbf{x}$
  -- Def: $x_{d+1} = 1,\ w_{d+1} = b \Rightarrow \hat{y} = \Theta(\mathbf{w}^T\mathbf{x})$

- **Lernen: Minimieren einer Fehlerfunktion. Hier: Gewichteter Falsch-Klassifikationsfehler**

- **Korrekte Klassifikation: Kein Fehler**
  Falsch-Klassifikation: positiver Fehler

$$F_P(w) = -\sum_{m=1}^{M} (y^{(m)} - \hat{y}(\mathbf{x}^{(m)}))\mathbf{w}^T\mathbf{x}^{(m)}$$





- **Perceptron-Lernregel:** **Gradientenabstieg**

$$\Delta \mathbf{w} = -\eta \nabla_{\mathbf{w}} F_P(\mathbf{w}) = \eta \sum_{m=1}^{M} (y^{(m)} - \hat{y}(\mathbf{x}^{(m)}))\, \mathbf{x}^{(m)}$$ **(Batch-Modus)**

Online-Lernregel:

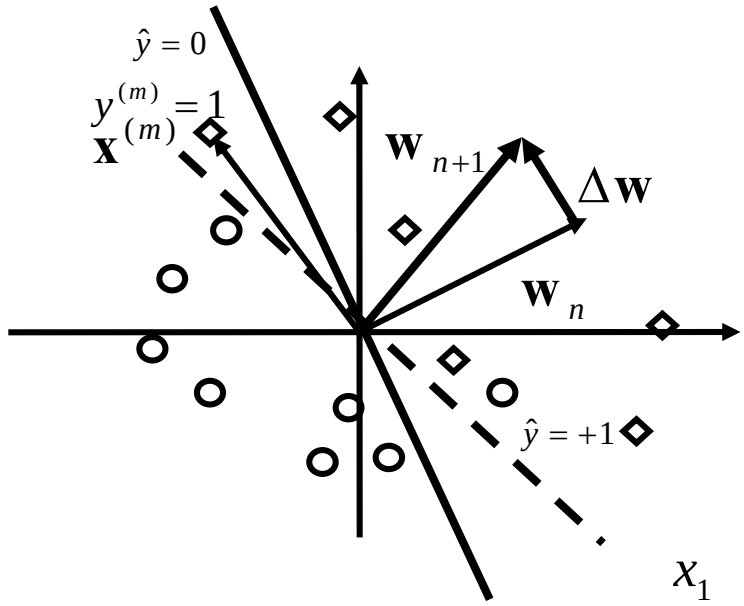
1. Wähle beliebiges $\mathbf{w}_0$

2. Wähle Muster m

3: $\mathbf{w}_{n+1} = \mathbf{w}_n + \eta(y^{(m)} - \hat{y}(\mathbf{x}^{(m)}))\, \mathbf{x}^{(m)}$

(d.h. tue nichts bei korrekter Klassifikation, biege $\mathbf{w}$ **bei Falsch-Klassifikation)**

4: Bis beste Klassifikation gehe zu 2



- **Bem:**

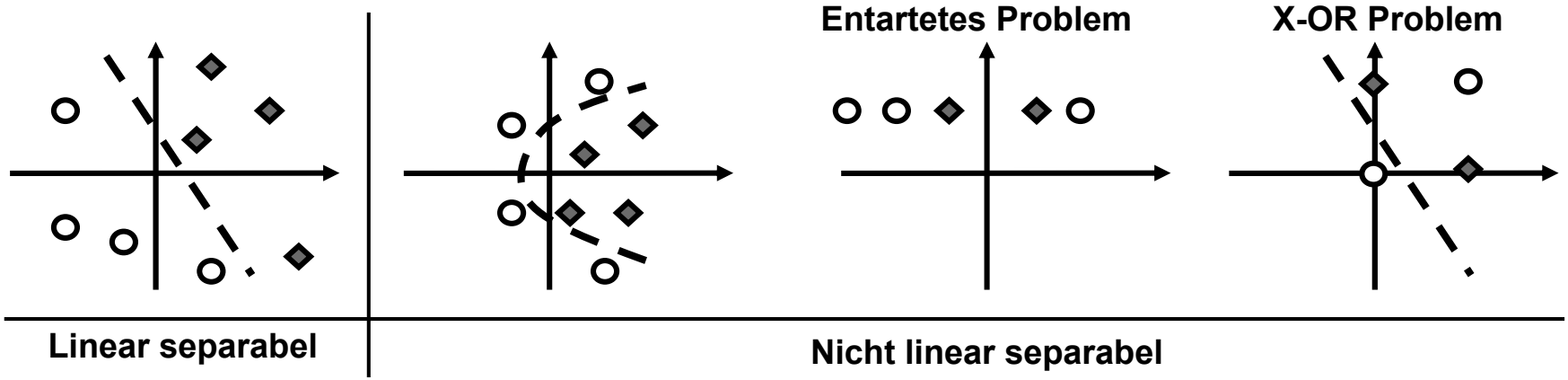
  -- Für linear separable Probleme Konvergenzgarantie in endl. vielen Schritten

  -- Funktioniert nur für linear separable Probleme

  -- Erweiterbar auf kontinuierliche Outputs

Lineare Separabilität:

Klassen können durch Hyperebene vollständig getrennt werden



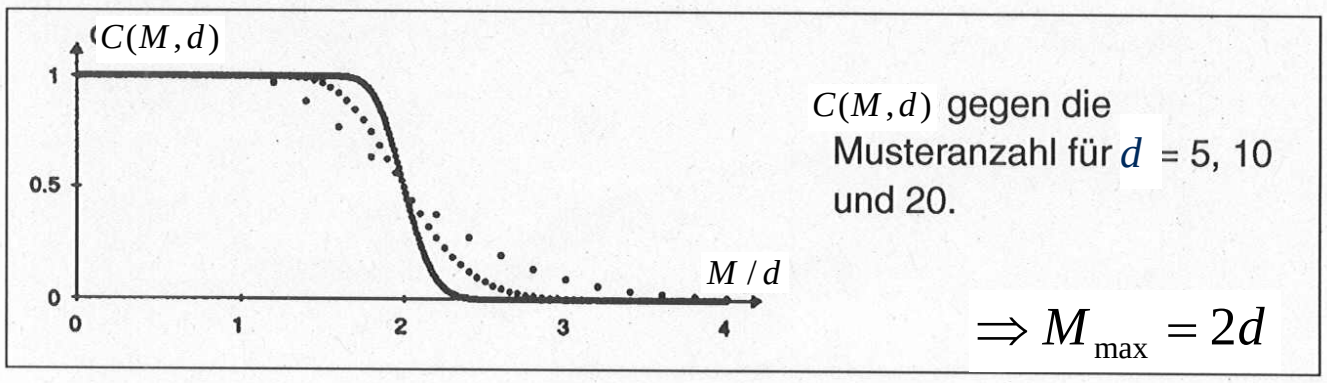
- **Wie wahrscheinlich sind M Muster in d Dimensionen linear separabel?**

  -- Wähle zufällig M d-dimensionale Muster

  -- Verteile zufällig Klassenlabels

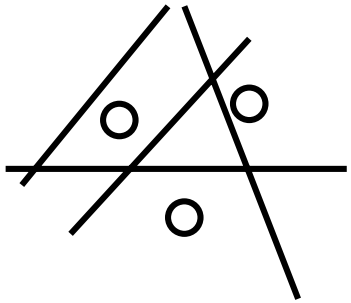
  -- Bestimme Wa. für lin. Separabilität

  -- Erg. f. hohe Dimensionen:



Behandlung nicht linear separabler Probleme 1: Dimensionalität

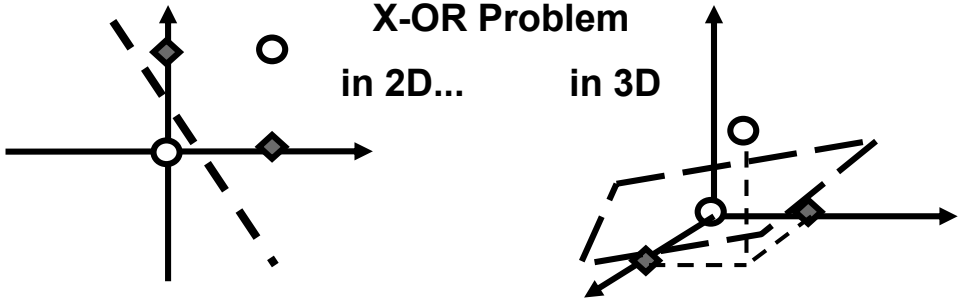
Lineare Klassifikation entspricht einer bestimmten Modellkomplexität
-- Wie komplex sind die Funktionen, die ein solches Modell implementieren kann?
--> „Zerschmettern“ von M Datenpunkten: Fähigkeit, alle 2^M möglichen Funktionen zu implementieren.



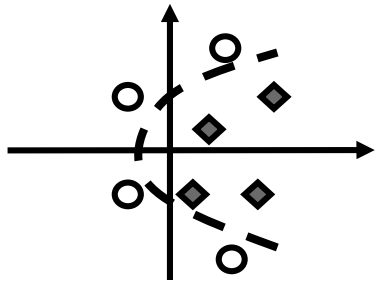
Def: Vapnik-Chervonenkis-Dimension (VC-Dimension) eines Modells:
Größtes M, das das Modell zerschmettern kann

VC-Dimension e. linearen Klassifikators in d Dimensionen: $d+1$

Lineare Klassifikation in höherer Dimension ... oder ...



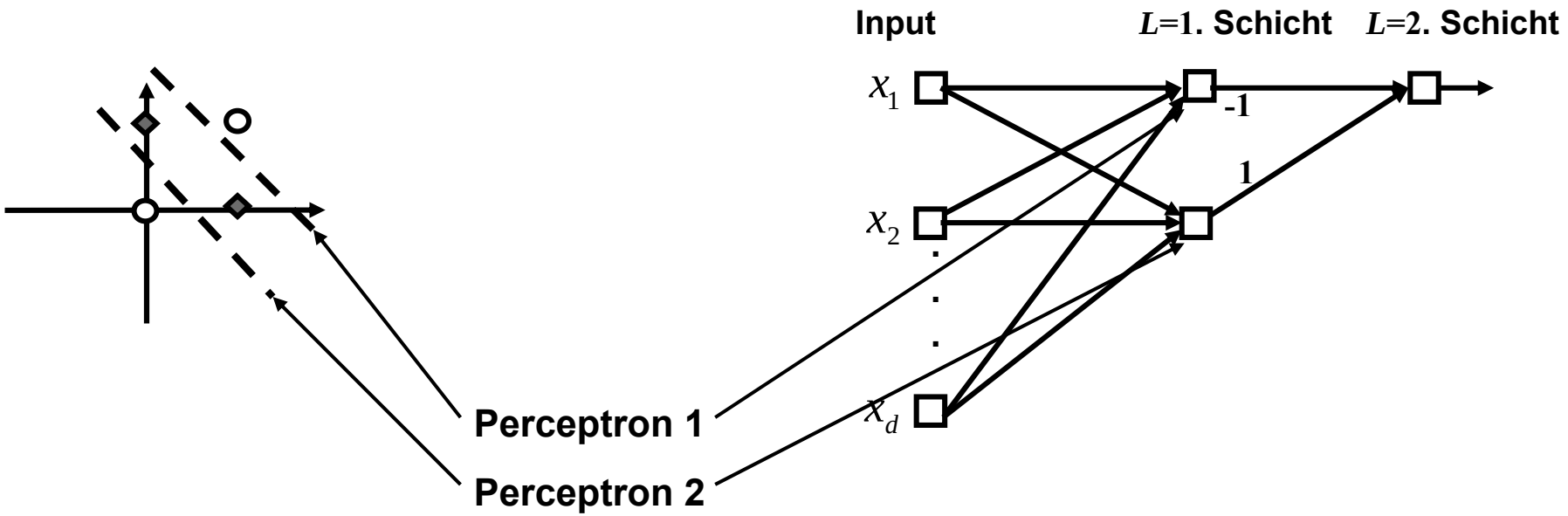
Nichtlineare Klassifikation



Vgl: Support Vector Machine (siehe später)

Vom Perceptron zum Multilagen-Perceptron

Behandlung nicht linear separabler Probleme 2: Mehrere Lagen



Perceptron:

- **Regression „komponentenweise nichtlinearer Funktionen“**
- **Klassifikation linear sparabler Probleme**

Multilagen Perceptron:

- **Allgemeine Funktionsapproximation**
- **Regression und Klassifikation**

Das Multilagen-Perceptron (MLP) für Regression

- **Motivation:**
  - **Nervenzellen im Gehirn sind hintereinandergeschaltet**
  - **Geschachtelte nichtlineare Transformationen sind mächtiger als eine einzige**

- **Aufbau**
  - **Feed-forward**
  - **Jedes Neuron gibt nichtlineare Funktion g des summierten Inputs weiter**

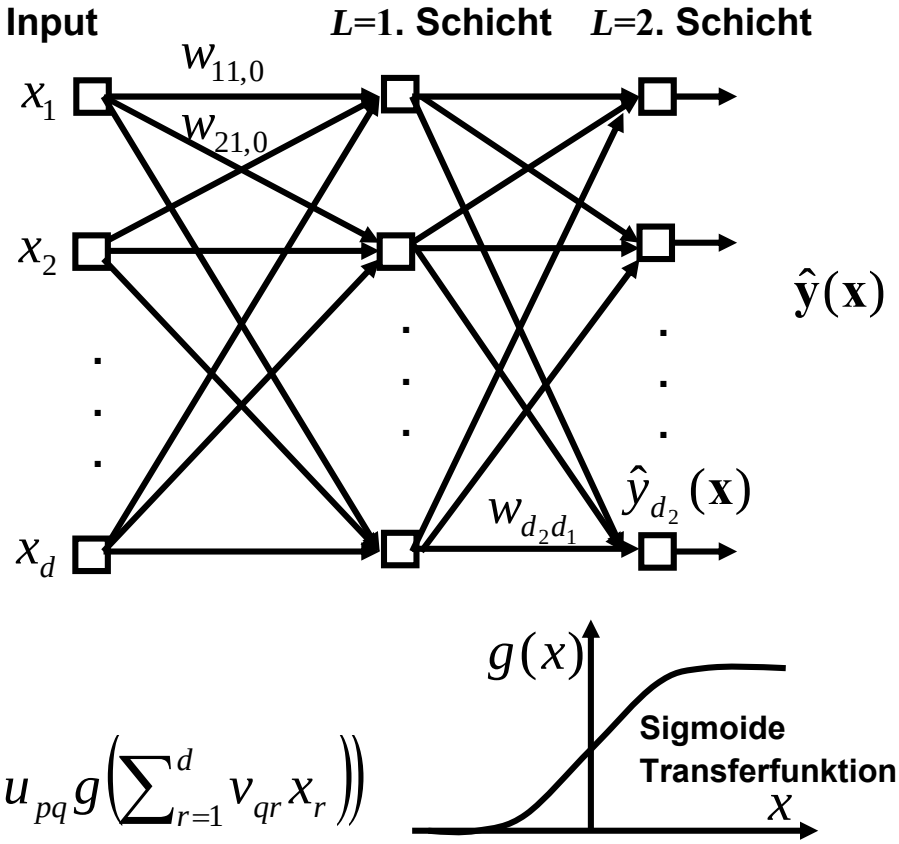
$$y_{i,0} = x_i, \quad d_0 = d$$

$$y_{i,L+1} = g\left(\sum_{j=1}^{d_L} w_{ij,L} y_{j,L} - \theta_i\right)$$

  - **mit** $d_L \to d_L + 1, \; y_{d_L+1,L} = -1, \; w_{id_L+1,L} = \theta_i$

$$y_{i,L+1} = g\left(\sum_{j=1}^{d_L} w_{ij,L} y_{j,L}\right)$$

  - **Bsp: zwei Schichten:** $\hat{y}_p \equiv y_{p,L=2} = g\left(\sum_{q=1}^{d_1} u_{pq} g\left(\sum_{r=1}^{d} v_{qr} x_r\right)\right)$



- **Als Regressionsmodell geschrieben**

$$y_i = \hat{y}_i(\mathbf{x}) + n_i = f_i(\mathbf{x} \mid \mathbf{w}) + n_i \equiv g\left(\sum_{j=1}^{d_L} w_{ij,L}\, g\left(\sum_{k=1}^{d_{L-1}} w_{jk,L-1} \ldots g\left(\sum_{q=1}^{d} w_{pq,0} x_q\right)\right)\right) + n_i$$

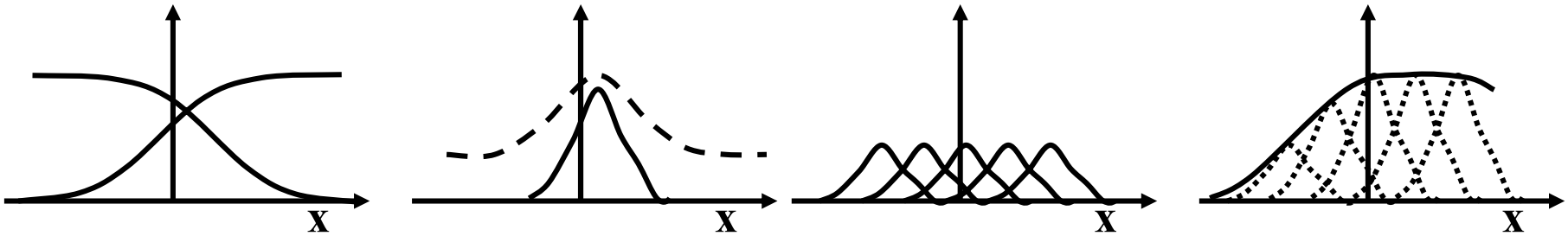
- **Wichtigkeit der Nichtlinearität: Falls g linear, entspricht MLP einschichtigem Fall**

$$y_i = \sum_{j=1}^{d_1} w_{ij,1} \sum_{k=1}^{d} w_{jk,0} x_k = \sum_{k=1}^{d} \left(\sum_{j=1}^{d_1} w_{ij,1} w_{jk,0}\right) x_k = \sum_{k=1}^{d} u_{ik} x_k$$

- **Universale Approximationseigenschaft**

  Ein dreischichtiges Netz kann jede beliebige kontinuierliche Funktion approximieren

  Anschaulich:

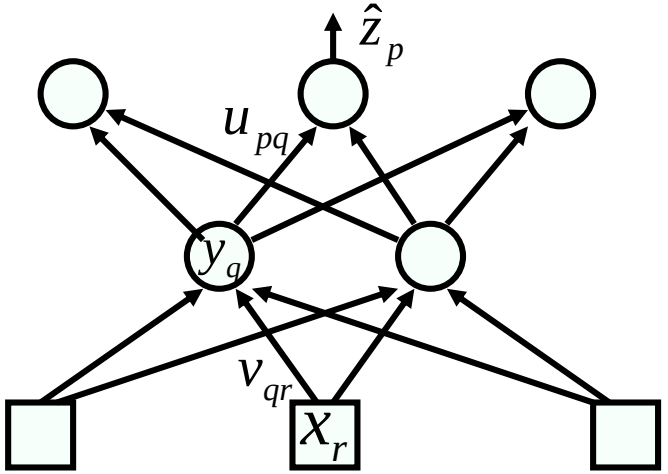


1. Schicht
Pool von kontinuierlichen Perceptrons

2. Schicht
Lokalisierte Antwort durch Kombination von Peceptron-Outputs plus sigmoide Transformation

3. Schicht
Kombination zur gewünschten Funktion

- **<u>Unklar</u>**
  - **Wieviele versteckte Neuronen werden benötigt**
  - **Wie sichert man sich gegen Überfitten ab? (=> Kreuzvalidierung)**
  - **Wie trainiert man die versteckten Neuronen?**
    => Error-Backpropagation-Algorithmus (Rumelhart, Werbos, `80er Jahre)



- **<u>Jetzt:</u> Betrachte zweischichtiges Netzwerk**

$$\hat{z}_p = g\left(\sum_q u_{pq} y_q\right) = g\left(\sum_q u_{pq} g\left(\sum_r v_{qr} x_r\right)\right)$$

- **Datensatz** $D = \{\mathbf{x}^{(m)}, \mathbf{z}^{(m)}\}, \; m = 1, \ldots, M$
- **Fehlerfunktion:**

$$F(\mathbf{w}) = \sum_m \sum_p (z_p^{(m)} - \hat{z}_p^{(m)})^2 = \sum_m \sum_p \left(z_p^{(m)} - g\left(\sum_q u_{pq} g\left(\sum_r v_{qr} x_r^{(m)}\right)\right)\right)^2, \quad \mathbf{w} \equiv (\mathbf{u}, \mathbf{v})$$

- **Modular:** $F(\mathbf{w}) = \sum_m E^{(m)}(\mathbf{w})$ $\quad E^{(m)}(\mathbf{w}) = \sum_p \left(z_p^{(m)} - g\left(\sum_q u_{pq} g\left(\sum_r v_{qr} x_r^{(m)}\right)\right)\right)^2$

- **Backpropagation Algorithmus: Herleitung**

  --> Geschickte Anwendung des Gradientenabstiegs $\Delta \mathbf{w} = -\eta \nabla_{\mathbf{w}} E(\mathbf{w})$

  - **Def: Inputs** $a_p = \sum_q u_{pq} y_q, \quad b_q = \sum_r v_{qr} x_r$
  - **Def. Fehler:** $E(\mathbf{w}) = \sum_p \left(z_p - g\left(\sum_q u_{pq} g\left(\sum_r v_{qr} x_r \right) \right) \right)^2 =$

    (ohne index *m*)

    $$= \sum_p \left(z_p - g\left(\sum_q u_{pq} g(b_q) \right) \right)^2 = \sum_p \left(z_p - g(a_p) \right)^2 = \sum_p \left(z_p - \hat{z}_p \right)^2$$
  - **Partielle Ableitungen für Versteckt-zu-Output Gewichte**

$$-\frac{\partial E}{\partial u_{ij}} = 2(z_i - \hat{z}_i)\frac{\partial \hat{z}_i}{\partial u_{ij}} = 2\underbrace{(z_i - \hat{z}_i) g'(a_i)}_{\delta_i} \frac{\partial a_i}{\partial u_{ij}} = \delta_i y_j$$

$$\delta_i = 2(z_i - \hat{z}_i) g'(a_i)$$

„Error“

  - **Partielle Ableitungen für Input-zu-Versteckt Gewichte**

$$-\frac{\partial E}{\partial v_{jk}} = \sum_p \underbrace{2(z_p - \hat{z}_p) g'(a_p)}_{\delta_p} \frac{\partial a_p}{\partial v_{jk}} = \sum_p \delta_p \frac{\partial}{\partial v_{jk}} \left(\sum_q u_{pq} g\left(\sum_r v_{qr} x_r \right) \right)$$

$$= \underbrace{\sum_p \delta_p u_{pj} g'(b_j)}_{=:\delta_j} \underbrace{\frac{\partial b_j}{\partial v_{jk}}}_{x_k} = \delta_j x_k,$$

$$\delta_j = \sum_p \delta_p u_{pj} g'(b_j)$$

„Error-Backpropagation“

- **Backprop-Algorithmus:**

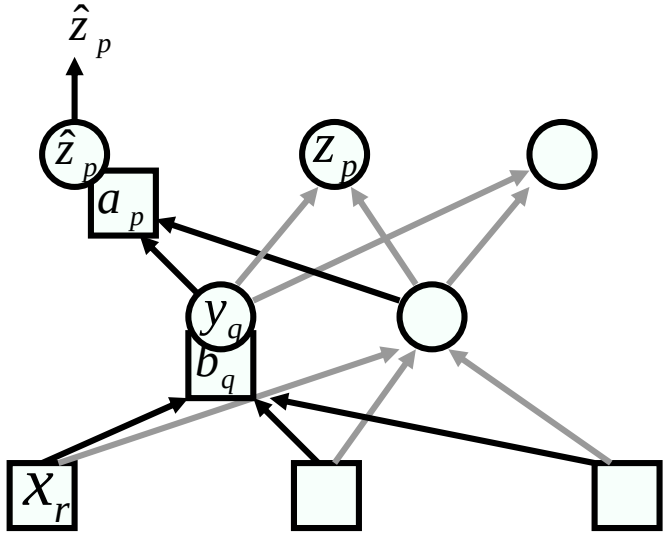
  Nach Zufallsbelegung der Gewichte auf kleine Werte

  - **Präsentiere Muster** $\mathbf{x}^{(m)}$

  Signal forward-propagation:

  - **Berechne und speichere von jeder Schicht die Inputs und Aktivitäten, also**

$$b_q = \sum_r v_{qr} x_r^{(m)}, \quad y_q = g(b_q)$$
$$a_p = \sum_r u_{pq} y_q, \quad \hat{z}_p = g(a_p)$$



  - **Berechne Fehler an der Outputschicht**

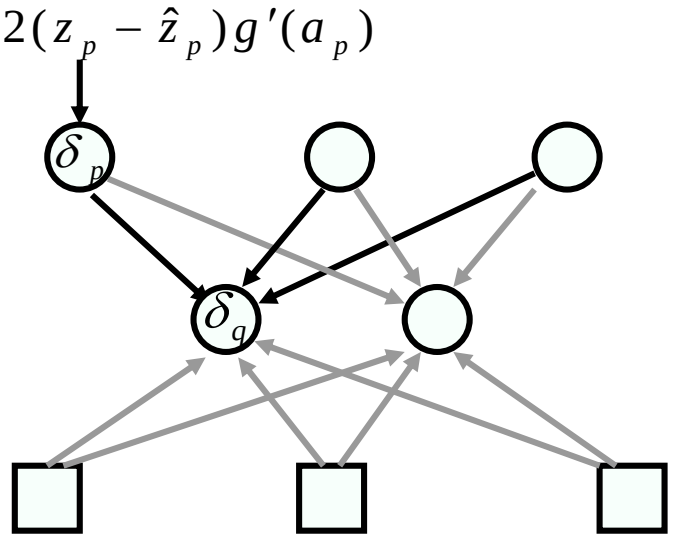
$$\delta_p = 2(z_p - \hat{z}_p) g'(a_p)$$

  Error-Backpropagation:

  - **propagiere Fehler durch versteckte Schichten zurück**

$$\delta_q = \sum_p \delta_p u_{pq} g'(b_q)$$

  - **Lerne mit Regel:** $\Delta u_{ij} = -\eta \frac{\partial E}{\partial u_{ij}} = \eta \delta_i y_j, \quad \Delta v_{jk} = \eta \delta_j x_k$



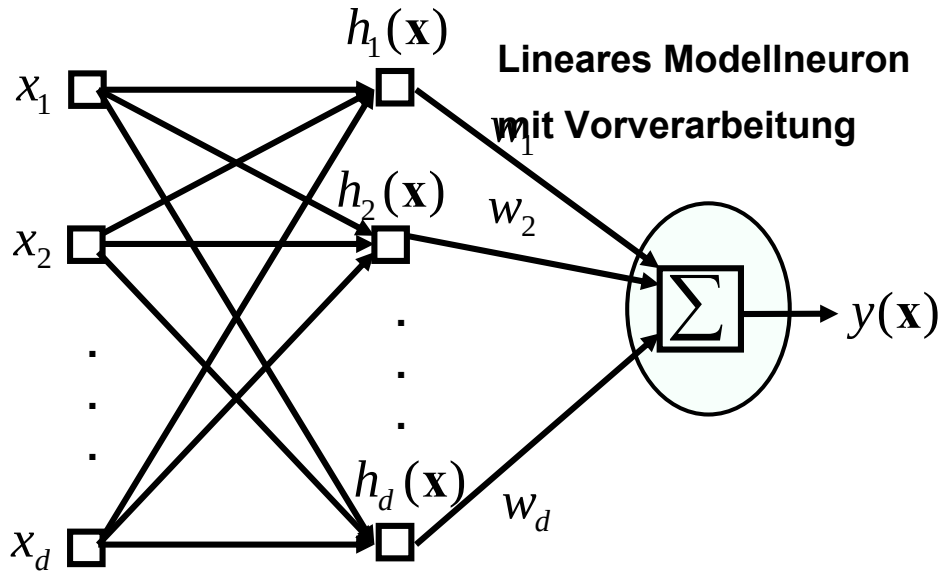
Wiederholung: Das lineare Modell

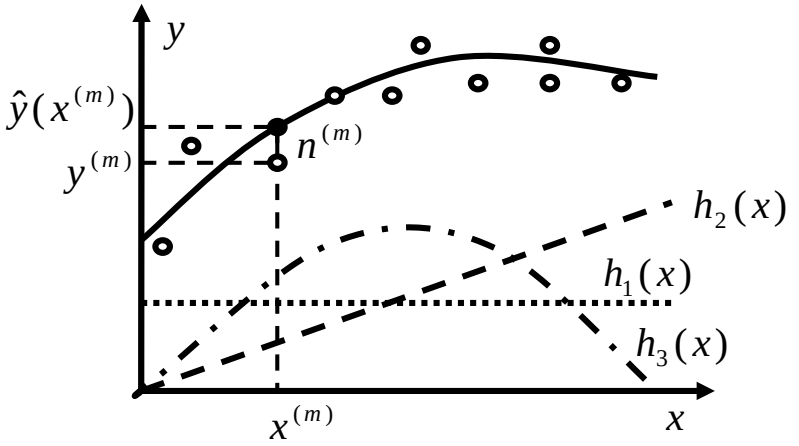
$$y(\mathbf{x}) = h_1(\mathbf{x})w_1 + h_2(\mathbf{x})w_2 + \ldots + h_d(\mathbf{x})w_d$$
$$= \sum_{i=1}^{d} h_i(\mathbf{x})w_i$$
$$\mathbf{y} = \mathbf{H}\mathbf{w} + \mathbf{n}, \quad (\mathbf{H})_{mi} = h_i(\mathbf{x}^{(m)})$$

- **Funktionen $\mathbf{h(x)}$ sind von Hand vorgegeben, werden nicht gelernt**
- **Aber: Einfache Lösung**

$$\Rightarrow \hat{\mathbf{w}} = \mathbf{w}^{ML} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$$

analytisch berechenbar!





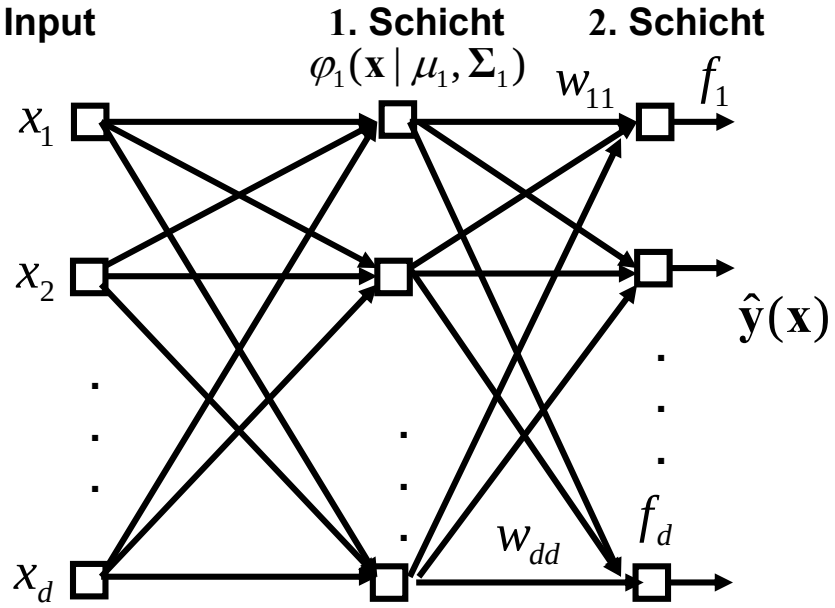
Radiale Basisfunktionen (RBF) Netzwerke

- **Motivation:**
  - **Hybrid zwischen linearem Modell und universalem Funktionsapproximator**
  - **Approximiere Output als lineare Summe nichtlinearer (Gauss-) Funktionen**

- **Aufbau**
  - **1. Schicht Gauss-Funktionen (anstatt $h(\mathbf{x})$) approximieren Segmente des Inputs**

    $\varphi_j(\mathbf{x} \mid \mu_j, \Sigma_j)$
  - **2. Schicht: Linearkombination der Gaussfunktionen**

    $$\hat{y}_i = f_i(\mathbf{x}) = \sum_{j=1}^{d_1} w_{ij} \varphi_j(\mathbf{x} \mid \mu_j, \Sigma_j)$$



- **Bem: -- 1. Schicht nichtlinear, aber Gaussisch => effizient zu optimieren**
  -- 1. Schicht entspricht Schicht 1,2 des universalen Approximators
  -- 2. Schicht linear => einfache analytische ML Lösung

- **Optimierung**

  - **Optimiere μ,Σ der Gaussfunktionen nur basierend auf Input-Daten**
    -- Entspricht Mixture of Gaussian Dichteschätzer

$$\Phi(\mathbf{x} \mid \boldsymbol{\mu}^{ML}, \boldsymbol{\Sigma}^{ML}) := (\varphi_1(\mathbf{x} \mid \mu_1^{ML}, \boldsymbol{\Sigma}_1^{ML}), \ldots, \varphi_{d_1}(\mathbf{x} \mid \mu_{d_1}^{ML}, \boldsymbol{\Sigma}_{d_1}^{ML}))^T = \Phi(\mathbf{x})$$

  - **Verwende optimale Gaussfunktionen als Modellfunktionen eines linearen Modells**

$$\mathbf{y}(\mathbf{x}) = \mathbf{W}\Phi(\mathbf{x}) + \mathbf{n}$$

Def: -- $\Phi_{mi} = \varphi_i(\mathbf{x}^{(m)})$

-- $\mathbf{Y}_{mi} = y_i^{(m)}$

$$\hat{\mathbf{W}}^T = \left(\Phi^T\Phi\right)^{-1}\Phi^T\mathbf{Y}$$

Selbstorganisierende Merkmalskarten

- **Motivation (Gehirn)**
- **Netzwerk-Architektur**
- **Topographische Merkmalskarten (Früher: „Kohonen-Karten“)**
- **Selbstorganisierende Merkmalskarte (Kohonen-Lernregel)**
- **Anwendungsbeispiele**

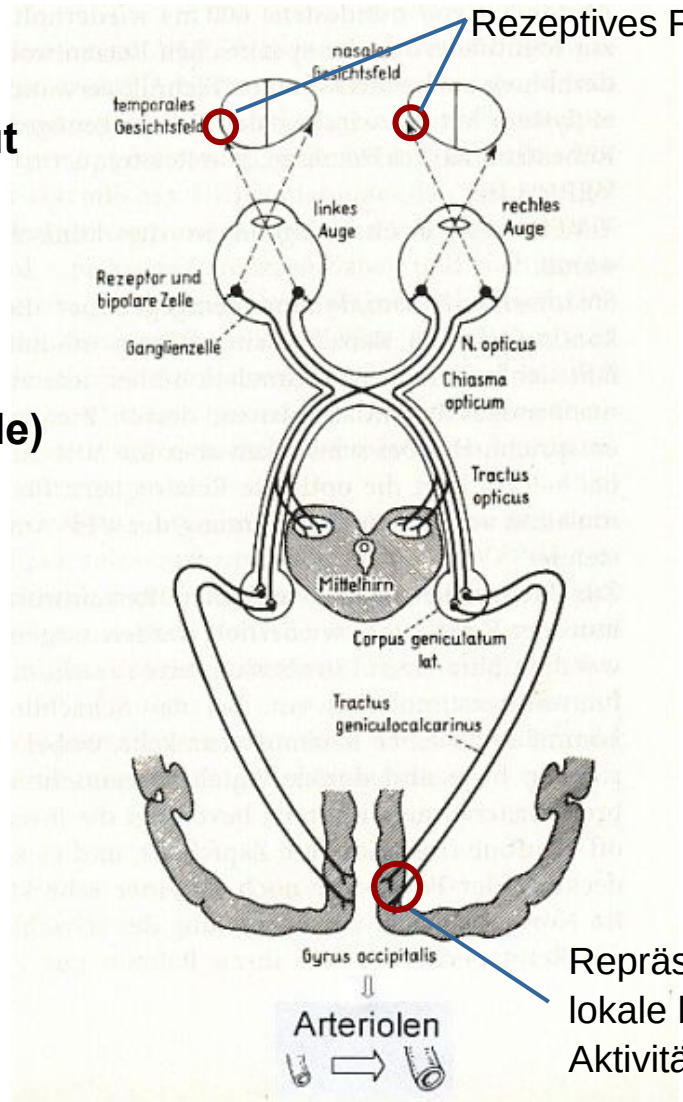
Motivation: Topografische Merkmalskarten im Gehirn

Frühe Sehbahn:

- **Zellen der primären Sehrinde verarbeiten Input aus lokalem Bereich: „Rezeptives Feld“**
- **Benachbarte rezeptive Felder erregen benachbarte Kortexbereiche**
- **Benachbarte Stimuluseigenschaften (Merkmale) erregen benachbarte Kortexbereiche**

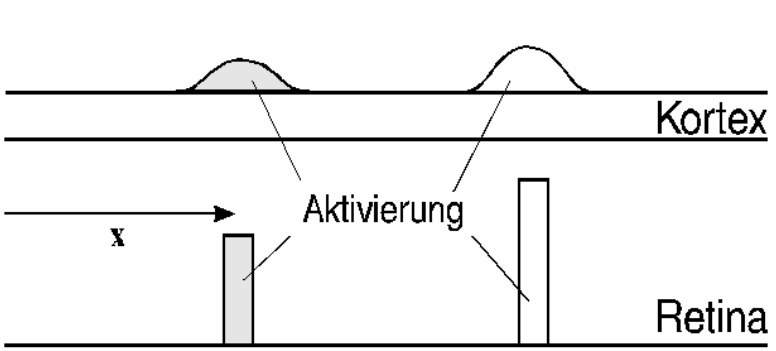
Also: Struktur interner Repräsentationen:

- **Merkmale der Umwelt werden durch den Ort der stärksten Aktivierung in der Großhirnrinde kodiert („Merkmalskarte“).**
- **Diese Kodierung ist stetig, d.h. benachbarte kortikale Orte kodieren ähnliche Reizmerkmale („Topographisch“).**



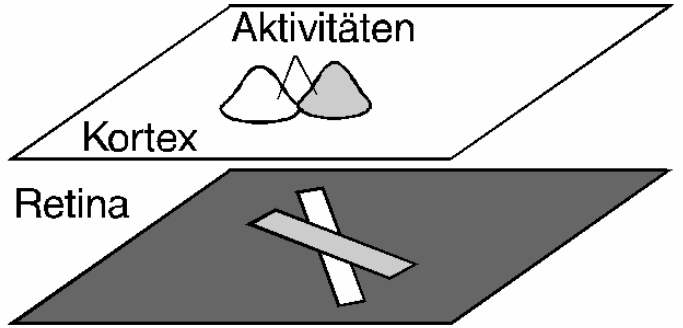
Beispiel „Retinotopie“:

- **Erscheinungsort eines Merkmals wird durch den Ort kortikaler Erregung kodiert**



Beispiel „Orientierungspräferenz-Karte“:

- **Reizorientierung wird durch den Ort kortikaler Erregung kodiert**



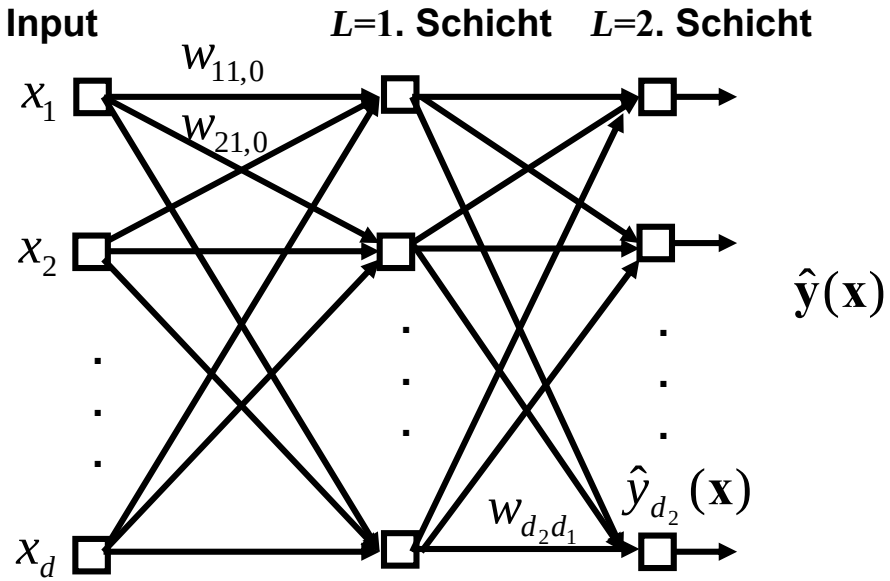
Kohonens Idee:

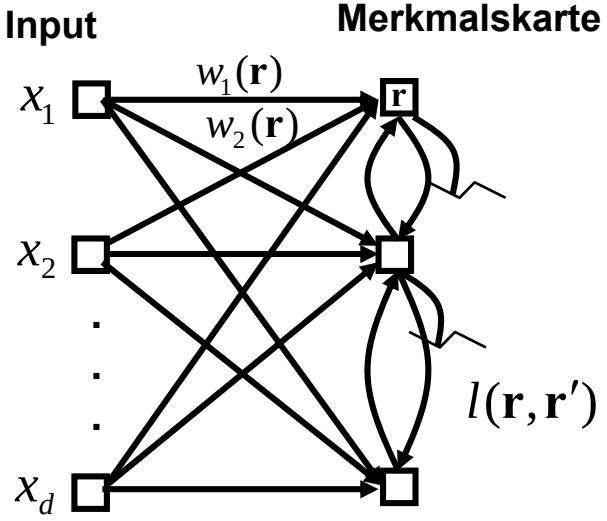
- **Benutze topographische Merkmalskarte als Prinzip zur Datenrepräsentation**
- **Jedes Neuron repräsentiert einen Teil des Datenraums**
- **Wo viele Daten sind, sind viele Neuronen zuständig => Dichteschätzung**
- **Gute Datenrepräsentation wird gelernt**

Implementierung in einem Neuronalen Netz

- **MLP:**
  - **Feed-Forward**
  - **Input-Neuronen verantwortlich für Halbraum**
  - **Verarbeiteter Input wird zu Output transformiert**
  - **Überwachtes Lernverfahren**

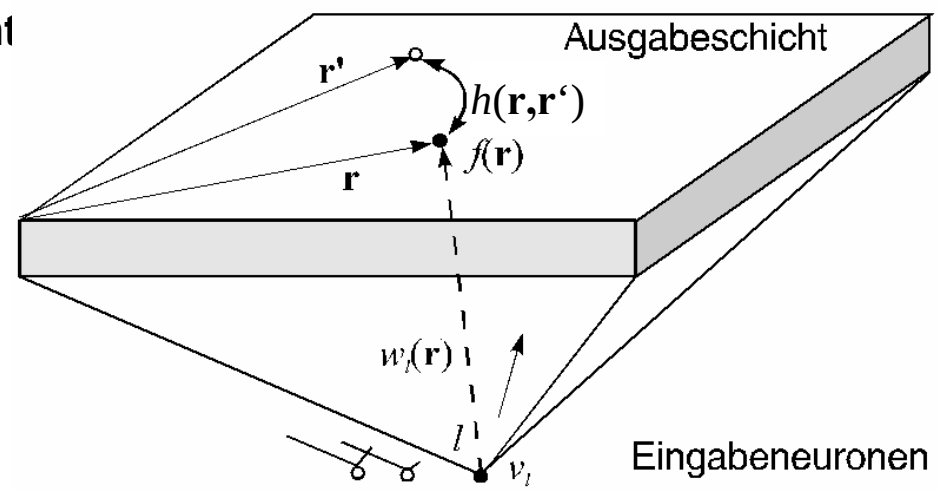
- **Selbstorganisierende Merkmalskarte:**
  - **Rückgekoppelte Verbindungen**
  - **Input-Neuronen verantwortlich für lokalisierten Bereich (wie RBF)**
  - **Verarbeiteter Input wird zu sich selbst in Verbindung gesetzt**
  - **Unüberwachtes Lernverfahren**



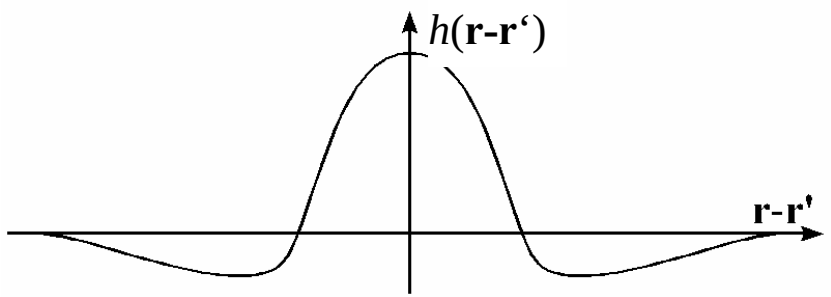


Kohonen-Netzwerk: Verschaltungsstruktur

- **Rekurrent vernetzte Ausgabeschicht**



- **Mexican-Hat-förmige, zeitunabhängige laterale Wechselwirkung**



Im Gegensatz zum Hopfieldnetz sind die lateralen Verbindungen *h*(r,r') im Ortsraum festgelegt. Biologisch motiviert: Mexican-Hat-Struktur („Umfeldhemmung").

Netzwerk-Dynamik:

- **Die Inputneuronen initialisieren ein Aktivitätsmuster in der Ausgabeschicht**
- **Dieses wird durch die rückgekoppelte Netzwerkdynamik verändert, läuft in einen Attraktor**
- **Dynamische Gleichung:** $$\frac{d}{dt} f(\mathbf{r},t) = -f(\mathbf{r},t) + g\left(\sum_{l=1}^{d} w_l(\mathbf{r}) x_l + \int d\mathbf{r}' h(\mathbf{r}-\mathbf{r}') f(\mathbf{r}',t) - \theta \right)$$
- **Analytische Fixpunkt-Lösung schwierig, aber:**

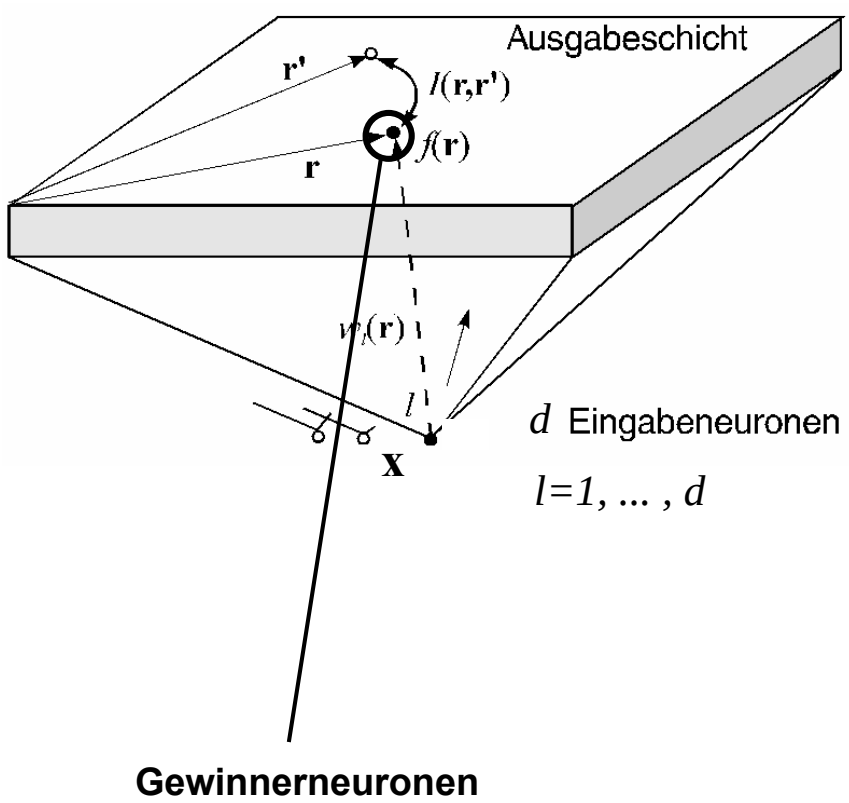
Beobachtung:

- **Laterale Wechselwirkungen vom Mexican-Hat-Typ mit genügend starker Inhibition führen immer zur Ausbildung lokalisierter Aktivitäts-"Blobs"**
- **Denn: Im Laufe der Iterationen inhibiert das anfangs am stärksten aktivierte Neuron seine Nachbarn am stärksten, vermindert damit deren inhibitorische Wirkung, kann so immer stärker aktiv werden, u.s.w...**
- **Also: Das anfangs am stärksten aktivierte Neuron + Nachbarn gewinnen:**
- **Winner-take-all Aktivierung**

Das Kohonen-Modell: „Self-Organizing Feature-Map“ (SOM)

Architektur einer Selbstorganisierenden Merkmalskarte

- ***d* Eingabeneuronen senden Inputs zu allen Neuronen im zweidimensionalen Gitter der Ausgabeschicht.**
- **Die Ausgabeneuronen stehen durch eine Nachbarschaftsfunktion miteinander in Beziehung**



Merkmalskarte durch „Winner-Take All“

- **Das Neuron mit dem stärksten Input sowie seine Nachbarn erhalten den „Zuschlag“, dürfen also den Input repräsentieren**

Selbstorganisation:

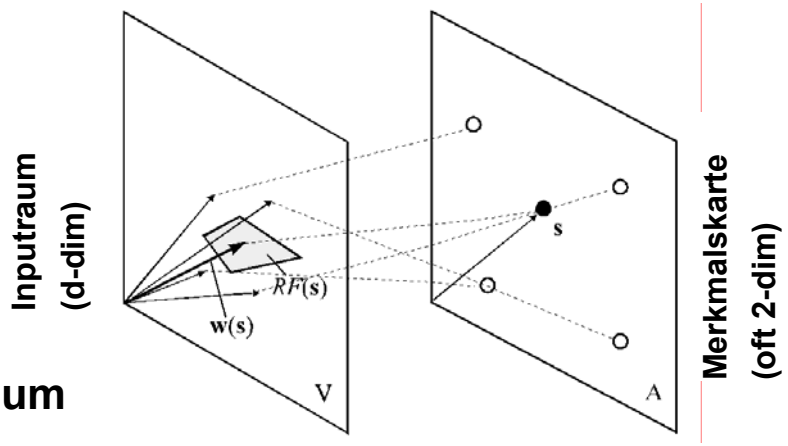
- **Gewichtsvektoren der aktivierten Nachbarn rücken näher zueinander**

Def: Merkmalskarte:

- **Abbildung, die jedem Vektor des Inputraumes (Musterraum, Merkmalsraum) einen Ort $\mathbf{s}$ in einer repräsentierenden Schicht (Karte) zuweist**

$$\phi_{\mathbf{w}} : \mathbf{x} \to \phi_{\mathbf{w}}(\mathbf{x}) = s \left\| \mathbf{w}(\mathbf{s}) - \mathbf{x} \right\| = \min_{\mathbf{r}} \left\| \mathbf{w}(\mathbf{r}) - \mathbf{x} \right\|$$

- **Bem: Die Struktur der Karte hängt von den Gewichtsvektoren ab**



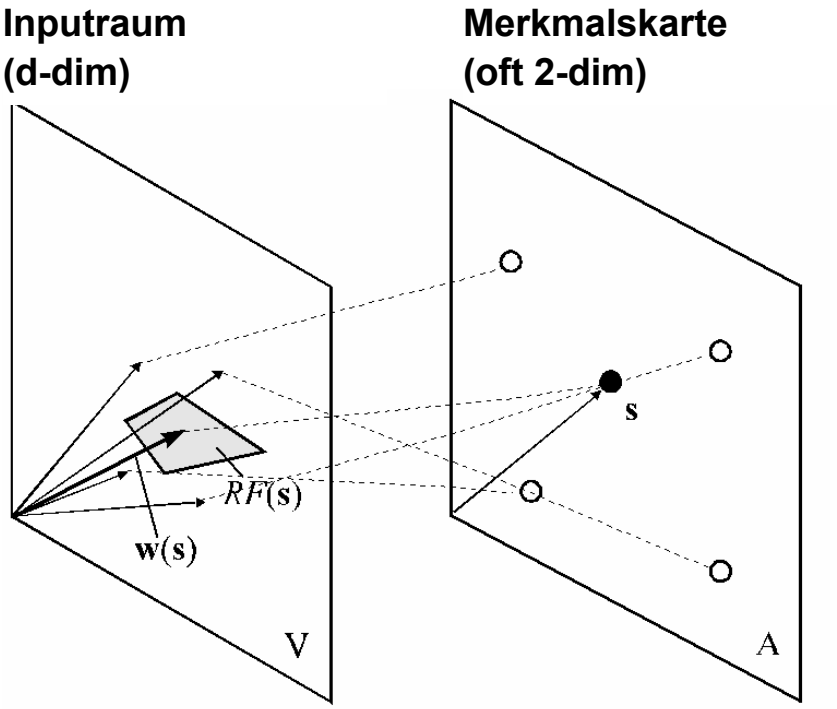
Berechnung der Gewinner:

- **„Merkmal“ = Vektor $\mathbf{x}$ im d-dimensionalen Inputraum**
- **Gewichtsvektor $\mathbf{w}$ jedes Neurons lebt gleichermaßen im d-dimensionalen Inputraum**
- **Gewinner-Neuron: Neuron $\mathbf{s}$, dessen normalisierter Gewichtsvektor am nähesten am normalisierten Datenpunkt ist**

$$\mathbf{s} = \arg\min_{\mathbf{r}} \left(\left\| \mathbf{w}(\mathbf{r}) - \mathbf{x} \right\| \right)$$

- **Das Gewinner-Neuron und seine Nachbarn werden gemäß der Funktion $l(\mathbf{s},\mathbf{r})$ aktiviert: Es entsteht ein lokaler Aktivitäts-Blob um $\mathbf{s}$**
- **Bsp:** $$l(\mathbf{s},\mathbf{r}) = \exp\left(\frac{-(\mathbf{s}-\mathbf{r})^2}{2\sigma^2} \right)$$

Rezeptive Felder und Merkmalskarten :



- **Beobachtung: Jeder Gewichtsvektor / jedes Neuron ist Gewinner in einem ganzen Abschnitt des Inputraumes: „Rezeptives Feld“**

$$RF(\mathbf{s}) = \left\{ \mathbf{x} \in V \middle| \left\| \mathbf{w}(\mathbf{s}) - \mathbf{x} \right\| = \min_{\mathbf{r}} \left(\left\| \mathbf{w}(\mathbf{r}) - \mathbf{x} \right\| \right) \right\}$$

- **<u>Topographische</u> Merkmalskarte: Benachbarte rezeptive Felder sollten zu benachbarten Neuronen in der Karte gehören**
- **Ziel: Lernregel, die eigenständig diese topographische Ordnung herstellt: „Selbstorganisierende Merkmalskarte“**

<u>Vorteile:</u>

- **Nachbarschaftsbeziehungen im „unübersichtlichen“ Inputraum können direkt in der Ausgabeschicht abgelesen werden**
- **Auch andere Eigenschaften repräsentierbar; z.B.: Punktdichten => Dichteschätzung**

Die Kohonen-Lernregel :

- **Idee: Für jeden Datenvektor: Nähere die Gewichtsvektoren des Gewinners und seiner Nachbarn in der Karte dem Inputmuster an.**
- **Dadurch erhalten Nachbarneuronen schließlich benachbarte rezeptive Felder**
- **Dadurch werden Regionen mit vielen Datenpunkten durch viele Vektoren repräsentiert (Dichteschätzung)**

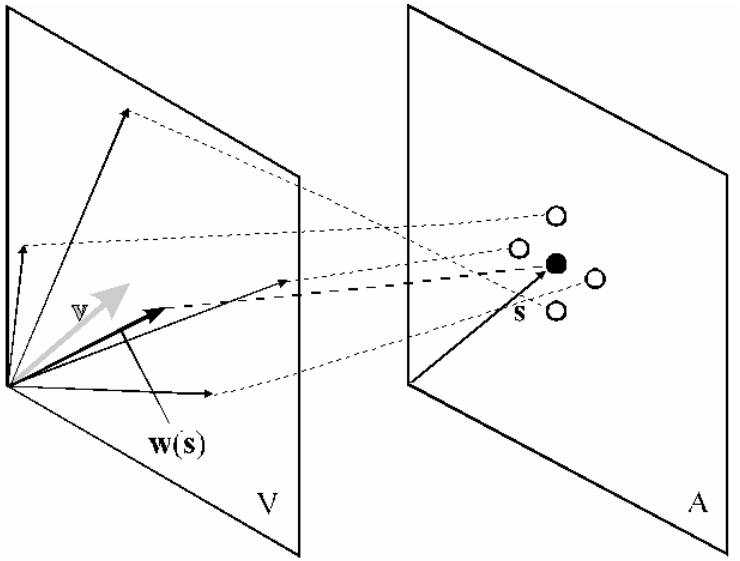
Algorithmus:

- **Geg: Datensatz auf Länge 1 normierter Datenvektoren** $D = (\mathbf{x}^{(1)},...,\mathbf{x}^{(m)},...,\mathbf{x}^{(M)})$
- **Belege Gewichtsvektoren mit Zufallswerten**
- **Präsentiere Datenvektor** $\mathbf{x}^{(m)}$
- **Ermittle den Gewinner** $\mathbf{s}^{(m)} = \mathbf{s}\left\|\mathbf{w}(\mathbf{s}) - \mathbf{x}^{(m)}\right\| = \min_{\mathbf{r}}\left\|\mathbf{w}(\mathbf{r}) - \mathbf{x}^{(m)}\right\|$
- **Nähere Gewichtsvektoren proportional zur Nachbarschaftsfunktion zueinander an**

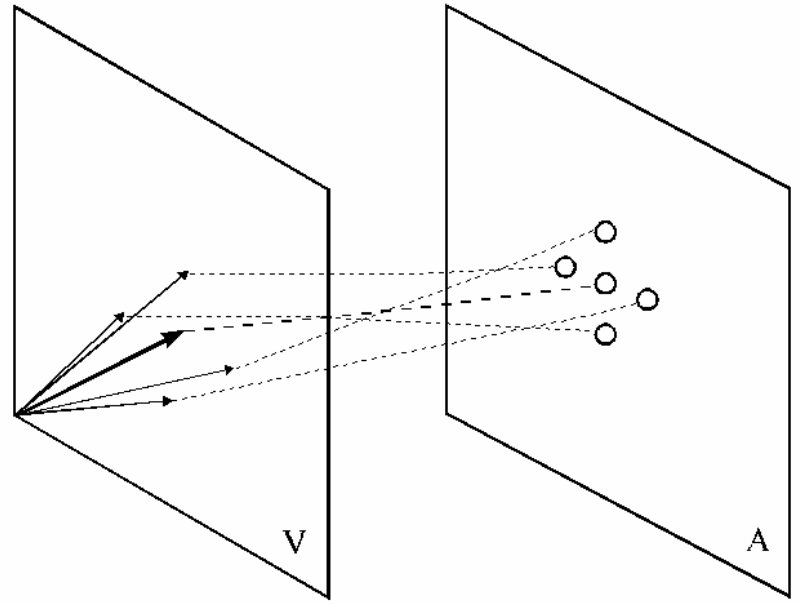
$$\mathbf{w}^{\text{neu}}(\mathbf{r}) = \mathbf{w}^{\text{alt}}(\mathbf{r}) + \Delta\mathbf{w}(\mathbf{r}) \quad \text{mit} \quad \Delta\mathbf{w}(\mathbf{r}) = \eta l(\mathbf{s},\mathbf{r})\left(\mathbf{x}^{(m)} - \mathbf{w}(\mathbf{r})\right)$$

- **Normiere Gewichtsvektoren auf Länge 1, präsentiere nächsten Datenpunkt**

Effekt der Lernregel:



Vor dem Training



Nach dem Training

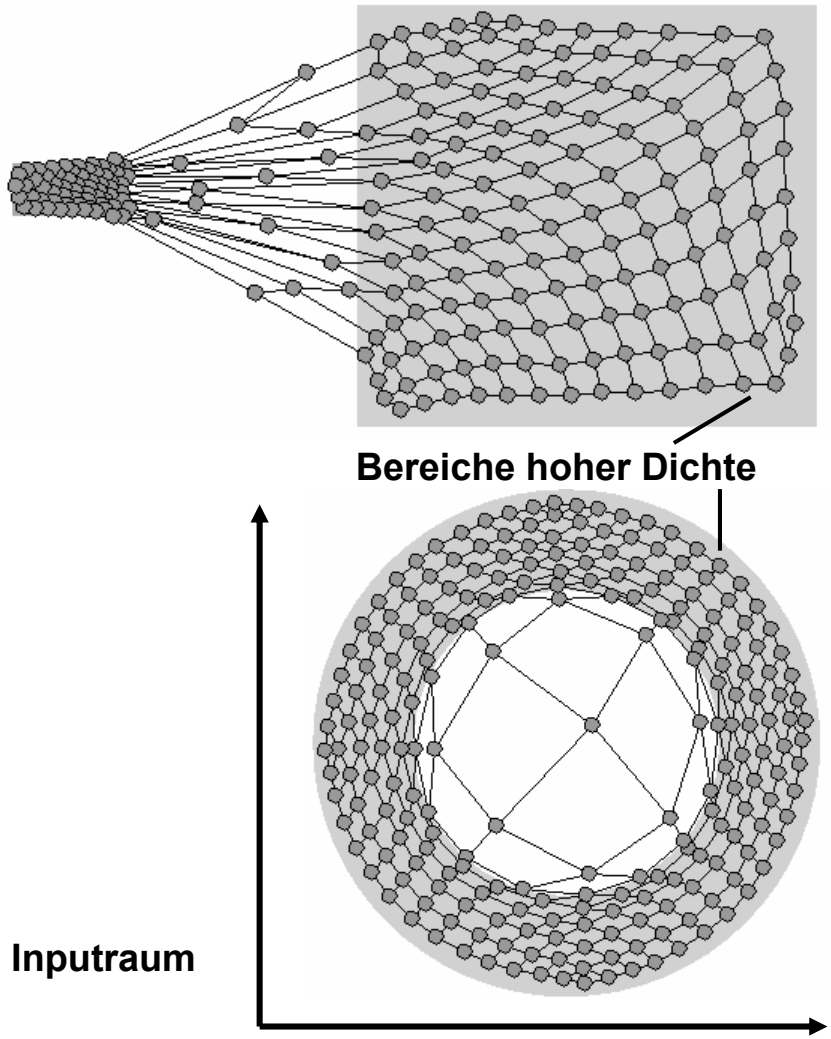
Bemerkungen:

- **Üblicherweise werden die Breite σ der Nachbarschaftsfunktion und die Lernschrittweite η im Lauf der Zeit verringert:** $\sigma = \sigma(t)$, $\eta = \eta(t)$.
- **Konvergenzbeweise gegen einen statistisch beschreibbaren Gleichgewichtszustand existieren für:** $\eta(t) = \eta t^{-\alpha}, 0 < \alpha \leq 1$ $\lim_{t \to \infty} \sigma(t) = 0$

Anwendungsbeispiele

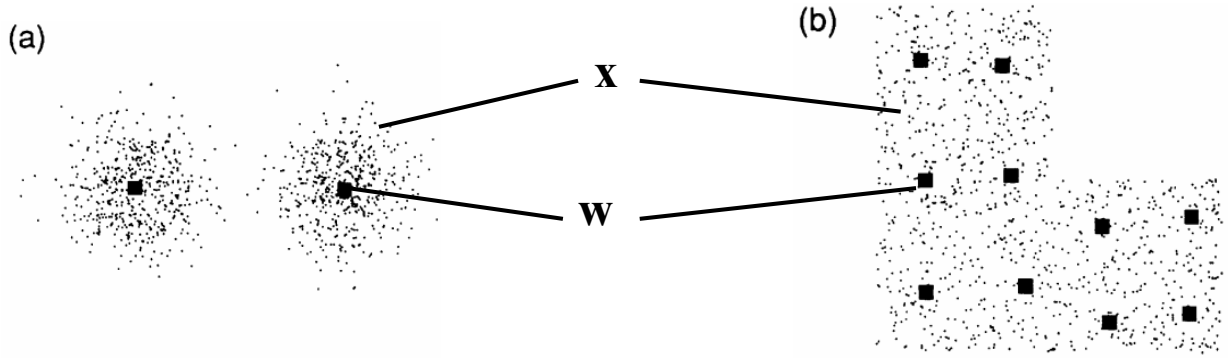
Dichteschätzung:

- **Schätzen einer Wahrscheinlichkeitsdichte, die den Daten zugrunde liegt**
- **Die Merkmalskarte wird mit Vektoren trainiert, die aus der zu schätzenden pdf als Stichproben gezogen wurden**
- **Häufig auftretende Merkmale werden von der SOM durch mehr Neuronen repräsentiert**
- **Beispiel: Merkmalskarte mit 15x15 Neuronen**



Spezialfall Vektorquantisierung: **(Verfahren zur Datenkompression)**

- **Folge von Datenvektoren** $\mathbf{x}(t)$, $t = 1, \ldots,$ **sollen durch eine feste Anzahl von Referenzvektoren** $\mathbf{w}_s$ **approximiert werden**
  - **Kompression: Speicherung des Index** $s(\mathbf{x})$ **mit minimalem** $\|\mathbf{w}_{s(\mathbf{x})} - \mathbf{x}\|$ **für jedes** $\mathbf{x}(t)$
  - **Restauration:** $\mathbf{x}(t) := \mathbf{w}_{s(\mathbf{x}(t))}$, $t = 1, \ldots$ **(Es gibt einen Restaurationsfehler!)**
- **Ziel: Finde optimale Verteilung der Referenzvektoren mit min. Restaurationsfehler**
- **Kostenfunktion**
$$E = \int P(\mathbf{x}) \left\| w_{s(\mathbf{x})} - \mathbf{x} \right\|^2 d\mathbf{x} \equiv \min$$
- **Gradientenabstieg**
$$\mathbf{w}_s(t+1) = \mathbf{w}_s(t) - \frac{\eta}{2}\frac{\partial E}{\partial \mathbf{w}_s} = \mathbf{w}_s(t) + \eta \int_{RF(s)} (\mathbf{x} - \mathbf{w}_s(t)) P(\mathbf{x}) d\mathbf{x}$$
- **Empirische Lernregel**
$$\mathbf{w}_{s(\mathbf{x})}(t+1) = \mathbf{w}_{s(\mathbf{x})}(t) + \eta(\mathbf{x} - \mathbf{w}_{s(\mathbf{x})})$$
- **Bsp:**

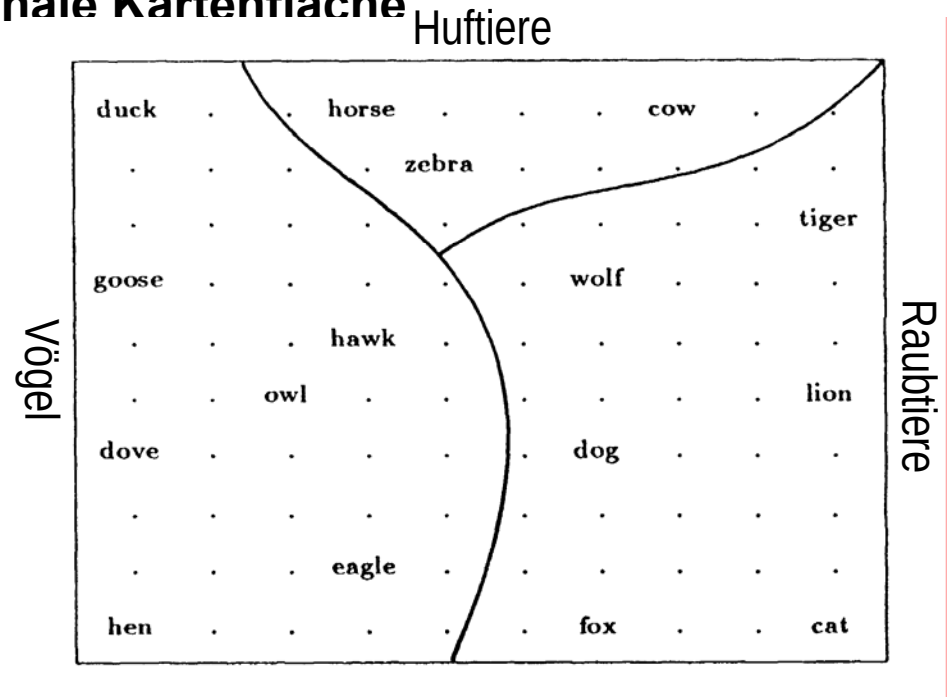


Clustering und Visualisierung:

- **Problem: Hochdimensionaler Datenraum**
- **Aufgabe: Finde Korrelationen in den Daten**
- **Lösung mit dem Kohonenalgorithmus: Merkmale, die im Inputraum nahe beieinanderliegen, werden auf benachbarte Gebiete der SOM abgebildet**

	klein	mittel	groß	2 Beine	4 Beine	Haare	Hufe	Mähne	Federn	jagt	rennt	fliegt	schwimmt
Taube	1	0	0	1	0	0	0	0	1	0	0	1	0
Henne	1	0	0	1	0	0	0	0	1	0	0	0	0
Ente	1	0	0	1	0	0	0	0	1	0	0	0	1
Gans	1	0	0	1	0	0	0	0	1	0	0	1	1
Eule	1	0	0	1	0	0	0	0	1	1	0	1	0
Falke	1	0	0	1	0	0	0	0	1	1	0	1	0
Adler	0	1	0	1	0	0	0	0	1	1	0	1	0
Fuchs	0	1	0	0	1	1	0	0	0	1	0	0	0
Hund	0	1	0	0	1	1	0	0	0	0	1	0	0
Wolf	0	1	0	0	1	1	0	1	0	1	1	0	0
Katze	1	0	1	0	1	1	0	0	0	1	0	0	0
Tiger	0	0	1	0	1	1	0	0	0	1	1	0	0
Löwe	0	0	1	0	1	1	0	1	0	1	1	0	0
Pferd	0	0	1	0	1	1	1	1	0	0	1	0	0
Zebra	0	0	1	0	1	1	1	1	0	0	1	0	0
Kuh	0	0	1	0	1	1	1	0	0	0	0	0	0

- **Die Merkmale werden ihrer Ähnlichkeit entsprechend auf der SOM angeordnet**
- **Topologieerhaltende Abbildung des hochdimensionalen Inputraumes auf die zweidimensionale Kartenfläche**



Spezialfall: Dimensionsreduktion

- **Geg: Signale mit vielen Freiheitsgraden (hohe Dimensionalität).**
- **Ges: Bestmögliche Repräsentation in einer niedrigdimensionalen Neuronenstruktur (typisch: 1 bis 2-dimensional).**

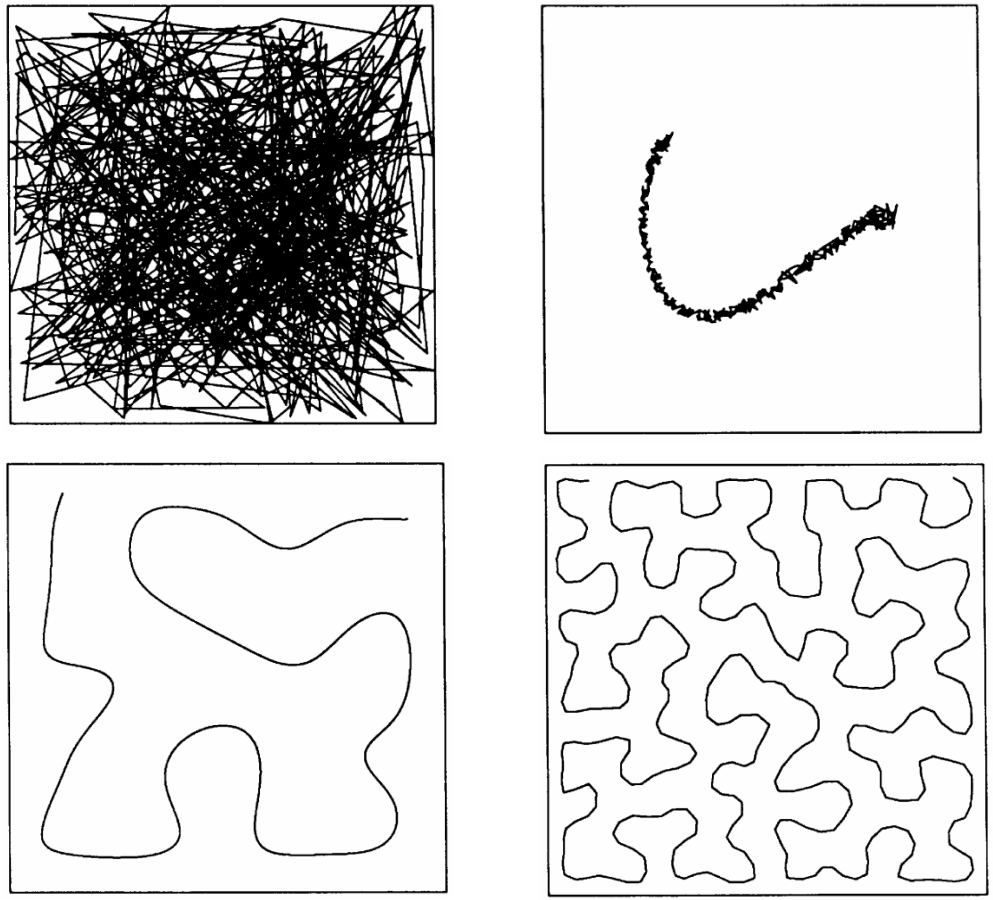
- **Der Kohonenalgorithmus führt zu einer optimalen Abdeckung des höherdimensionalen Raumes gemäß der Wahrscheinlichkeitsverteilung der präsentierten Inputmuster**

- **Beispiel: Eindimensionale Neuronenkette wird mit zweidimensionalen Vektoren aus dem Einheitsquadrat trainiert**

- **Zeitabhängige Breite der Nachbarschaftsfunktion:** $\sigma(t) = 100(0.01)^{10^{-5}t}$

- **bewirkt sukzessive Ausbildung immer feinerer Strukturen**

Dimensionsreduktion: Lernverlauf



Zuordnung: zu Beginn, nach 200, nach 50000, nach 100000 Schritten

Beispiel Sensorik: Positionskodierung einer Schallquelle

Ausgangspunkt:

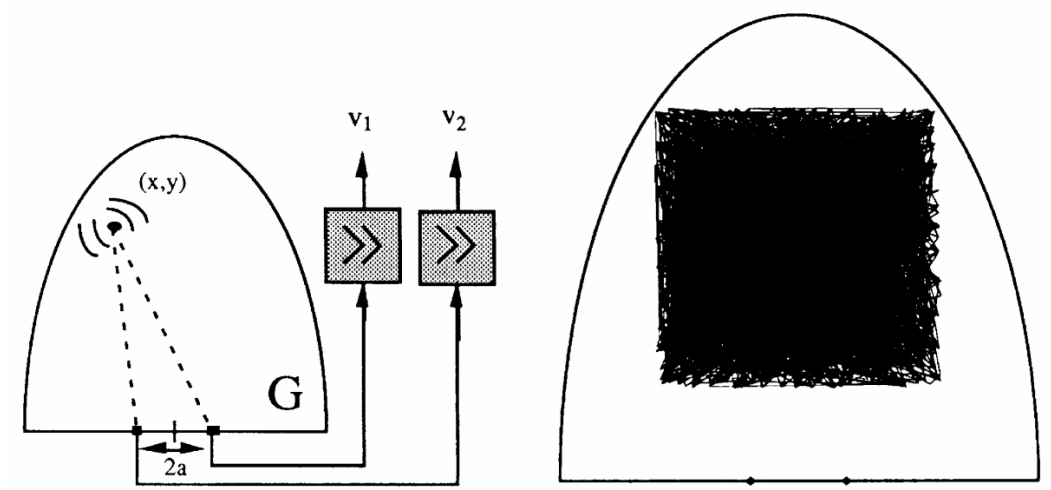
- **Schallsignale einheitlicher Lautstärke ertönen in beliebigen Positionen innerhalb eines krummlinig begrenzten Gebietes**
- **Die Signale werden von zwei Mikrofonen aufgenommen, deren Ausgangsintensität den Abstand der Schallquelle kodiert**
- **Mit der nichtlinearen Verstärkerkennlinie $f(x)$ werden die Signale zu:**

$$v_1 = f\left(\left[(x-a)^2 + y^2\right]^{-1}\right)$$

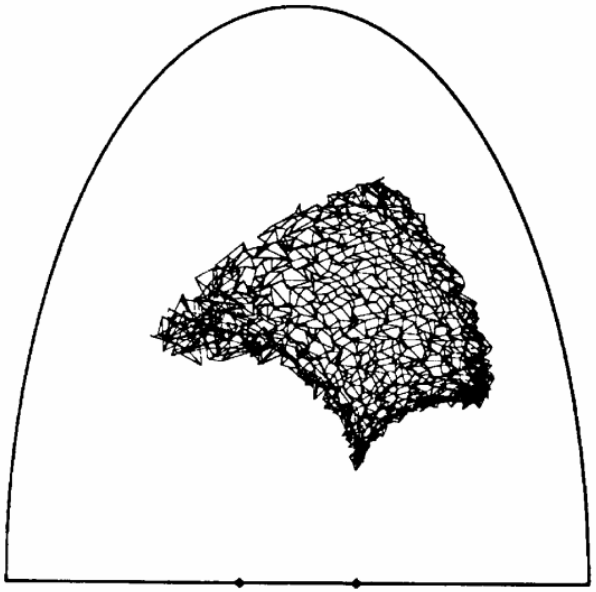
$$v_2 = f\left(\left[(x+a)^2 + y^2\right]^{-1}\right)$$

Ziel:

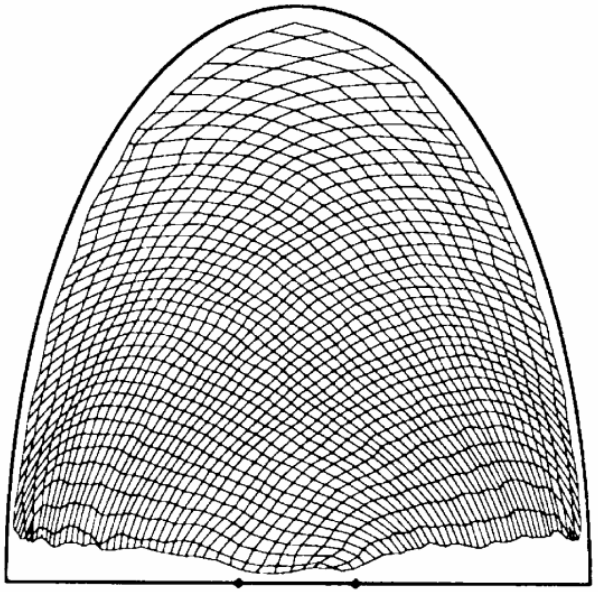
- **Lerne Merkmalskarte zur Rekonstruktion des Ortes aus dem gemessenen Schallsignal**



Positionskodierung einer Schallquelle: Lernverlauf



Karte nach 100 Lernschritten

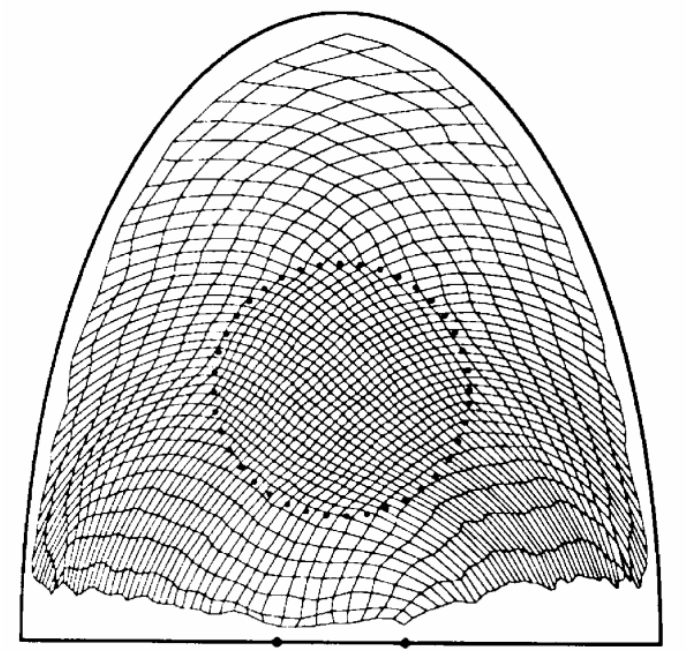


Karte nach 40000 Lernschritten

- **Es sind die Gewichtsvektoren und damit die Positionen höchster Sensitivität für ein Gitter von 40x40 Neuronen dargestellt. Nach dem Training kodiert jedes Neuron einen Teil-bereich des betrachteten Gebietes. Das Netzwerk hat die der Positionskodierung innewohnende nichtlineare Transformation invertiert.**

Positionskodierung einer Schallquelle: Dichteschätzung des sensorischen Inputs

- **Die Repräsentation passt sich der Wahrscheinlichkeitsverteilung der Inputsignale an, d.h. häufig präsentierte Muster werden durch mehr Kohonen-Neuronen kodiert**



Im zentralen Kreis wurde die Signalhäufigkeit gegenüber außen um einen Faktor 3 erhöht.

Jedes Muster hat eine „anziehende“ Wirkung auf die im Inputraum benachbarten Gewichtsvektoren: Kumulation bei Peaks der Wahrscheinlichkeitsverteilung.

(s. a. Vektorquantisierung).

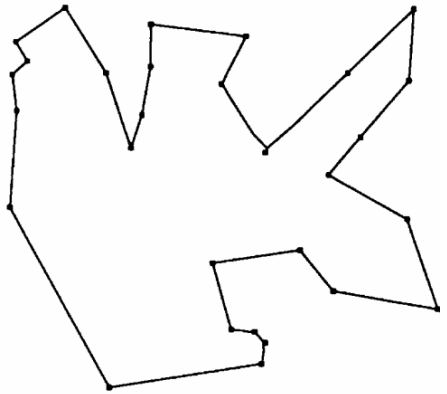
Denselben Effekt kann man durch lokale Erhöhung der Netzwerkelastizität (Verbreiterung von l) erreichen

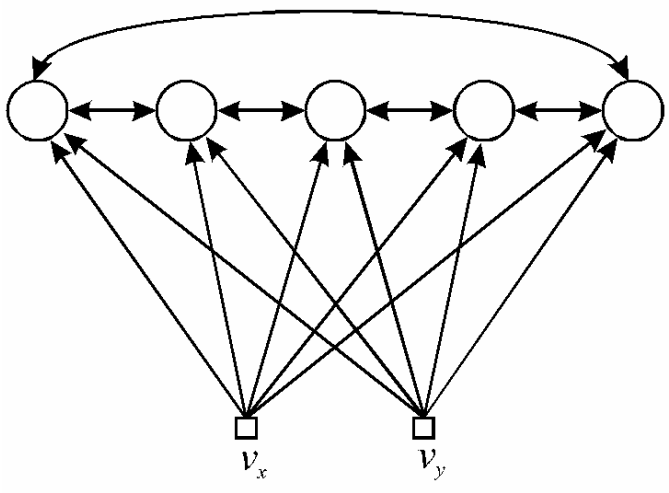
Optimierungsprobleme:

- **Lösungsaufwand für ein System mit *L* Komponenten steigt wie exp(*L*) bzw. *L!***

- **Beispiel: Handlungsreisenden-Problem:**
  - **Finde die kürzeste Route, die *L* gegebene Städte berührt.**
  - **Ansatz: Wähle eindimensionalen Neuronenring mit $N \geq L$ Neuronen ($N > L$ empfohlen) und zweidimensionalen Gewichtsvektoren.**

    Die Inputsignale kodieren *x*- und *y*-Positionen der Städte.





Netzwerkarchitektur für das Handlungsreisenden-Problem. Die Inputs kodieren die xy-Position, die Ausgabeneuronen implementieren eine Ringnachbarschaft.

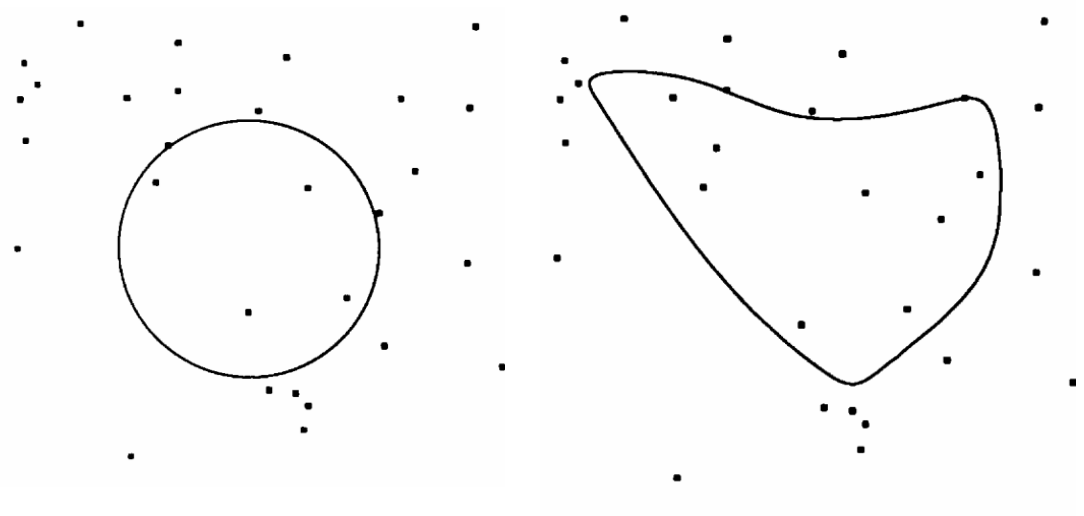
- **Präsentiere die Städtepositionen als Inputmuster und trainiere mit**

$$\Delta \mathbf{w}(r,t) = \eta(t)\, l(s(\mathbf{x}), r, t)(\mathbf{x} - \mathbf{w}(r,t))$$

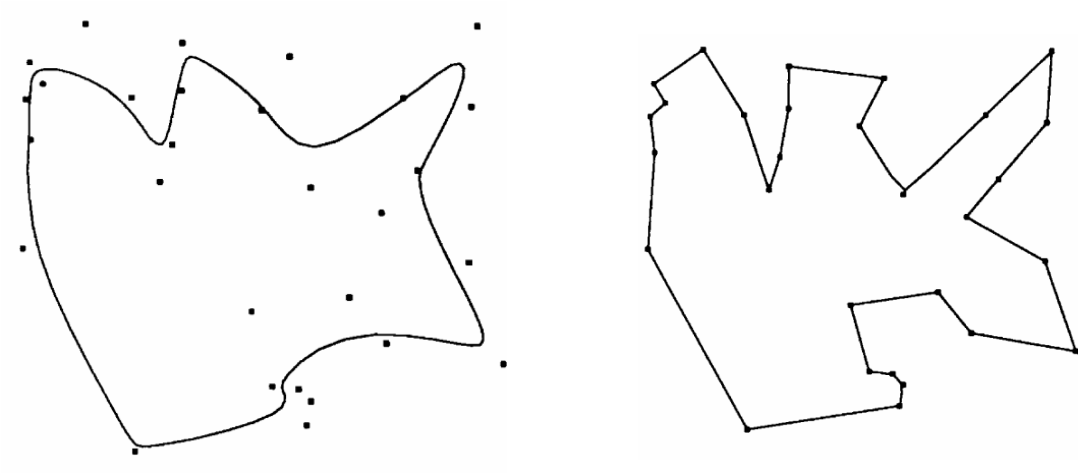
- **Die Nachbarschaftsfunktion** $l(r,r')$ **versucht, die Repräsentation des Rings im Ortsraum möglichst kurz zu halten („kurzer Weg"-Forderung)**
- **Unter dieser Bedingung werden die Städtepositionen sukzessive approximiert**

Simulationsablauf (nach Durbin und Willshaw 1987):

$L = 30$
$N = 800$
$h = 0.8$
$s(t) = 50\ 0.02^{(t/t\max)}$
$t_{max} = 10000$



Verlauf einer Simulation mit 30 Städten es werden die Gewichtsvektoren im zweidimensionalen Inputraum gezeigt nach 0, 5000, 7000 und 10000 Lernschritten



Lernen von Datenmodellen

Approximation

- **Bayesianische Philosophie und Bayes‘sches Schließen**
- **Bayesianische Datenmodellierung**
- **Spezialfall: Maximum-Likelihood-Schätzung**

Ziel maschinellen Lernens (=statistische Inferenz)

Gegeben:

- **Man verfügt nur über einen Datensatz** $D = \left\{ \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(M)} \right\}$ **als Beispiel**
- **Statistische Datenmodelle verfügen über freie Parameter** $\mathbf{w}$

Ziel:

- **Gute modellhafte Beschreibung der zugrundeliegenden statistischen Struktur.... Also**
- **Gute Beschreibung des Datensatzes** D **(=> Optimierung der Parameter** $\mathbf{w}$**) ... Aber auch**
- **Gute Generalisierung, d.h. Beschreibung neuer Datensätze (=> Regularisierung)**

Bayesianische Philosophie und Bayes‘sches Schließen

Wiederholung:

Frequentistische Philosophie:

- **Der Datenpunkt ist eine Stichprobe aus einer existierenden „wahren“ Verteilung** $p(x)$
- **=>** $p(x)$ **kann als Grenzwert der relativen Häufigkeit interpretiert werden**
- **„Wahrscheinlichkeit als Prozentsatz“**
- **Typisches Beispiel: Wahrscheinlichkeit, 6 zu würfeln**

Bayesianische Philosophie:

- **Nur der Datenpunkt** x_0 **existiert, es gibt keine zugrundeliegende Verteilung**
- **„Wahrscheinlichkeit als Glaube (Belief) des Eintretens eines Sachverhalts“**
- **Basiert allein auf Vorwissen und auf den beobachteten Daten**
- **Typisches Beispiel: Wahrscheinlichkeit, den nächsten Marathon zu gewinnen**

Motivation:

- **Es gilt, die Daten zu erklären.**
- **Es gibt kein einzelnes zugrundeliegendes Modell**
- **Bestimme Wahrscheinlichkeitsverteilung über die Modelle $\mathbf{w}$, gegeben**
  -- den Datensatz D
  -- unser Vertrauen (Vorwissen, „Prior Belief“) in die Modelle

Prinzip: Bayes‘sches Gesetz

$$p(\mathbf{w} \mid D) = \frac{1}{p(D)} p(D \mid \mathbf{w}) p(\mathbf{w})$$

$p(\mathbf{w})$ **= a Priori-Wissen bzw. Vertrauen (Belief) in ein Modell ohne Daten („prior“)**

$p(D \mid \mathbf{w})$ **= Wahrscheinlichkeit der Daten im Lichte des Modells $\mathbf{w}$ („likelihood“)**

$p(\mathbf{w} \mid D)$ **= a Posteriori Wahrsch. (Belief) in ein Modell geg. Daten („posterior“)**

$p(D)$ **= Evidenz („evidence“), bewertet Güte der Modellfamilie (vgl. „Hyperparameter“)**

Beispiel zum Bayes'schen Schließen:

Ein Labortest gibt mit Unsicherheit behaftete Auskunft über einen Krebstyp

-- Man weiß, dass 0.8 % der Bevölkerung diesen Krebs entwickeln.

-- Der Test reagiert in 98% der Fälle positiv (+), wenn der Patient Krebs hat.

-- Der Test reagiert in 97% der Fälle negativ (-), wenn der Proband gesund ist.

Frage: Wie sicher hat der Proband Krebs, wenn das Testergebnis bekannt ist?

Vor dem Test:

$$p(krebs) = 0.008$$
$$p(gesund) = 0.992$$

$$p(+ \mid krebs) = 0.98$$
$$p(- \mid krebs) = 0.02$$
$$p(+ \mid gesund) = 0.03$$
$$p(- \mid gesund) = 0.97$$

$$p(+) = p(+ \mid krebs)\, p(krebs) + p(+ \mid gesund)\, p(gesund)$$
$$= 0.98 \cdot 0.008 + 0.03 \cdot 0.992$$
$$= 0.0376$$
$$p(-) = 0.9624$$

Nach dem Test:

$$p(krebs \mid +) = p(+ \mid krebs)\, p(krebs) / p(+) = 0.98 \cdot 0.008 / 0.0376 = 0.209$$
$$p(krebs \mid -) = p(- \mid krebs)\, p(krebs) / p(-) = 0.02 \cdot 0.008 / 0.9624 = 0.00016$$

-- Eintreffen eines Datenpunktes verändert die Schätzung

-- Positiver Test erhöht die Krebschance nur moderat

Bayesianische Datenmodellierung

Bayes'sche Dichteschätzung:

- **Benutze alle Modelle zur Schätzung der Wahrscheinlichkeit für neuen Datenpunkt $\mathbf{x}$:**

$$p(\mathbf{x} \mid D) = \int p(\mathbf{x} \mid \mathbf{w})\, p(\mathbf{w} \mid D) d\mathbf{w}$$

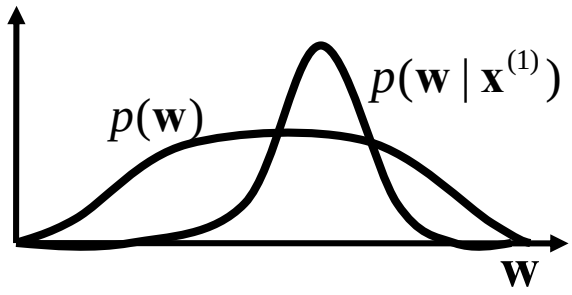
Bayes'sche Regression:

- **Marginalisiere über Modelle und bilde Erwartungswert über Outputs:**

$$\hat{y}(x \mid D) = \int y \int p_n(y - f(x \mid \mathbf{w}))\, p(\mathbf{w} \mid D) d\mathbf{w} dy$$

Iteratives Lernen von Bayes-Schätzern:

$$D = \left\{ \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)} \right\}$$



- **Vor dem ersten Datenpunkt:** $p(\mathbf{w})$
- **Nach dem ersten Datenpunkt:** $p(\mathbf{w} \mid \mathbf{x}^{(1)}) \propto p(\mathbf{x}^{(1)} \mid \mathbf{w})\, p(\mathbf{w})$
- **Nach dem zweiten Datenpunkt:** $p(\mathbf{w} \mid \mathbf{x}^{(1)}, \mathbf{x}^{(2)}) \propto p(\mathbf{x}^{(2)} \mid \mathbf{w})\, p(\mathbf{w} \mid \mathbf{x}^{(1)})$

. . . $p(\mathbf{w} \mid \mathbf{x}^{(1)}, \mathbf{x}^{(2)}) \propto p(\mathbf{x}^{(2)}, \mathbf{x}^{(1)} \mid \mathbf{w})\, p(\mathbf{w}) = p(\mathbf{x}^{(2)} \mid \mathbf{w})\, p(\mathbf{x}^{(1)} \mid \mathbf{w})\, p(\mathbf{w}) \propto p(\mathbf{x}^{(2)} \mid \mathbf{w})\, p(\mathbf{w} \mid \mathbf{x}^{(1)})$

Likelihood-Funktion und konjugierter Prior:

- **Beobachtung: Posterior im m-ten Schritt ist Prior im $m+1$. Schritt**
- **Wünschenswert: Funktionelle Form von Prior und Posterior sollen übereinstimmen**
- **Geg: Likelihood-Funktion $p(D \mid \mathbf{w})$**
- **Def: Prior $p(\mathbf{w})$ ist zur Likelihood-Funktion konjugiert, wenn $p(\mathbf{w})$, $p(\mathbf{w} \mid D)$ dieselbe funktionelle Form haben**
- **Bsp:**

Likelihood (Fkt. von $\mathbf{x}$, param. durch $\mathbf{w}$)		**konjugierter Prior** (Fkt. von $\mathbf{w}$)	
Gauss	$p(x \mid \mu) \propto \exp(-(x-\mu)^2 / 2\sigma^2)$	**Gauss**	$p(\mu) \propto \exp(-(\mu-\alpha_1)^2 / 2\alpha_2{}^2)$
Exp.	$p(x \mid w) = w\exp(-wx)$	**Gamma**	$p(w) \propto w^{\alpha_1} \exp(-\alpha_2 w)$
Binomial	$p(k \mid w) \propto w^k (1-w)^{M-k}$	**Beta**	$p(w) \propto w^{\alpha_1 - 1}(1-w)^{\alpha_2 - 1}$

- α_1, α_2 **Bsp. für Hyperparameter des Prior,** σ **Bsp. für Likelihood-Hyperparameter**

Hierarchische Bayes-Modelle:

- **Hierarchisches Modell: Schätzung der Hyperparameter aus der Evidenz**

$$p(\mathbf{w}) \equiv p(\mathbf{w} \mid \alpha), \quad p(D \mid \mathbf{w}) \equiv p(D \mid \mathbf{w}, \beta)$$

α = Prior-Hyperparameter, β = Likelihood-Hyperparameter. Die Evidenz wird

$$p(D) \equiv p(D \mid \alpha, \beta) = \int p(D \mid \mathbf{w}, \beta)\, p(\mathbf{w} \mid \alpha) d\mathbf{w}$$

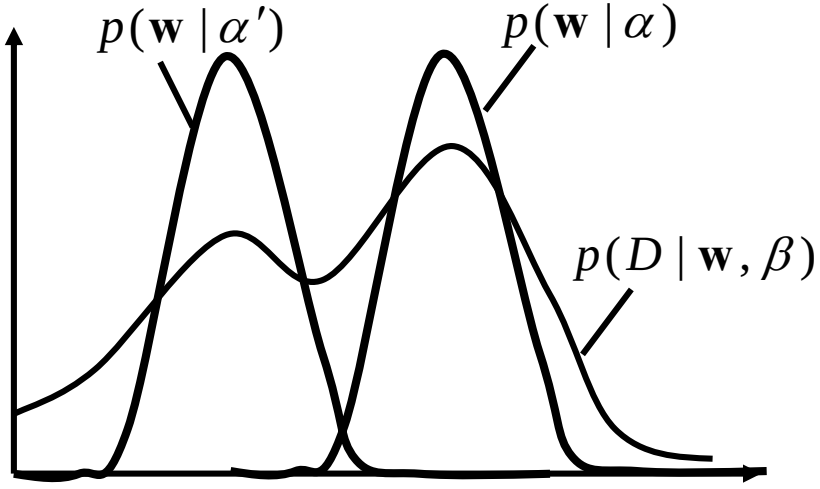
- **„Evidenz“ = Evidenz von α, β im Lichte der Daten**
  = Fähigkeit der gesamten Modellklasse, beschrieben durch $\alpha, \beta,$
  die Daten zu erklären

- **Bayes‘sche Schätzung der Hyperparameter:**

$$p(\alpha, \beta \mid D) \equiv \frac{p(D \mid \alpha, \beta)\, p(\alpha, \beta)}{p(D)}$$

$p(\alpha, \beta)$ **= Hyperprior**

- **Hierarchische Modellierung:**
  Evidenz Level k = Likelihood Level k+1



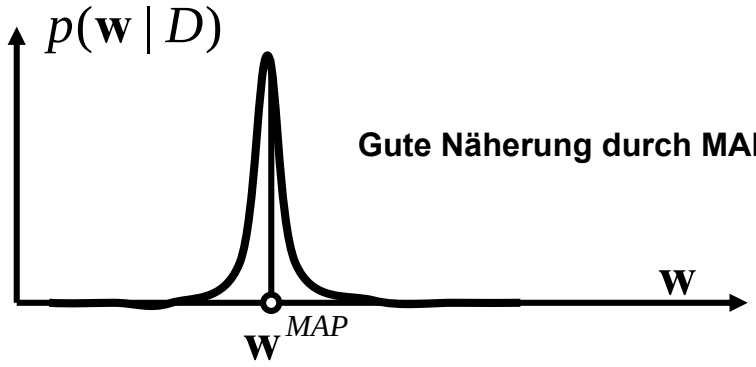
Bsp: Zwei Prior-Hyperparameter

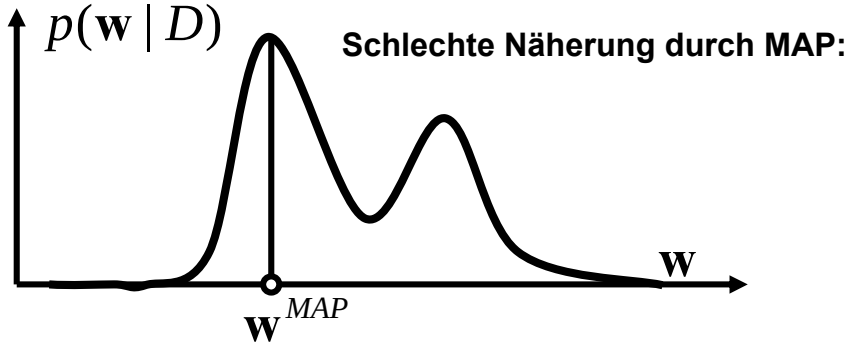
Maximum a Posteriori (MAP) Schätzer:

- **Marginalisierung über Modelle oft schwierig**
- **Näherung: Verwende das Modell mit der größten a-posteriori-Wahrscheinlichkeit**

$$\mathbf{w}^{MAP} = \arg\max_{\mathbf{w}} p(\mathbf{w} \mid D)$$

- **Gute Näherung bei konzentriertem, symmetrischem posterior (=> viele Daten)**
- **Übergang zu frequentistischer Sicht (relative Häufigkeiten, ein wahres Modell)**





MAP Lernen: Maximiere posterior $p(D \mid \mathbf{w})\, p(\mathbf{w}) = \max$

Äquivalent : Minimiere -log posterior $-\ln p(D \mid \mathbf{w}) - \ln p(\mathbf{w}) + const = \min$

⇧ ⇧

Später: Risikominimierung Regularisierung

- **Nur ein Modell, aber das muss man finden! => „Optimierung“**

Maximum-Likelihood-Parameterschätzung

Ziel:

- **Maximiere Wahrscheinlichkeit, dass die Daten aus dem Modell erzeugt wurden => Maximum Likelihood**
- **Bem: Maximum-Likelihood (ML) entspricht MAP ohne Vorwissen**

Prinzip:

- **Likelihood:** $p(D \mid \mathbf{w}) = p(\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(M)}\} \mid \mathbf{w})$
- **Max. Likelihood:** $\mathbf{w}^{ML} = \arg\max_{\mathbf{w}} p(D \mid \mathbf{w})$
- **Äquivalent:** $\mathbf{w}^{ML} = \arg\min_{\mathbf{w}} -\ln p(\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(M)}\} \mid \mathbf{w}) =: \arg\min_{\mathbf{w}} L(\mathbf{w})$

  $L(\mathbf{w})$ **wird auch als negative log-likelihood oder „Likelihood-Funktion“ bezeichnet**

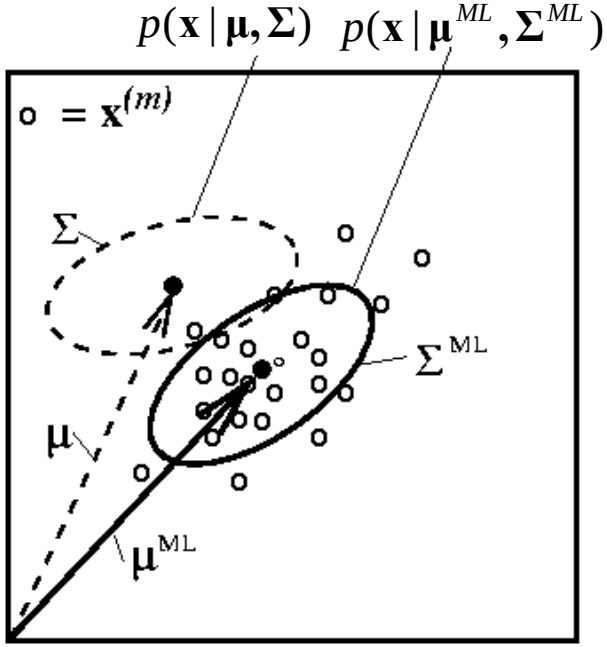
- **Für iid („independent, identically distributed“) gezogene Datenpunkte sind die Likelihood-Funktionen verschiedener Datenpunkte unabhängig:**

$$L(\mathbf{w}) = -\ln p\left(\left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}\right\} \mid \mathbf{w}\right) = -\ln \prod_{m=1}^{M} p(\mathbf{x}^{(m)} \mid \mathbf{w}) = -\sum_{m=1}^{M} \ln p(\mathbf{x}^{(m)} \mid \mathbf{w})$$

Bsp: ML-Schätzung einer Gauss-Verteilung (parametrische Dichteschätzung):

$$\varphi(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} (\det \boldsymbol{\Sigma})^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right)$$

$$L(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{m=1}^{M} \frac{1}{2} (\mathbf{x}^{(m)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(m)} - \boldsymbol{\mu}) + \sum_{m=1}^{M} \frac{1}{2} \ln \det \boldsymbol{\Sigma} + c$$



Eindimensional: $L(\mu, \sigma^2) = \sum_{m=1}^{M} \frac{(x^{(m)} - \mu)^2}{2\sigma^2} + M \ln \sigma + c$

ML-Schätzung für μ: Verschwindende Ableitung:

$$\frac{\partial}{\partial \mu} L(\mu, \sigma^2) != 0 = -\frac{1}{\sigma^2} \sum_{m=1}^{M} (x^{(m)} - \mu) \Rightarrow \boxed{\mu^{ML} = \frac{1}{M} \sum_{m=1}^{M} x^{(m)}}$$

ML-Schätzung für σ:

$$\frac{\partial}{\partial \sigma} L(\mu, \sigma^2) != 0 = -\sum_{m=1}^{M} \frac{(x^{(m)} - \mu^{ML})^2}{\sigma^3} + \frac{M}{\sigma} \Rightarrow \boxed{\sigma^{2ML} = \frac{1}{M} \sum_{m=1}^{M} (x^{(m)} - \mu^{ML})^2}$$

Bem: Bei Gauss-verteilten Daten mißt L gerade die mittlere quadratische Abweichung der Daten vom Mittel („quadratischer Fehler“, siehe später)

Lernen von Datenmodellen

Maximum-Likelihood Schätzung und Fehlerminimierung

- **Beispiele für Maximum-Likelihood Schätzung**
- **Maximum-Likelihood und Fehlerminimierung**
- **Kostenfunktionen**

Maximum-Likelihood Schätzung für Regression:

Likelihood für Input-Output-Paare $(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}),\ m = 1,\ldots,M$

$$p(D \mid \mathbf{w}) = \prod_{m=1}^{M} p(\mathbf{y}^{(m)}, \mathbf{x}^{(m)} \mid \mathbf{w}) = \prod_{m=1}^{M} p(\mathbf{y}^{(m)} \mid \mathbf{x}^{(m)}, \mathbf{w})\, p(\mathbf{x}^{(m)}) \quad \Rightarrow L(\mathbf{w}) = -\sum_{m=1}^{M} \ln p(\mathbf{y}^{(m)} \mid \mathbf{x}^{(m)}, \mathbf{w}) + c$$

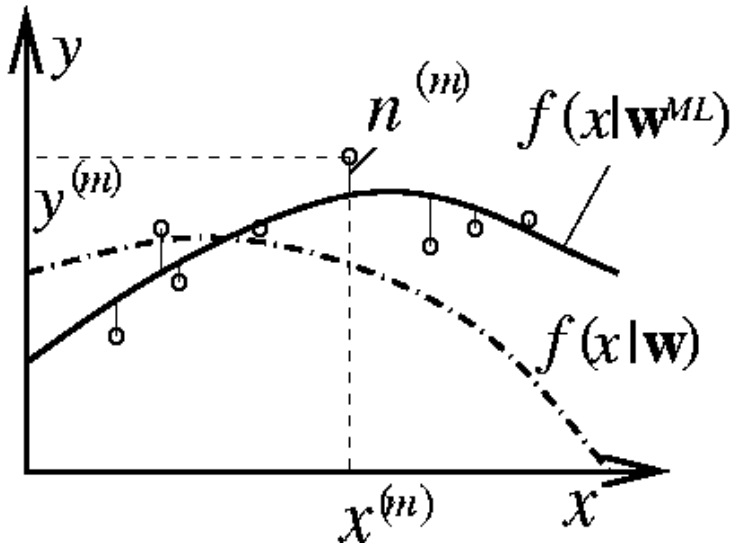
Datenmodell Regression: Alles, was nicht durch f erklärt werden kann, muss Rauschen sein

$$\mathbf{y}^{(m)} = \mathbf{f}(\mathbf{x}^{(m)} \mid \mathbf{w}) + \mathbf{n}^{(m)},\ \Rightarrow \mathbf{n}^{(m)} = \mathbf{y}^{(m)} - \mathbf{f}(\mathbf{x}^{(m)} \mid \mathbf{w})$$

Bei Rauschverteilung $p_n(\mathbf{n})$ **(ohne Konstanten):**

$$L(\mathbf{w}) = -\sum_{m=1}^{M} \ln p(\mathbf{y}^{(m)} \mid \mathbf{x}^{(m)}, \mathbf{w})$$

$$= -\sum_{m=1}^{M} \ln p_n(\mathbf{y}^{(m)} - f(\mathbf{x}^{(m)} \mid \mathbf{w}))$$



Bem: Für Gauss'sches Rauschen gilt:

$$L(\mathbf{w}) = \frac{1}{2\sigma^2} \sum_{m=1}^{M} (\| \mathbf{y}^{(m)} - f(\mathbf{x}^{(m)} \mid \mathbf{w}) \|^2)$$

ML-Schätzung bei Gaussischem Rauschen entspricht dem Least-Squares-Fit

- **Bsp./Wiederholung: Maximum-Likelihood für das lineare Modell**

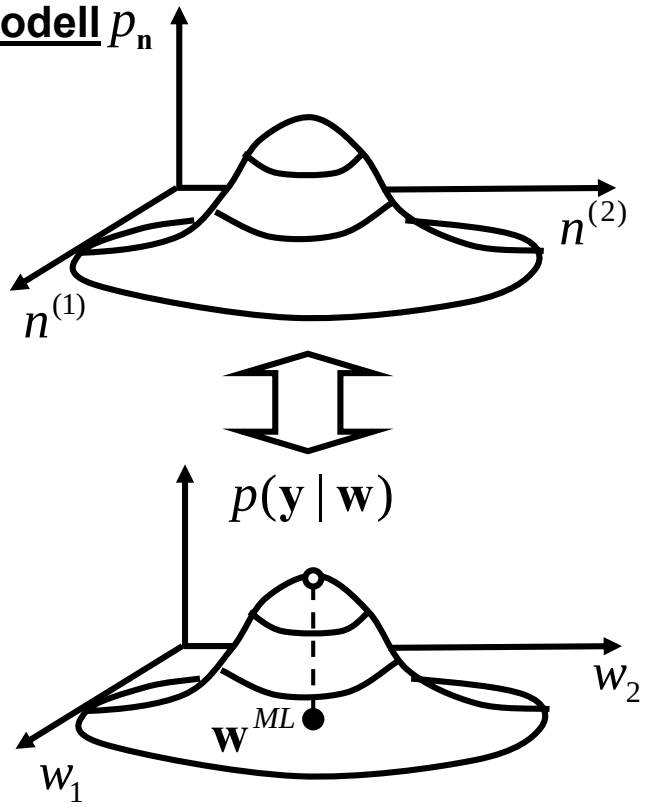
Likelihood: $p(D \mid \mathbf{w}) \equiv p(\mathbf{y} \mid \mathbf{w}) = p_{\mathbf{n}}(\mathbf{y} - \mathbf{H}\mathbf{w})$

Annahme: Gaussisches weißes Rauschen

$$p_{\mathbf{n}}(\mathbf{n}) = \prod_m p_n(n^{(m)}) = \prod_m \frac{1}{(2\pi\sigma_n^2)^{1/2}} \exp\left(-\frac{n^{(m)2}}{2\sigma_n^2}\right)$$

$$= \frac{1}{(2\pi\sigma_n^2)^{M/2}} \exp\left(-\frac{\mathbf{n}^T\mathbf{n}}{2\sigma_n^2}\right)$$

$$p(\mathbf{y} \mid \mathbf{w}) \propto \exp\left(-\frac{(\mathbf{y}-\mathbf{H}\mathbf{w})^T(\mathbf{y}-\mathbf{H}\mathbf{w})}{2\sigma_n^2}\right)$$



- **Maximum-Likelihood Parameter**

$$0! = \nabla_{\mathbf{w}} \ln p(\mathbf{y}|\mathbf{w}) = -\frac{1}{2\sigma_n^2}\nabla_{\mathbf{w}}(\mathbf{y}^T\mathbf{y} - 2\mathbf{w}^T\mathbf{H}^T\mathbf{y} + \mathbf{w}^T\mathbf{H}^T\mathbf{H}\mathbf{w})$$

$$= \frac{1}{\sigma_n^2}(\mathbf{H}^T\mathbf{y} - \mathbf{H}^T\mathbf{H}\mathbf{w})$$

$$= \frac{(\mathbf{H}^T\mathbf{H})}{\sigma_n^2}((\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y} - \mathbf{w})$$

$$\Rightarrow \hat{\mathbf{w}} = \mathbf{w}^{ML} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$$

analytisch berechenbar!

- **Beispiel: Hyperparameter im linearen Modell :**

Schätzer Rauschvarianz: $\hat{\sigma}_n^2 = \dfrac{\hat{\mathbf{n}}^T \hat{\mathbf{n}}}{tr(\mathbf{R})}$ $\quad \hat{\mathbf{n}} = \mathbf{y} - \mathbf{H}\hat{\mathbf{w}} = \mathbf{y} - \mathbf{H}(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}\mathbf{y} =: \mathbf{R}\mathbf{y}$

- **Bem:** **Rausch-Schätzung entspricht Schätzung eines Likelihood-Hyperparameters**

Likelihood-Hyperparam.: $p(D \mid \mathbf{w}) = p(D \mid \mathbf{w}, \beta) \quad p(D \mid \beta) = \int p(D \mid \mathbf{w}, \beta)\, p(\mathbf{w} \mid \alpha) d\mathbf{w}$

$$p(\beta \mid D) \equiv \frac{p(D \mid \beta)\, p(\beta)}{\int p(D \mid \beta)\, p(\beta) d\beta}$$

Hier: $p(D \mid \mathbf{w}, \beta) \equiv p(\mathbf{y} \mid \mathbf{w}, \sigma_n^2) = p_n(\mathbf{y} - \mathbf{H}\mathbf{w} \mid \sigma_n^2) \propto \exp\left(-\dfrac{(\mathbf{y} - \mathbf{H}\mathbf{w})^T (\mathbf{y} - \mathbf{H}\mathbf{w})}{2\sigma_n^2} \right)$

MAP/ML: $p(D \mid \beta) \equiv p(\mathbf{y} \mid \sigma_n^2) = \int p(\mathbf{y} \mid \mathbf{w}, \sigma_n^2)\, p(\mathbf{w}) d\mathbf{w} \approx p(\mathbf{y} \mid \mathbf{w}^{ML}, \sigma_n^2)$

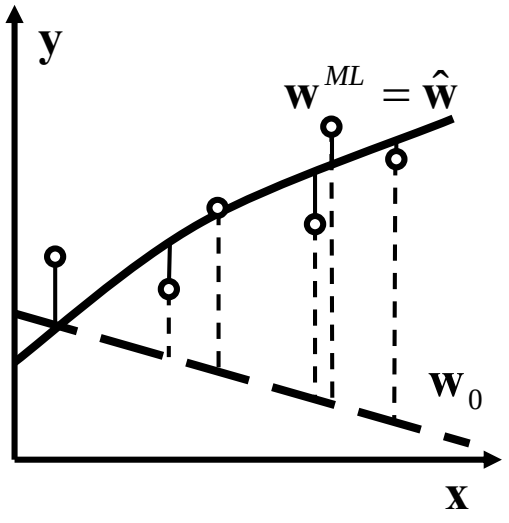
Maximiere bez: σ_n^2 $\qquad \dfrac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\dfrac{\hat{\mathbf{n}}^T \hat{\mathbf{n}}}{2\sigma_n^2} \right)$ $\qquad$ **(Siehe ML-Bsp):** $\hat{\sigma}_n^2 = \dfrac{\hat{\mathbf{n}}^T \hat{\mathbf{n}}}{M}$

Maximum-Likelihood und Fehlerminimierung

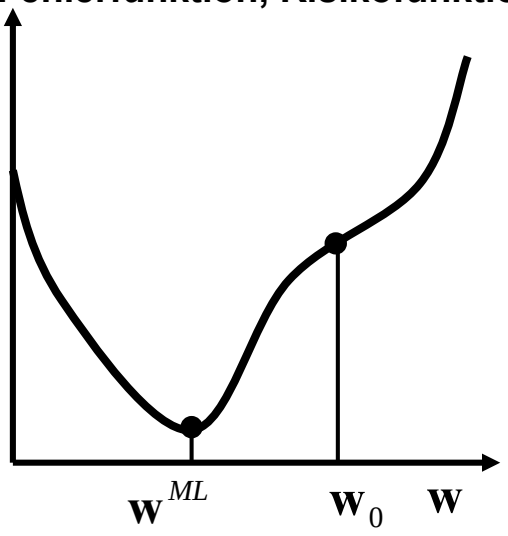
Betrachte überwachtes Lernproblem (Regression, Klassifikation)

- **<u>Likelihood:</u> Gegeben ein Rauschmodell $p_n(\mathbf{n})$, wie wahrscheinlich ist eine Abweichung des tatsächlichen vom vorhergesagten Output?**
- **<u>Negative log-Likelihood:</u> Wie unwahrscheinlich ist die Abweichung**
  Also: Wie groß ist der Fehler, den das Modell macht?

- **Teilweise synonym (beobachtet):**
  ++ negative log-Likelihood
  ++ neg. log. Rauschdichte
  -- „Risikofunktion“
  -- „Fehlerfunktion“
  -- „Verlustfunktion“
  -- „Energiefunktion“
- **Anpassen („Fitten“) eines Modells durch Risiko/Fehlerminimierung**



Fehlerfunktion, Risikofunktion



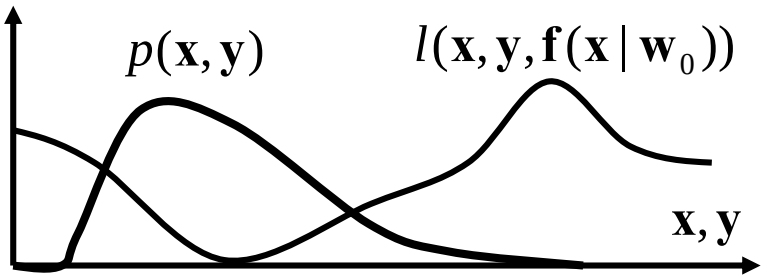
Fehler und empirischer Fehler

- **Def. Fehlerfunktion:** $l(\mathbf{x}, \mathbf{y}, \mathbf{f}(\mathbf{x} \mid \mathbf{w})) \geq 0$ **mit** $l(\mathbf{x}, \mathbf{y}, \mathbf{y}) = 0 \quad \forall \quad \mathbf{x}, \mathbf{y}$

  Bem: -- Es genügt, dass l **am wahren Output sein Minimum annimmt**

- **Mittlerer (erwarteter) Fehler:**

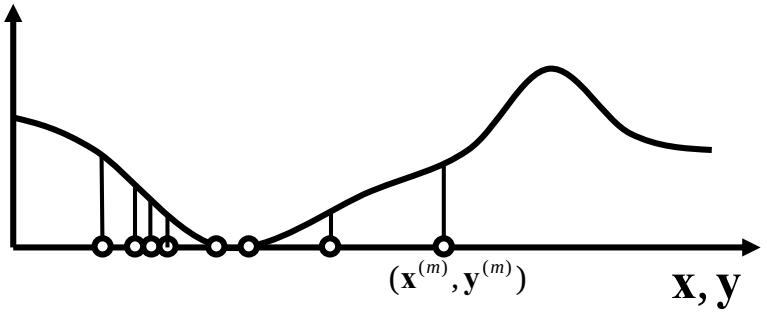
$$L_e(\mathbf{w}) = \int l(\mathbf{x}, \mathbf{y}, f(\mathbf{x} \mid \mathbf{w}))\, p(\mathbf{x}, \mathbf{y}) d\mathbf{x}\, d\mathbf{y}$$



- **Empirischer Fehler:**

  Dichte ist in der Regel unbekannt.
  Man hat Daten $(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}),\ m = 1, \ldots, M$

$$L(\mathbf{w}) = \frac{1}{M} \sum_{m=1}^{M} l(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}, f(\mathbf{x}^{(m)} \mid \mathbf{w}))$$



- **Bsp: Quadratischer Fehler**

$$L(\mathbf{w}) = \frac{1}{M} \sum_{m=1}^{M} l(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}, f(\mathbf{x}^{(m)} \mid \mathbf{w})) = \frac{1}{M} \sum_{m=1}^{M} (\mathbf{y}^{(m)} - f(\mathbf{x}^{(m)} \mid \mathbf{w}))^2$$

Minimierung des quadratischen Fehlers= „Least Squares Fit“

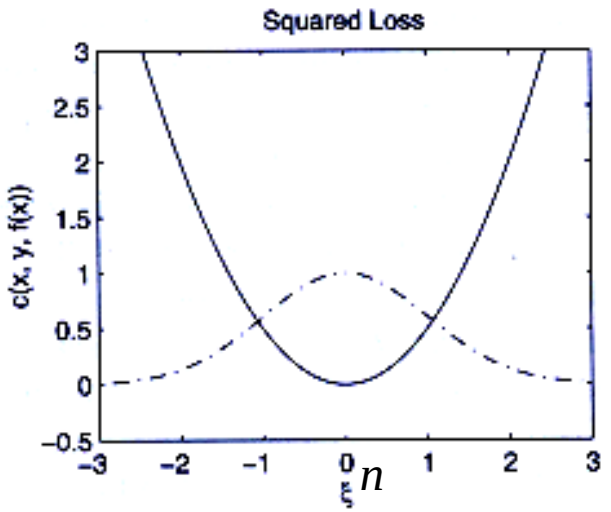
Fehlerfunktionen für Regression

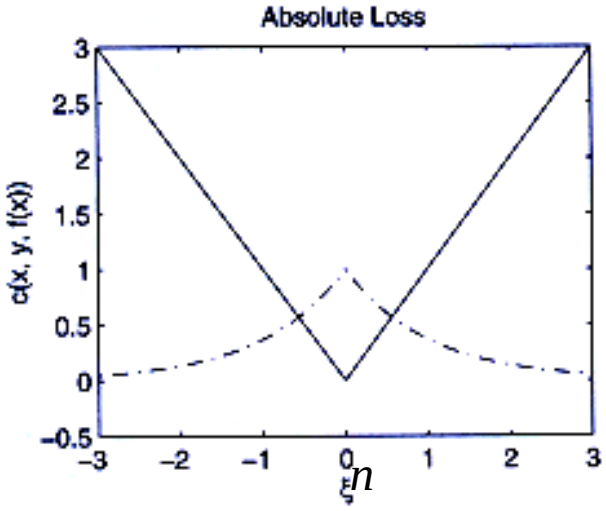
$$l(\mathbf{x},\mathbf{y},\mathbf{f}(\mathbf{x}\,|\,\mathbf{w})) = -\ln p(\mathbf{y}\,|\,\mathbf{x},\mathbf{w}) + c = -\ln p_n(\mathbf{y}-\mathbf{f}(\mathbf{x}\,|\,\mathbf{w})) + c$$

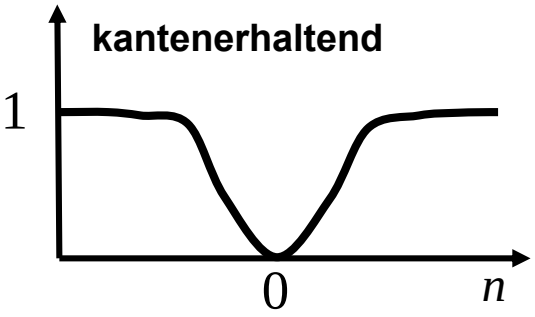
Rauschmodell	Verteilung $p_n(n)$	Fehler	Fehlerfunktion
Gauss	$\frac{1}{\sqrt{2\pi}}\exp(-\frac{n^2}{2})$	**quadratisch**	n^2
Laplace	$\frac{1}{2}\exp(-\lvert n\rvert)$	**absolut (oszilliert)**	$\lvert n\rvert$
Huber's robust	$\propto \begin{cases} \exp(-\frac{n^2}{2\sigma}), & \lvert n\rvert \le \sigma \\ \exp(\frac{\sigma}{2}-\lvert n\rvert) & \text{sonst} \end{cases}$	**Outlier-robust**	$\propto \begin{cases} \frac{1}{2\sigma}n^2 & \lvert n\rvert \le \sigma \\ \lvert n\rvert - \frac{\sigma}{2} & \text{sonst} \end{cases}$
ε-insensitiv	$\propto \begin{cases} 1/2(1+\varepsilon), & \lvert n\rvert \le \varepsilon \\ \exp(\lvert n\rvert-\varepsilon)/2(1+\varepsilon) & \text{sonst} \end{cases}$	**ε-insensitiv**	$\propto \begin{cases} 0, & \lvert n\rvert \le \varepsilon \\ \lvert n\rvert-\varepsilon & \text{sonst} \end{cases}$
	——	**Kantenerhaltend**	$1-\exp(-\frac{n^2}{2\sigma^2})$

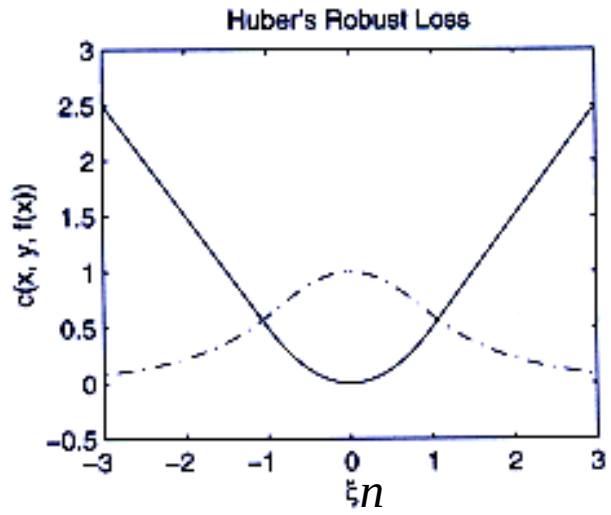
Bem: Es gibt Fehlerfunktionen, für die kein äquivalentes Rauschmodell existiert

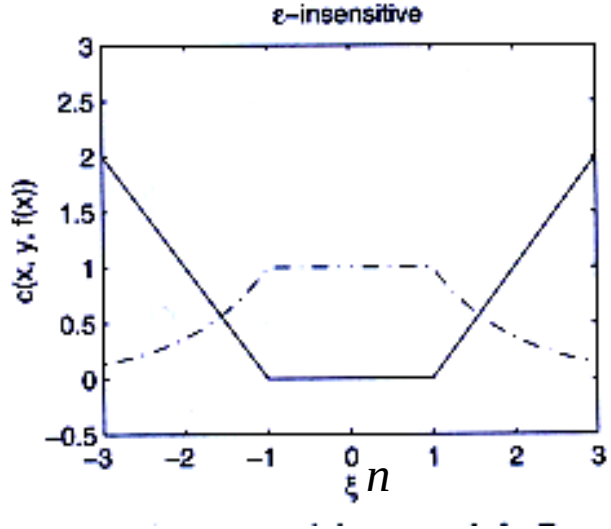
Fehlerfunktionen: Beispiele für Regression











Fehlerfunktionen für binäre Klassifikation

- **Likelihood: Wie groß ist die Wahrscheinlichkeit einer korrekten Klassenzuteilung?**

$$\Pr(y=1 \mid f(\mathbf{x} \mid \mathbf{w})) \equiv \Pr(y \mid \mathbf{x}, \mathbf{w})$$

- **Fehlerfunktion:**

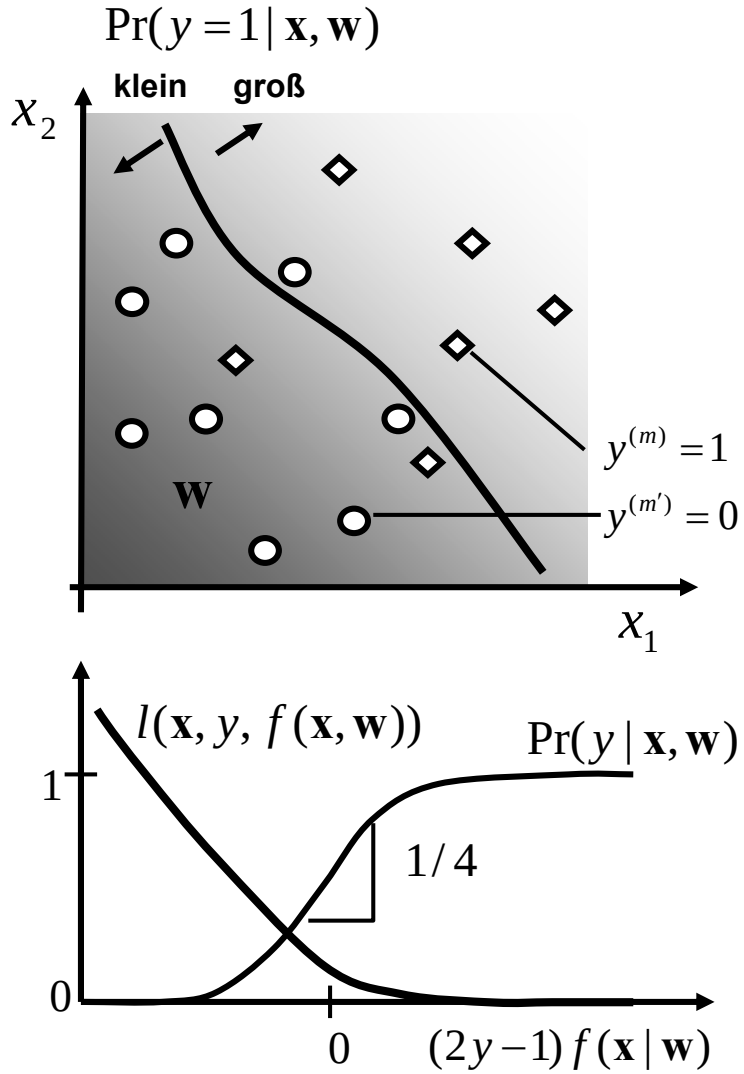
$$l(\mathbf{x}, y, f(\mathbf{x} \mid \mathbf{w})) = -\ln \Pr(y \mid \mathbf{x}, \mathbf{w})$$

- **Logistische Fehlerfunktion:**

$$\Pr(y=1 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1+\exp(-f(\mathbf{x} \mid \mathbf{w}))}$$

$$l(\mathbf{x}, y=1, f(\mathbf{x} \mid \mathbf{w})) = \ln(1+\exp(-f(\mathbf{x} \mid \mathbf{w})))$$

- **Logistischer empirischer Fehler:**

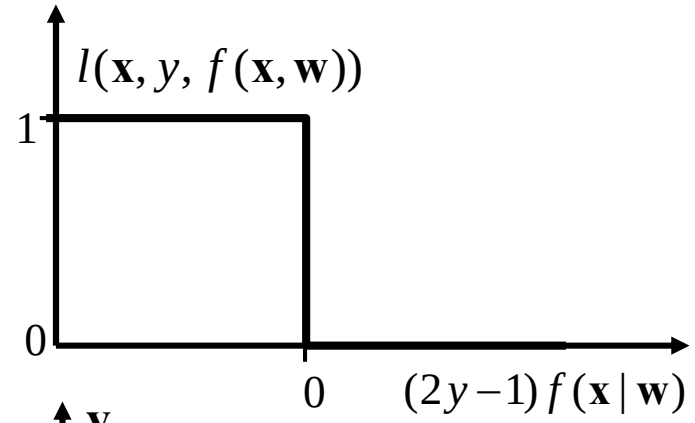


$$L(\mathbf{w}) = \sum_{y^{(m)}=1} \ln(1+\exp(-f(\mathbf{x}^{(m)} \mid \mathbf{w})) + \sum_{y^{(m)}=-1} \ln(1+\exp(+f(\mathbf{x}^{(m)} \mid \mathbf{w})) = \sum_{m=1}^{M} \ln(1+\exp(-(2y^{(m)}-1) f(\mathbf{x}^{(m)} \mid \mathbf{w}))$$

Weitere Fehlerfunktionen

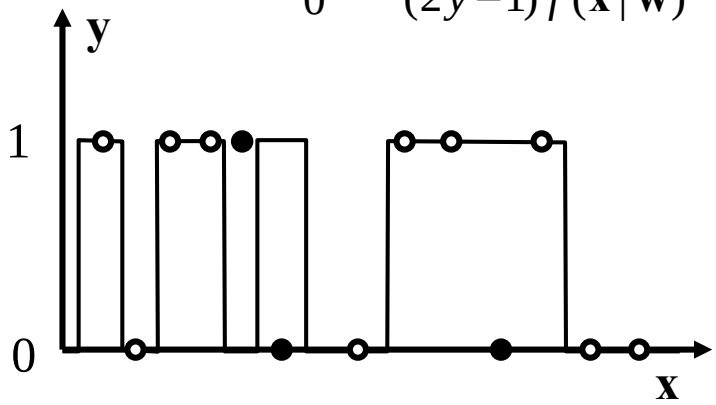
- **Falsch-Klassifikations-Zähler**

$$l(\mathbf{x}, y, f(\mathbf{x}\,|\,\mathbf{w})) = \begin{cases} 0, & (2y-1)\, f(\mathbf{x}\,|\,\mathbf{w}) \geq 0 \\ 1, & \text{sonst} \end{cases}$$



- **Inputabhängiger Falsch-Klassifikations-Zähler (Bsp: Klassifikation Steine und Diamanten)**

$$l(\mathbf{x}, y, f(\mathbf{x}\,|\,\mathbf{w})) = \begin{cases} 0, & y = f(\mathbf{x}\,|\,\mathbf{w}) \\ l_0(\mathbf{x}), & \text{sonst} \end{cases}$$



- **Outputabhängiger Falsch-Klassifikations-Zähler (Bsp: Klassifikation Krebs/gesund)**

$$l(\mathbf{x}, y, f(\mathbf{x}\,|\,\mathbf{w})) = \begin{cases} 0, y = f(\mathbf{x}\,|\,\mathbf{w}) \\ l_1, y = 1, f(\mathbf{x}\,|\,\mathbf{w}) = 0 \\ l_2, y = 0, f(\mathbf{x}\,|\,\mathbf{w}) = 1 \end{cases}$$

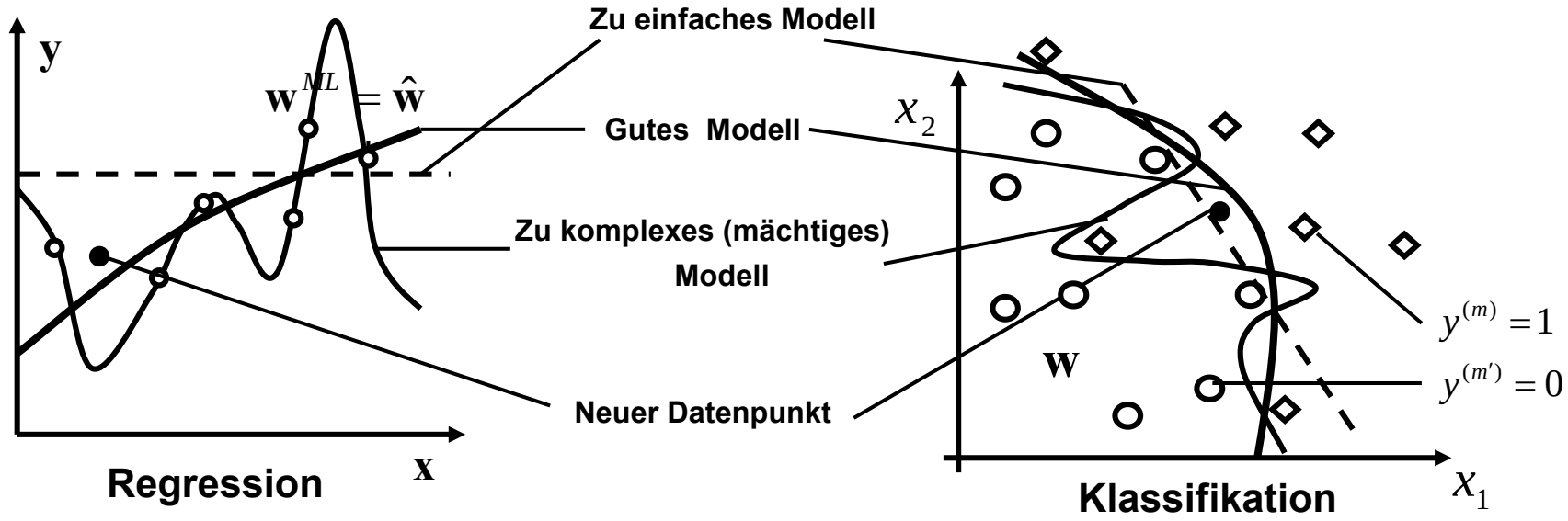
Lernen von Datenmodellen

Generalisierung und Regularisierung

- **Modell-Komplexität und Varianz-Bias Problem**
- **Generalisierung durch Regularisierung**
- **Kreuzvalidierung: Optimierung der Modellkomplexität**
- **Anwendungsbeispiel für Regularisierung: Funktionelle Kernspin-Bilder**

Generalisierung und Regularisierung

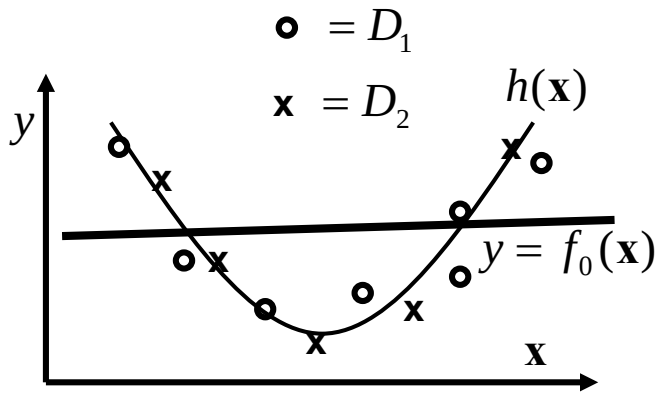
- **Betrachte überwachtes Lernproblem mit endlichem Datensatz** $(\mathbf{x}^{(m)}, \mathbf{y}^{(m)}),\ m = 1, \ldots, M$



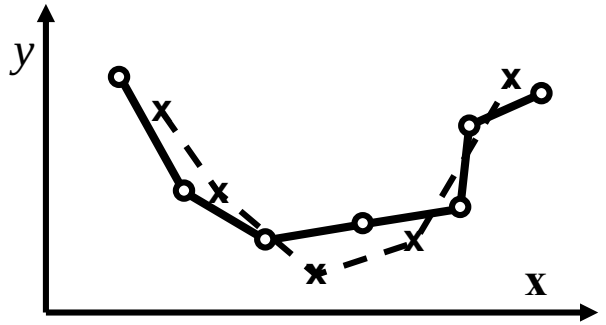
- **Beobachtungen:**
  - **-- Ein genügend komplexes Modell kann die Trainingsdaten beliebig genau erklären (kleiner Trainingsfehler)**
  - **-- Neue Datenpunkte werden dann aber schlechter erklärt (großer Testfehler, schlechte Generalisierungsfähigkeit, „Overfitting“)**
- **Finden der statistischen Struktur: Erkläre Daten mit möglichst einfachem Modell**

Etwas formaler: Das Bias-Varianz Problem

- **Ziel der Datenmodellierung:**
  -- Modellierung der statistischen Struktur h
  -- nicht Modellierung eines speziellen Datensatzes
- **Betrachte mittleres Verhalten über viele Datensätze:**
  $\mathbf{D} = \{D_1, D_2, ...\}$
- **Bias: Wie stark weicht das über alle Datensätze gemittelte Modell von dem wahren Modell ab?**
- **Varianz: Wie stark variiert das Modell (wie stark hängt es vom einzelnen Datensatz ab?)**
- **Sei** $E_{\mathbf{D}}[.]$ **der Erwartungswert über alle Datensätze**
  Für Regression und quadratischen Fehler:



Großer Bias, Varianz=0



Kleiner Bias, große Varianz

$$(f(\mathbf{x}\,|\,\mathbf{w}) - h(\mathbf{x}))^2 = (f(\mathbf{x}\,|\,\mathbf{w}) - E_{\mathbf{D}}[f(\mathbf{x}\,|\,\mathbf{w})] + E_{\mathbf{D}}[f(\mathbf{x}\,|\,\mathbf{w})] - h(\mathbf{x}))^2$$

$$E_{\mathbf{D}}[(f(\mathbf{x}\,|\,\mathbf{w}) - h(\mathbf{x}))^2] = (E_{\mathbf{D}}[f(\mathbf{x}\,|\,\mathbf{w})] - h(\mathbf{x}))^2 + E_{\mathbf{D}}[(f(\mathbf{x}\,|\,\mathbf{w}) - E_{\mathbf{D}}[f(\mathbf{x}\,|\,\mathbf{w})])^2]$$

Minimiere gemeinsam: Bias Varianz

Bias-Varianz-Trade-off in Bayes-Modellen

- **Betrachte Bayes-Schätzer:** $p(\mathbf{w} \mid D) = \frac{1}{p(D)} p(D \mid \mathbf{w}) p(\mathbf{w})$
- **MAP:**
- **Maximierung der Likelihood: Erzielung eines kleinen Bias (guter Fit der Datenpunkte)**
- **Der Prior kann zu komplexe Modelle bestrafen („Occam's Razor")**
  -- Maximierung des Prior favorisiert einfache Modelle
  -- Erzielung einer kleinen Varianz (kein overfitting)
- **Maximierung der Evidenz: Bestimmung der Hyperparameter**
  z. B. Optimierung der relativen Gewichtung von Bias und Varianz-Minimierung
- **Bsp: Weight-Decay Prior** $p(\mathbf{w}) = \frac{1}{Z(\alpha)} \exp(-\frac{\alpha}{2} \|\mathbf{w}\|^2)$
- **Favorisiert kleine Gewichtswerte**
  => favorisiert glattere Kurven (y ändert sich „langsamer" mit $\mathbf{x}$)
- **Negativer Log-Prior entspricht Regularisierungsterm**
  Hyperparameter α gewichtet den Regularisator

$$-\ln p(\mathbf{w}) \equiv \alpha R(\mathbf{w}) = \frac{\alpha}{2} \|\mathbf{w}\|^2 + c$$

Regularisierung: Favorisierung von Modellen mit bestimmter (z.B. niedriger) Komplexität

Bayes-Formalismus

-- negative log-Likelihood

-- negativer log-Prior

-- Max. a Posteriori

Schätztheorie

-- Fehlerfunktion L

-- Regularisierungsfunktion R

-- $F(\mathbf{w}) = L(\mathbf{w}) + \alpha R(\mathbf{w}) =! \min$

Beobachtung

-- Rauschen enthält alle (auch hohe) Frequenzen

-- Deterministische Zusammenhänge sind oft glatt (niedrige Frequenzen)

-- Sinnvolle Regularisierung: Bevorzuge „glatte“ Modelle

-- Glattere Modelle <= weniger Parameter, oder kleinere Gewichte

Beispiele für Regularisierungsfunktionen

- **Weight-Decay** $R(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$
- **Kurven-basierte Regularisierung** $R(\mathbf{w}) = \sum_{m=1}^{M}\sum_{j=1}^{c}\sum_{i=1}^{d}\left(\left.\frac{\partial^2 f(\mathbf{x}\,|\,\mathbf{w})}{\partial x_i \partial x_j}\right|_{\mathbf{x}^{(m)}}\right)^2$

Regularisierung und Generalisierung

- **Regularisiertes Modell hat niedrige Varianz**
  => hängt nicht stark vom speziellen Datensatz ab
  => kann gut auf neue Datensätze übertragen werden
  => kann gut generalisieren.

Optimierung von Regularisierungstermen

- **Problem: -- Regularisierungsterm ist Hypothese des Modellierers**
  -- kann nicht direkt aus Daten geschätzt werden
  -- Wie bestimmt man die freien Parameter (z.B. Gewicht α)?
- **Methode 1: Schätzung als Hyperparameter aus der Bayes-Evidenz**
- **Methode 2 (praktisch sehr wichtig): Kreuzvalidierung.**

Kreuzvalidierung

- **Beurteile das Modell anhand seiner Generalisierungsfähigkeit:**
- **Teile dazu die Daten in Trainings- und Validierungsdaten auf**
- **Lerne an Trainingsdaten; Bestimme Performance anhand der Validierungsdaten**
- **Bsp. für Algorithmus :**

1. Selektiere ein Regularisierungsgewicht α für sehr einfache Modelle

2. Teile dazu die Daten zufällig in Trainings- und Validierungsdaten auf: $D = D_{tr}, D_{te}$

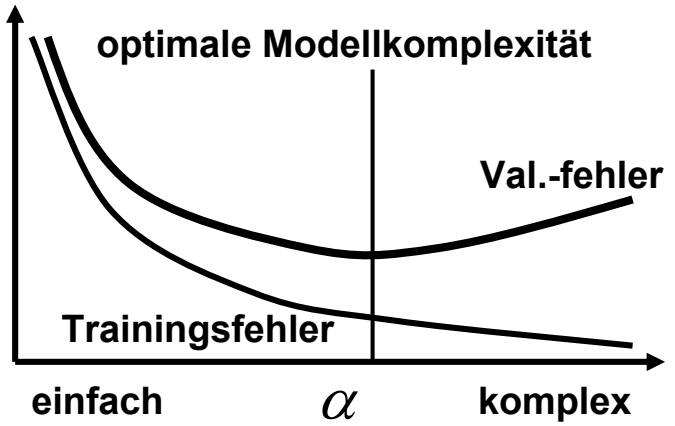
3. Optimiere $F(\mathbf{w})$ nur anhand der Trainingsdaten (siehe nächstes Kapitel)

4. Bestimme den Fehler anhand der Validierungsdaten

5. Wiederhole Schritte 2-4 oftmals mit neuen D_{tr}, D_{te}
=> mittlerer Validierungsfehler

6. Ändere α in Richtung komplexere Modelle,
gehe zu Schritt 2

7. Stoppe, wenn der Validierungsfehler minimal ist

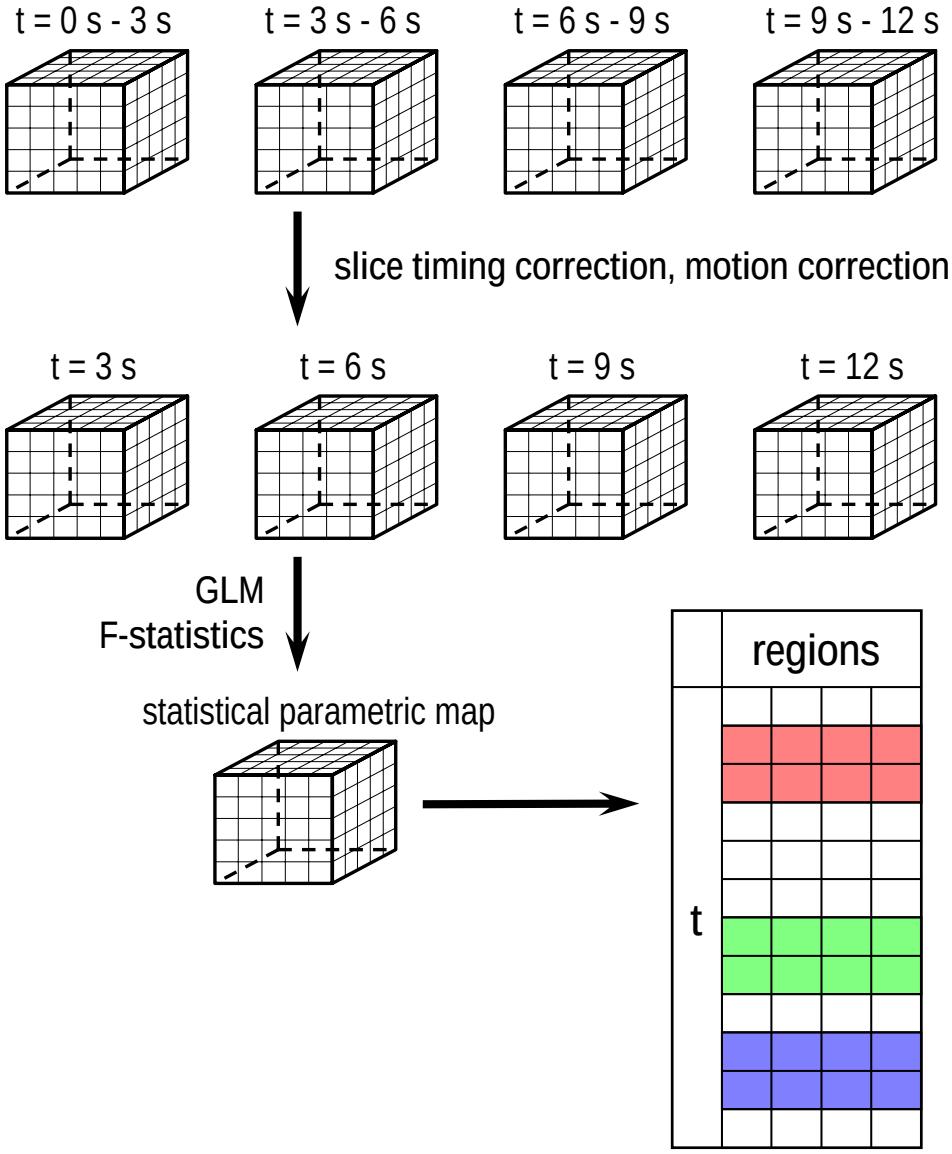


Anwendungsbeispiel für Regularisierung: Analyse von fMRI-Daten

Technik: funktionelle Kernspintomographie

- **„functional Magnetic Resonance Imaging“ (fMRI)**
- **Magnetmomente der Atome werden in einem starken Magnetfeld angeregt (Magnetresonanz)**
- **Sie relaxieren in den Grundzustand zurück**
- **Die Relaxation wird durch Gradientenfelder ortsabhängig gemacht (=> Imaging)**
- **Die Relaxation ist abhängig von der molekularen Umgebung**
  - **-- Wasserstoffkonzentration: strukturelle MRI**
  - **-- Paramagnetische Substanzen (Hbr): funktionelle MRI**

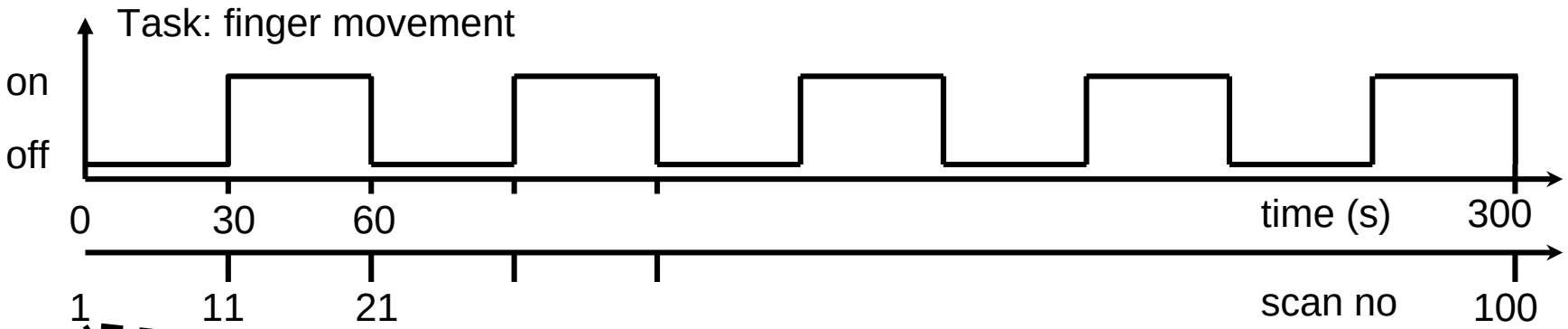
fMRI Data Format

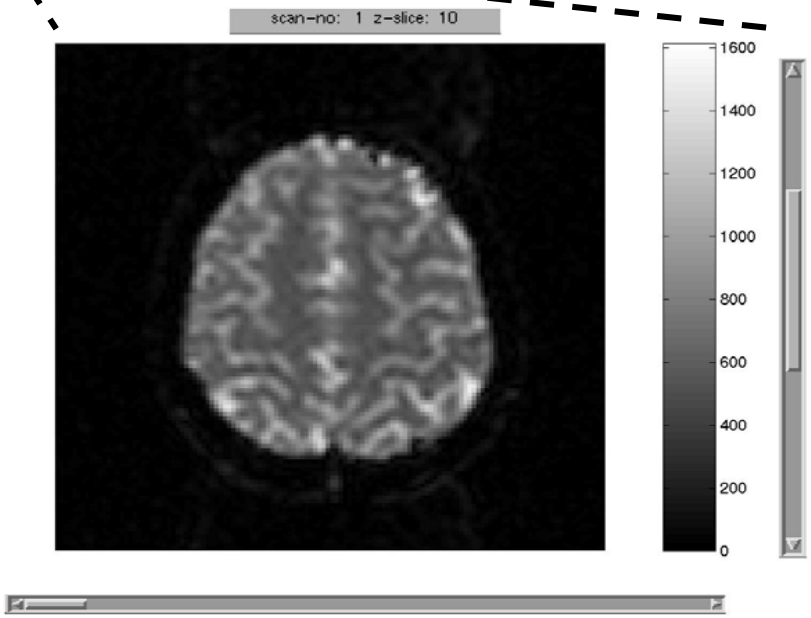


Task:

- **Extract small signal**
- **Separate signal components (BOLD from structural signal, background)**
- **Separate signal from noise**

Data Set mrea-g: Format





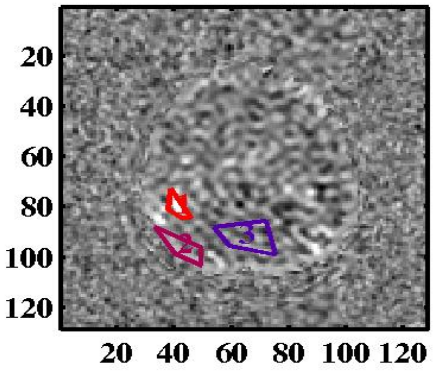
Raw data:

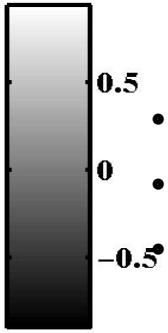
- EPI scans
- 100 scans,
- 128x128 slice size, 16 slices
- Scan time: 1.6 sec
- Scan interval: 3 sec

Raw data: scan 1, slice 10

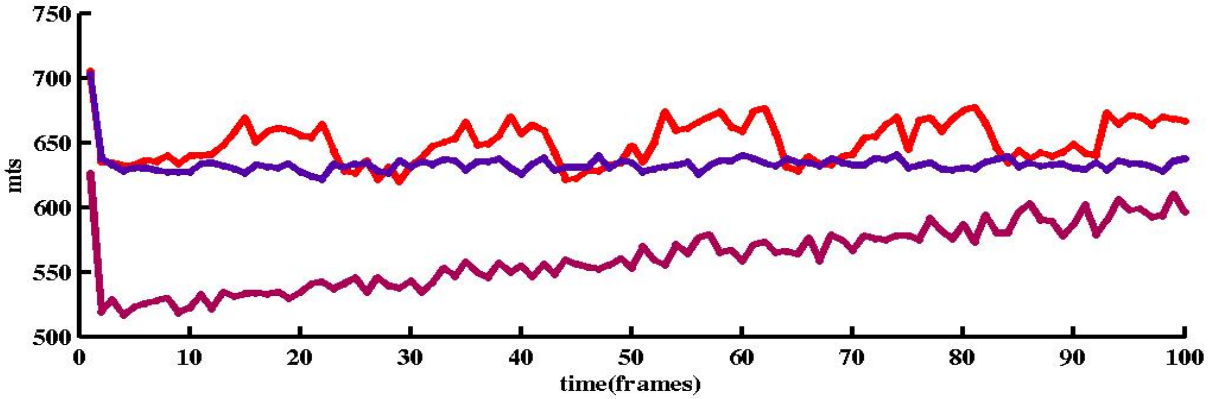
Data Set mrea-g: Visual Inspection of Time Series

Mean time series over three regions: Signal, drift, and background





- Task-related response
- linear drift present
- First frame has higher values (magnetic transient?)



General Linear Model: Design Matrix

Principle: $\mathbf{X} = \mathbf{Hw} + \mathbf{n}$

Mrea-g data set

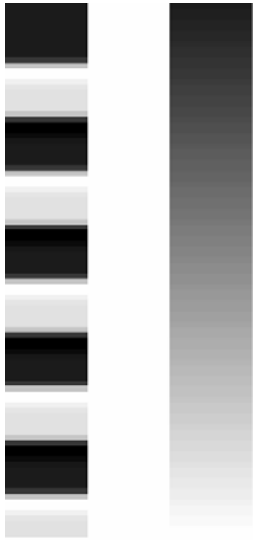
X = matrix of time series

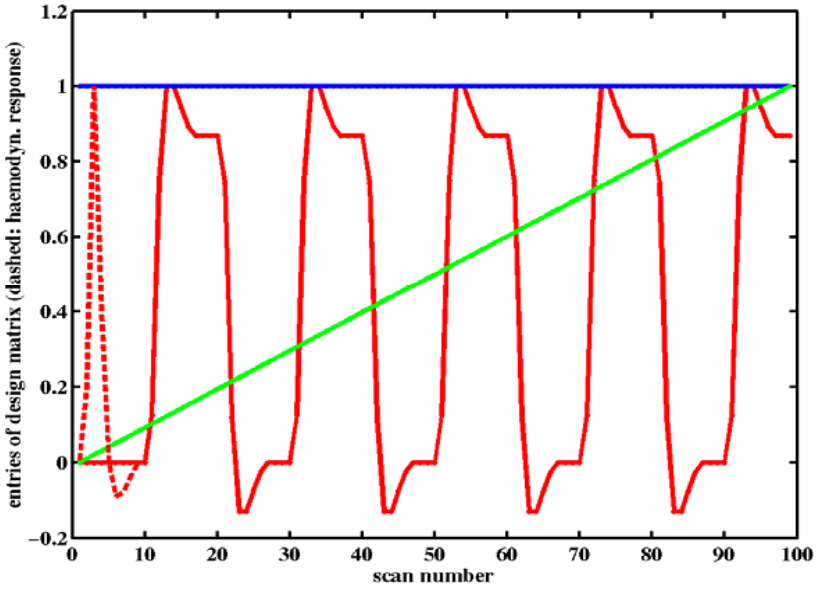
H = Design matrix

w= parameter (estimate used for t-test)

n = noise (estimate used for t-test)

Entries of design matrix used:

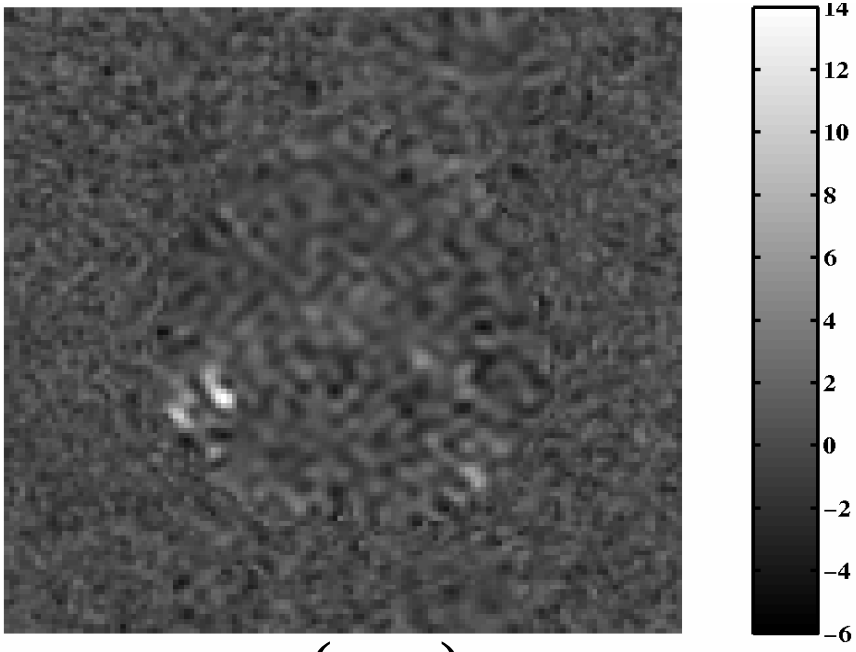




GLM: Statistical Parametric Maps (SPM)

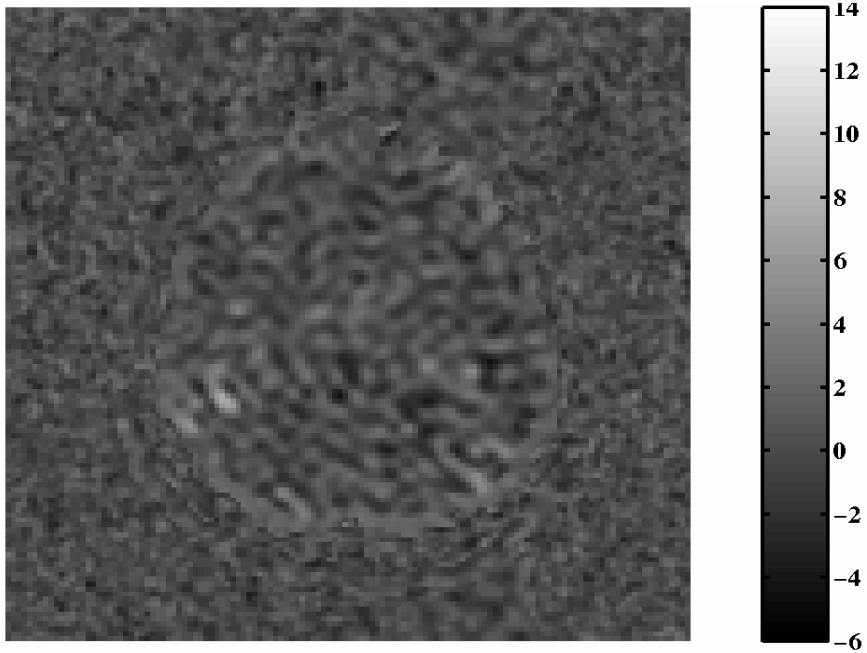
Mrea-g data set

3 component GLM on raw data



$w_1(x, y)$

Raw t-test on raw data



$t(x, y)$

- Higher absolute t-value in GLM (due to convolution with haemodynamic response
- t-values along head border are not suppressed (beause drift is modeled)

Regularized GLM

Introduce regularizing terms (Bayes: Priors):

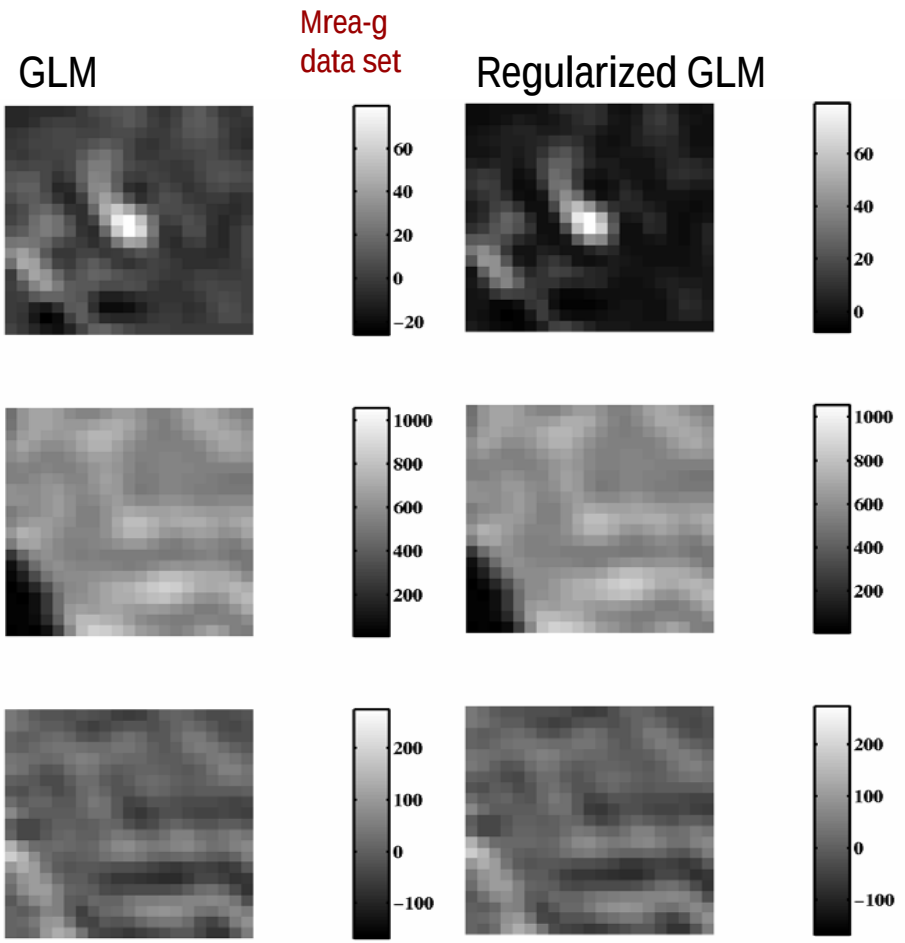
$$P(w(x,y)) \equiv P(\mathbf{w}) = P_A(\mathbf{w}) P_{MRF}(\mathbf{w})$$

- **For positive sign of BOLD signal**

$$-\ln P_A = \sum_{x,y} \ln\left(1+\exp(-w(x,y)/\sigma_A)\right)$$

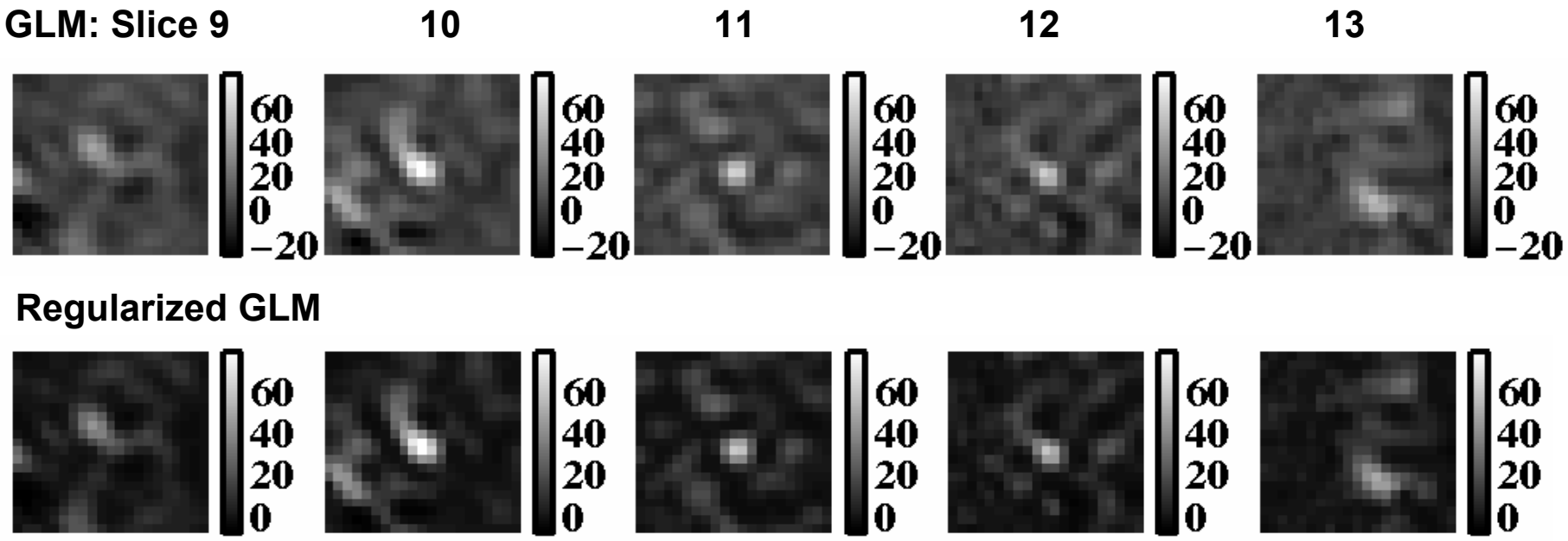
- **For smoothness in space (Markov Random Field Prior)**

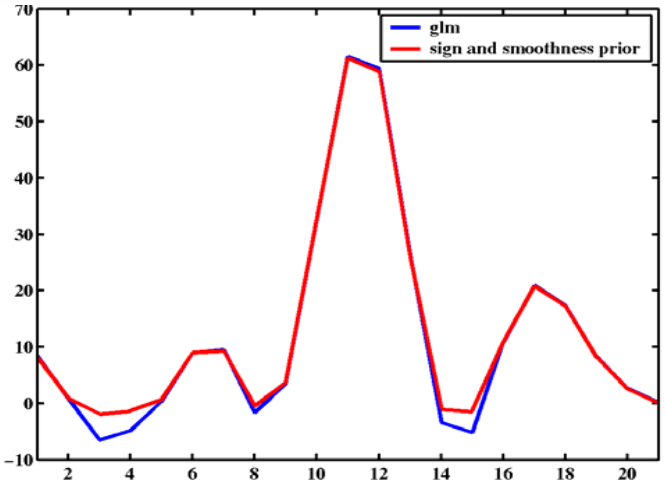
$$-\ln P_{MRF} = \sum_{x,y} \frac{(w(x+1,y)-w(x,y))^2 + (w(x,y+1)-w(x,y))^2}{2\sigma_{MRF}^2}$$



Regularized GLM: Stimulus-Related Signal

Mrea-g data set





Top: Five consecutive slices (small dataset)

Left: 1D-profile through active region (sl. 11, line 12)

Regularized GLM (Edge preserving MRF)

Introduce regularizing terms (Bayes: Priors):

$$P(w(x,y)) \equiv P(\mathbf{w}) = P_A(\mathbf{w}) P_{EPS}(\mathbf{w})$$

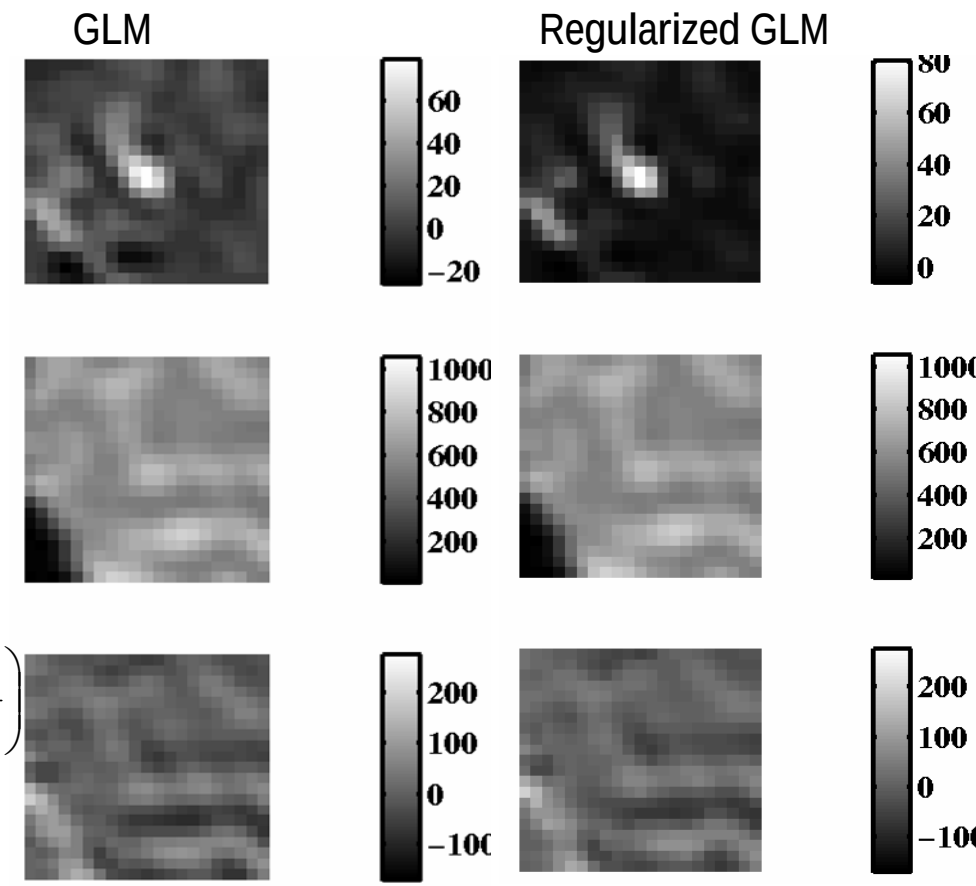
- **For positive sign of BOLD signal**

$$-\ln P_A = \sum_{x,y} \ln\left(1 + \exp(-w(x,y)/\sigma_A)\right)$$

- **For edge-preserving smoother**

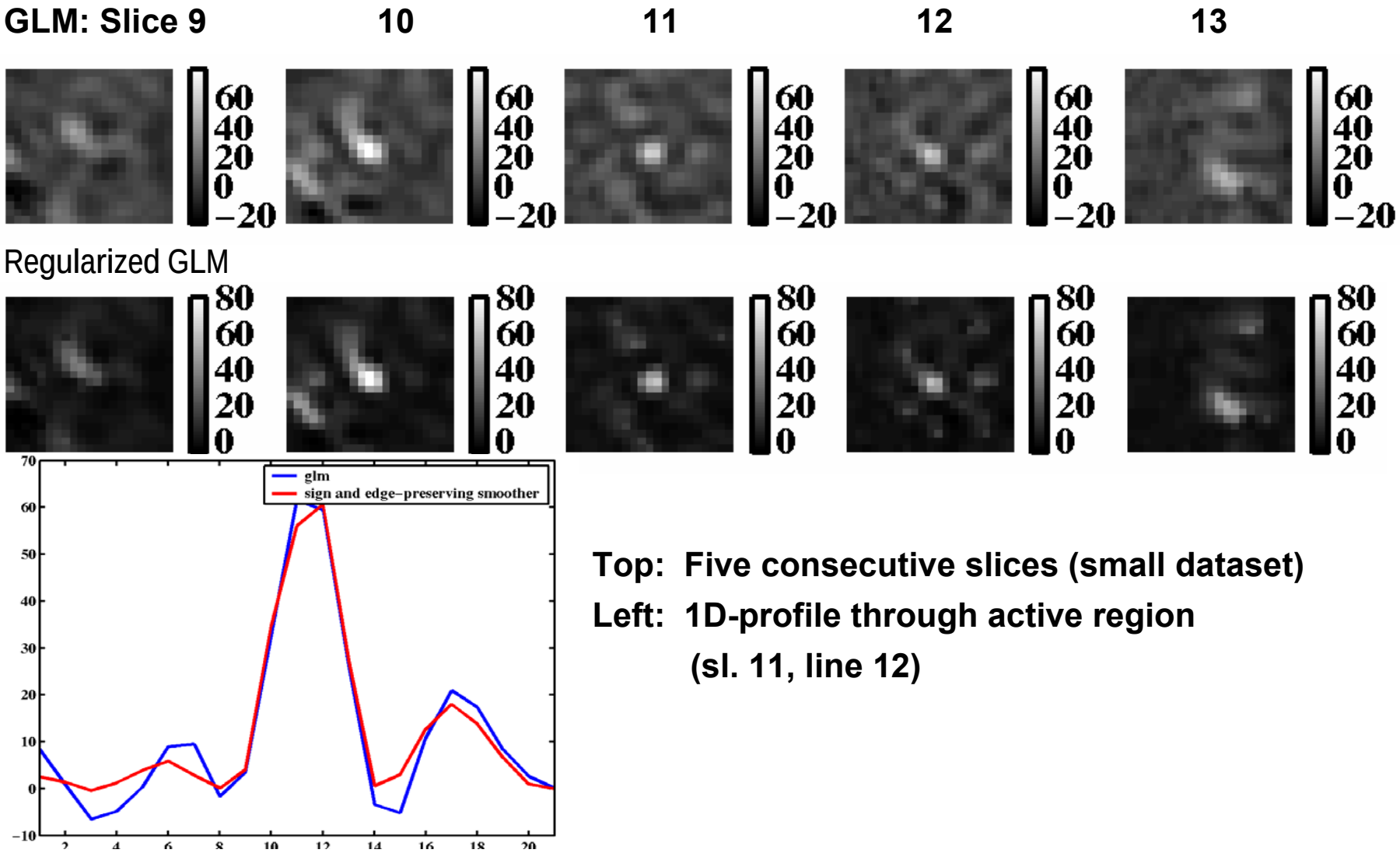
$$-\ln P_{EPS} =$$

$$\sum_{x,y} 1 - \exp\left(\frac{(w(x+1,y)-w(x,y))^2 + (w(x,y+1)-w(x,y))^2}{2\sigma_{MRF}^2}\right)$$



Regularized GLM (EP-MRF): Stimulus-Related Signal

Mrea-g data set



Top: Five consecutive slices (small dataset)

Left: 1D-profile through active region (sl. 11, line 12)

Optimierungsverfahren: Motivation

Bayes-Formalismus

-- negative log-Likelihood

-- negativer log-Prior

-- Max. a Posteriori

$$-\ln p(D \mid \mathbf{w}) - \ln p(\mathbf{w}) + const = \min$$

Schätztheorie

-- Fehlerfunktion L

-- Regularisierungsfunktion R

-- $F(\mathbf{w}) = L(\mathbf{w}) + \alpha R(\mathbf{w}) =! \min$

Lernen von Datenmodellen

Optimierungsverfahren

- **Optimierung konvexer Funktionen**
- **Optimierungsprobleme mit Randbedingungen**
- **Nichtkonvexe Optimierungsprobleme**

Optimierung konvexer Funktionen

- **Häufiges Problem bei Lernen: Maximierung bzw. Minimierung einer Funktion**
  -> <u>Optimierung</u>
  Bsp: Maximum Likelihood, Maximum a Posteriori, Fehlerminimierung ...

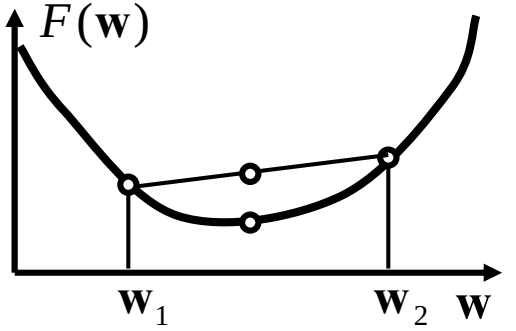
- **<u>Def: Konvexe Funktion</u>: Für alle** $\mathbf{w}_1, \mathbf{w}_2$

$$F(\lambda \mathbf{w}_1 + (1-\lambda)\mathbf{w}_2) \leq \lambda F(\mathbf{w}_1) + (1-\lambda)F(\mathbf{w}_2), \quad 0 \leq \lambda \leq 1$$

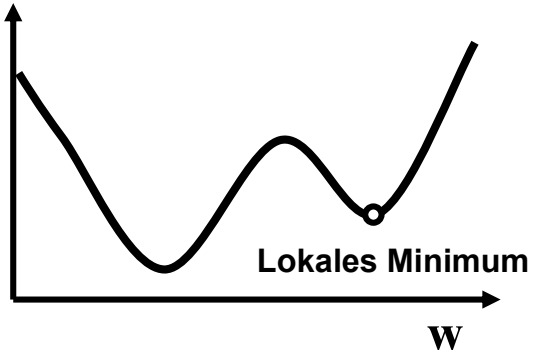
- **Für konvexe Optimierungsprobleme (Minimierung) gilt:**
  -- Es gibt genau ein Minimum
  -- Es gibt keine lokalen Minima
  -- Maxima liegen am Rand des Definitionsbereichs
  -- Viele Verfahren funktionieren noch bei „gutartig“ nicht-konvexen Funktionen

- **Ab jetzt: <u>Minimierung</u>**

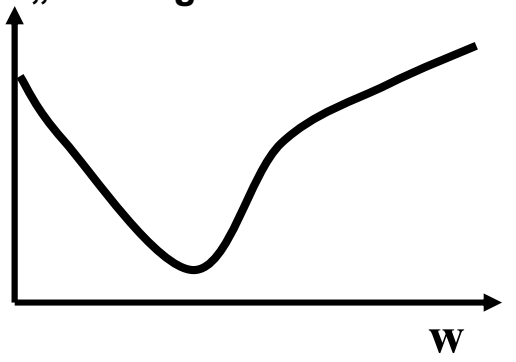
Konvexes Opimierungsproblem



Nichtkonvex

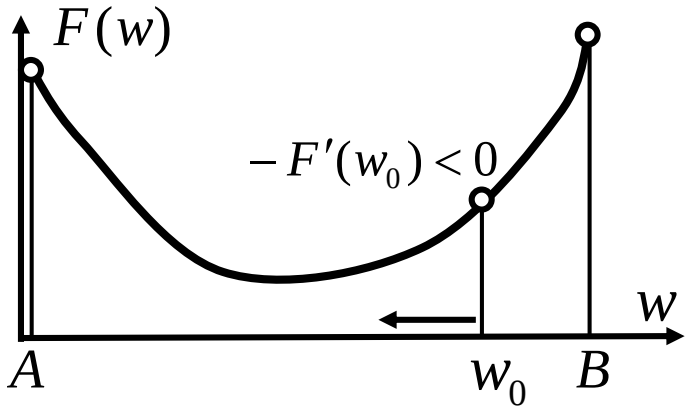


„Gutartig“ nichtkonvex



Funktionen einer Variable $F(w)$

- ***1D-Gradientenabstieg:***
  Gehe ein kleines Stück Richtung $-\frac{dF}{dw} = -F'(w)$

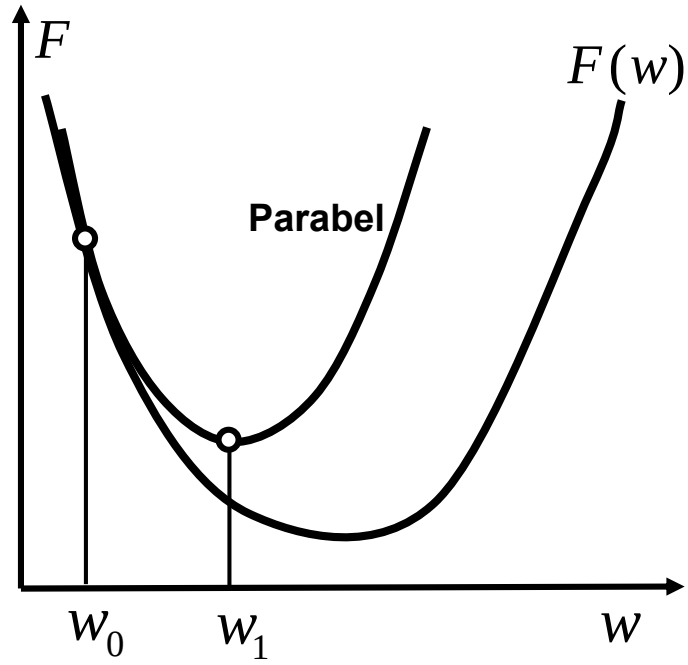


- ***Intervallschachtelung***
  -- Start an Intervallrändern A, B

  -- Auf Welcher Seite ist das Minimum?

  $F'((A+B)/2) > 0 \Rightarrow B = (A+B)/2$
  $F'((A+B)/2) < 0 \Rightarrow A = (A+B)/2$

  -- bis: $F'((A+B)/2) < \varepsilon \Rightarrow res = (A+B)/2$

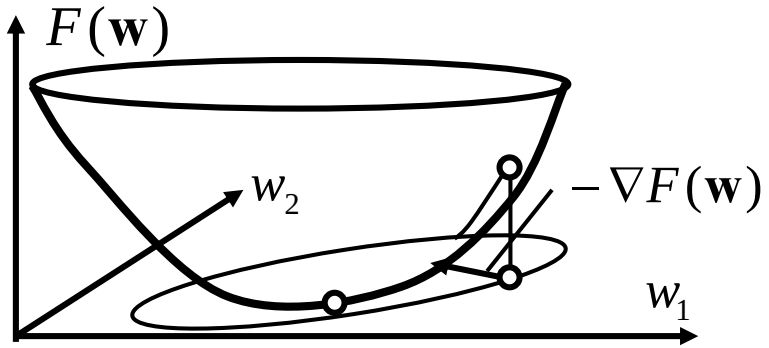


- ***Newton-Verfahren:***
  -- Start mit bel. w_0
  -- Taylorentwicklung der Funktion als Parabel

  $F(w) \approx F(w_n) + F'(w_n)(w - w_n) + \frac{1}{2} F''(w_n)(w - w_n)^2$

  -- Nehme Min der Parabel: $w_{n+1} = w_n - \frac{F'(w_n)}{F''(w_n)}$

Funktionen mehrerer Variablen $F(\mathbf{w})$



- **Gradientenabstieg:**

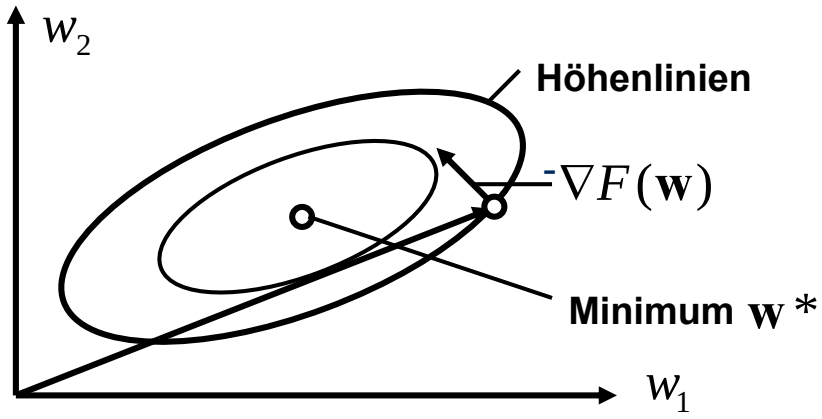
  Gehe ein kleines Stück Richtung des negativen Gradienten (steilster Abstieg)

$$\Delta \mathbf{w}_n = -\eta \nabla F(\mathbf{w}_n) = -\eta grad\ F(\mathbf{w}_n) =: -\eta \mathbf{g}_n =$$

$$-\eta \left(\frac{\partial F}{\partial w_1}(\mathbf{w}_n), \ldots, \frac{\partial F}{\partial w_d}(\mathbf{w}_n) \right)$$

  bis $\left\| \Delta \mathbf{w}_n \right\| < \varepsilon$

- **Probleme:**
  - **-- Steilster Abstieg zeigt nicht immer zum Minimum**
  - **-- Abstieg „fängt sich“ an flachen Stellen**
  - **-- enthält willkürliche Lernschrittweite η**
  - **-- kann unnötig langsam sein**
  - **-- kann oszillieren**



- **Newton-Verfahren (Hesse-Matrix):**

Analog 1D-Fall: Nähere F als quadratische Funktion

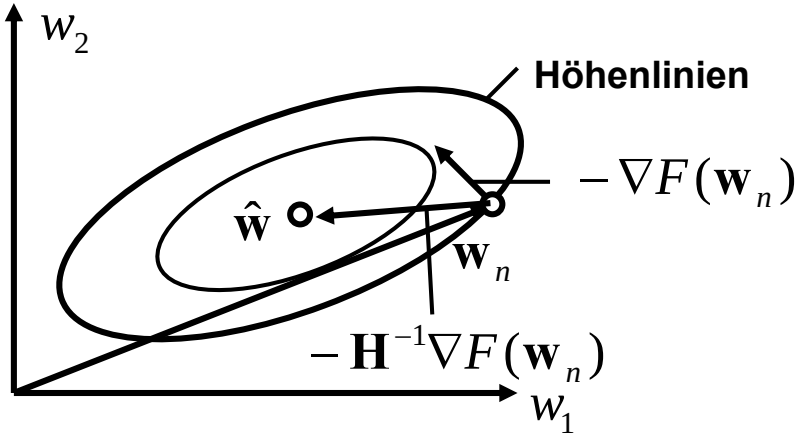
$$F(\mathbf{w}) \approx F(\mathbf{w}_n) + (\mathbf{w}-\mathbf{w}_n)^T \cdot \nabla F(\mathbf{w}_n) + \tfrac{1}{2}(\mathbf{w}-\mathbf{w}_n)^T \mathbf{H}(\mathbf{w}_n)(\mathbf{w}-\mathbf{w}_n)$$

$$\big(\mathbf{H}(\mathbf{w})\big)_{ij} = \frac{\partial^2 F}{\partial w_i \partial w_j}(\mathbf{w})$$ **Hessematrix**

wähle deren Minimum $\hat{\mathbf{w}}$ als $\mathbf{w}_{n+1}$

$$0 =! \nabla F(\hat{\mathbf{w}}) = \nabla F(\mathbf{w}_n) + \mathbf{H}(\mathbf{w}_n)(\hat{\mathbf{w}}-\mathbf{w}_n)$$

$$\Rightarrow \mathbf{w}_{n+1} = \hat{\mathbf{w}} = \mathbf{w}_n - \mathbf{H}^{-1}(\mathbf{w}_n)\nabla F(\mathbf{w}_n)$$



- **Bem:**

  -- $-\mathbf{H}^{-1}(\mathbf{w}_n)\nabla F(\mathbf{w}_n)$ **zeigt von** $\mathbf{w}_n$ **direkt zum Minimum der quadrat. Näherung**

- **Probleme:**

  -- H **bei hochdimensionalen Problemen aufwändig zu berechnen und zu invertieren**

  -- Nicht robust bei Übergang zu nichtkonvexen Optimierungsproblemen

Zurückführung auf eindimensionale Probleme: Linien-Suche

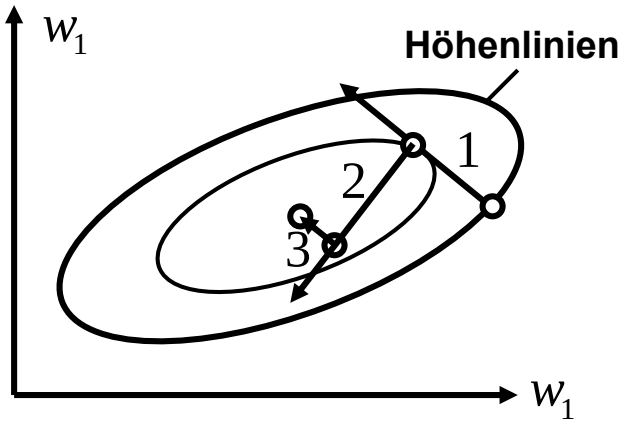
- **Gradientenbasierte Linien-Suche:**

-- Im n-ten Schritt wähle Richtung, z.B. $\mathbf{d}_n = -\nabla F(\mathbf{w}_n) = -\mathbf{g}_n$

-- Minimiere (wie gehabt) 1D-Funktion

$$f(\lambda) = F(\mathbf{w}_n + \lambda \mathbf{d}_n), \quad \hat{\lambda} = \arg\min f(\lambda)$$

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \hat{\lambda}\mathbf{d}_n$$



Bem:

-- Entspricht automatischer Wahl der Lernschrittweite

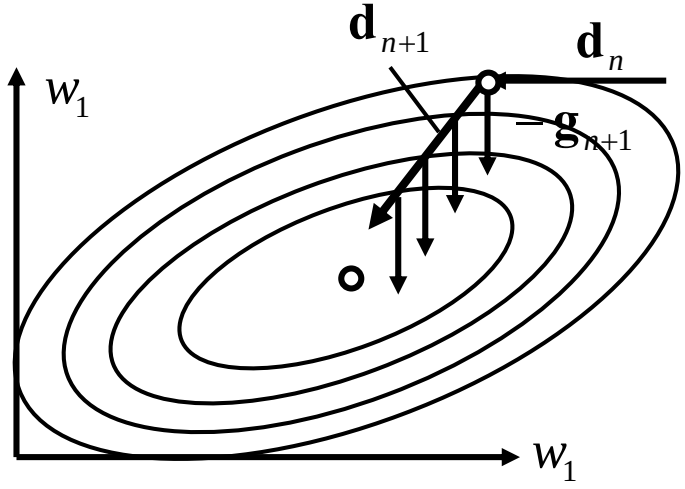
-- $\mathbf{d}_{n+1}{}^T \mathbf{d}_n \equiv \mathbf{g}_{n+1}^T \mathbf{g}_n = 0$

$$0 = \frac{d}{d\lambda} F(\mathbf{w}_n + \lambda \mathbf{d}_n)\Big|_{\hat{\lambda}} = \nabla F^T(\mathbf{w}_n + \hat{\lambda}\mathbf{d}_n)\mathbf{d}_n = \nabla F^T(\mathbf{w}_{n+1})\mathbf{d}_n = \mathbf{g}_{n+1}^T \mathbf{d}_n = -\mathbf{d}_{n+1}^T \mathbf{d}_n$$

Probleme:

-- „Zickzack-Kurs“, oft sogar bei quadratischer Fehlerfunktion

- **Konjugierte Gradientenverfahren:**



-- Gehe in „die gute Richtung“: $\mathbf{d}_{n+1} = -\mathbf{H}^{-1}\mathbf{g}_{n+1}$
aber ohne $\mathbf{H}^{-1}$ **explizit zu berechnen**

-> Äquivalent: Gehe in die Richtung, in die sich der Gradient (in 1. Näherung) nicht ändert. Denn:

(1) : Es gilt: $\mathbf{d}_{n+1}^T \mathbf{H} \mathbf{d}_n = 0$

$$0 = \frac{d}{d\lambda} F(\mathbf{w}_n + \lambda \mathbf{d}_n)_{\hat{\lambda}} = \nabla F(\mathbf{w}_n + \hat{\lambda}\mathbf{d}_n)^T \mathbf{d}_n = \nabla F(\mathbf{w}_{n+1})^T \mathbf{d}_n = \mathbf{g}_{n+1}^T \mathbf{d}_n = -\mathbf{d}_{n+1}^T \mathbf{H} \mathbf{d}_n$$

(2) : $\nabla F(\mathbf{w}_{n+1} + \varepsilon \mathbf{d}_{n+1})^T \mathbf{d}_n \approx \nabla F(\mathbf{w}_{n+1})^T \mathbf{d}_n + \varepsilon \mathbf{d}_{n+1}^T \mathbf{H} \mathbf{d}_n = \nabla F(\mathbf{w}_{n+1})^T \mathbf{d}_n$

-- Daraus läßt sich der Conjugate Gradient Algorithmus ableiten:

1. **Wähle Anfangs-Parameter** $\mathbf{w}_1$ **und** $\mathbf{d}_1 = -\mathbf{g}_1$
2. **im** n**-ten Schritt minimiere** $F(\mathbf{w}_n + \lambda \mathbf{d}_n), \quad \Rightarrow \quad \mathbf{w}_{n+1} = \mathbf{w}_n + \hat{\lambda}\mathbf{d}_n$
3. **Berechne** $\mathbf{g}_{n+1} = \nabla F(\mathbf{w}_{n+1})$
4. **Konjugierte Gradienten-Richtung:** $\mathbf{d}_{n+1} = -\mathbf{g}_{n+1} + \beta_n \mathbf{d}_n, \ \beta_n = \dfrac{\mathbf{g}_{n+1}^T (\mathbf{g}_{n+1} - \mathbf{g}_n)}{\mathbf{g}_n^T \mathbf{g}_n}$
5. **Falls Ergebnis verbesserungswürdig:** $n = n+1$**, gehe zu 2**

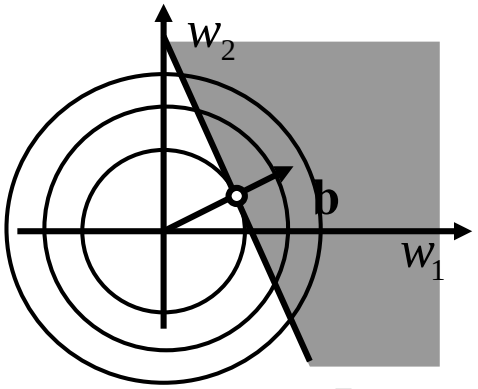
Optimierungsprobleme mit Randbedingungen

Def: Gleichheits-Randbedingung: **Minimiere** $F(\mathbf{w})$

unter den Bedingungen $G_j(\mathbf{w}) = 0, \quad j = 1, \ldots, l$

Def: Ungleichheits-Randbedingung: **Minimiere** $F(\mathbf{w})$

unter den Bedingungen $U_i(\mathbf{w}) \leq 0, \quad i = 1, \ldots, k$



$$\|\mathbf{w}\|^2 = \min, \quad geg.\ 1 - \mathbf{w}^T\mathbf{b} \leq 0$$

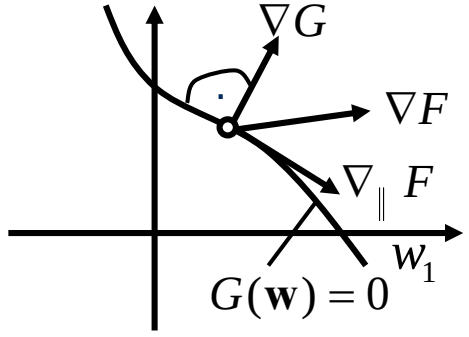
- **Lagrange-Multiplikatoren (Motivation):**

Ziel: Minimiere F**, aber bleibe auf der Hyperfläche** $G(\mathbf{w}) = 0$

-- Stelle sicher, nur entlang $\nabla_{\|} F$ **zu optimieren**

-- *grad* $(G(\mathbf{w}))$ **ist senkrecht auf die Fläche** $(G(\mathbf{w})=0)$**:**

Für $\boldsymbol{\varepsilon}$ **auf Fläche:** $G(\mathbf{w}+\boldsymbol{\varepsilon}) = G(\mathbf{w}) + \boldsymbol{\varepsilon}^T \nabla G =! G(\mathbf{w}) \Rightarrow \boldsymbol{\varepsilon} \perp \nabla G$



-- **Also:** $\nabla F_{\|}(\mathbf{w}) = \nabla F(\mathbf{w}) + \alpha \nabla G(\mathbf{w})$ **für geeignetes** α

-- **Ansatz:** $L(\mathbf{w}, \alpha) = F(\mathbf{w}) + \alpha G(\mathbf{w})$ **Löse:** $\dfrac{\partial L(\mathbf{w}, \alpha)}{\partial w_i} = 0, \ i = 1, \ldots, d$

-- Alternative Interpretation: An der Lösung sind $\nabla F(\mathbf{w}), \nabla G(\mathbf{w})$ **parallel**

$$\frac{\partial L(\mathbf{w}, \alpha)}{\partial \alpha} (= G(\mathbf{w})) = 0$$

-- α **heissen Lagrange-Multiplikatoren**

- **Lagrange-Multiplikatoren und Kuhn-Tucker Sattelpunkt-Bedingung:**

-- Betrachte Gleichheitsbedingung $G(\mathbf{w}) = 0$ **als Paar** $G(\mathbf{w}) \le 0, \quad -G(\mathbf{w}) \le 0$

-- Ab jetzt oBdA Ungleichheits-Randbedingungen

$F(\mathbf{w}) = \min, \quad \text{geg.} \;\; U_i(\mathbf{w}) \le 0, \quad i = 1, \ldots, k$

-- Betrachte Lagrange-Funktion $L(\mathbf{w}, \boldsymbol{\alpha}) = F(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i U_i(\mathbf{w}), \;\; \alpha_i \ge 0$

-- Wenn der Sattelpunkt $(\hat{\mathbf{w}}, \hat{\boldsymbol{\alpha}})$ **existiert, also für alle** $(\mathbf{w}, \boldsymbol{\alpha} \ge \mathbf{0})$

$L(\hat{\mathbf{w}}, \boldsymbol{\alpha}) \le L(\hat{\mathbf{w}}, \hat{\boldsymbol{\alpha}}) \le L(\mathbf{w}, \hat{\boldsymbol{\alpha}})$ **dann löst** $\hat{\mathbf{w}}$ **das Optimierungsproblem mit RB**

-- Beweis: (1) $L(\hat{\mathbf{w}}, \boldsymbol{\alpha}) \le L(\hat{\mathbf{w}}, \hat{\boldsymbol{\alpha}}) \Rightarrow \sum_{i=1}^{k} (\alpha_i - \hat{\alpha}_i) U_i(\hat{\mathbf{w}}) \le 0$ **für bel.** $\{\alpha_i \ge 0, i = 1, \cdots, k\}$

mit $\alpha_i = \hat{\alpha}_i + 1, \;\; \alpha_j = \hat{\alpha}_j, \;\; j \ne i \;\; \Rightarrow U_i(\hat{\mathbf{w}}) \le 0$ **Randbedingung erfüllt**

mit $\alpha_i = 0, \qquad \alpha_j = \hat{\alpha}_j, \;\; j \ne i \;\; \Rightarrow \hat{\alpha}_i U_i(\hat{\mathbf{w}}) \ge 0$

$\boxed{\Rightarrow \hat{\alpha}_i U_i(\hat{\mathbf{w}}) = 0}$ **Karush-Kuhn-Tucker (KKT)-Bedingung**

Nur entweder $\hat{\alpha}_i$ **oder** $U_i(\hat{\mathbf{w}})$ **können von Null abweichen**

(2) $L(\hat{\mathbf{w}}, \hat{\boldsymbol{\alpha}}) \le L(\mathbf{w}, \hat{\boldsymbol{\alpha}})$

$$\Rightarrow F(\hat{\mathbf{w}}) + \sum_{i=1}^{k} \hat{\alpha}_i U_i(\hat{\mathbf{w}}) = F(\hat{\mathbf{w}}) \le F(\mathbf{w}) + \sum_{i=1}^{k} \hat{\alpha}_i U_i(\mathbf{w}) \le F(\mathbf{w})$$

$$= 0 \qquad\qquad \le 0 \text{ wo RB erfüllt}$$

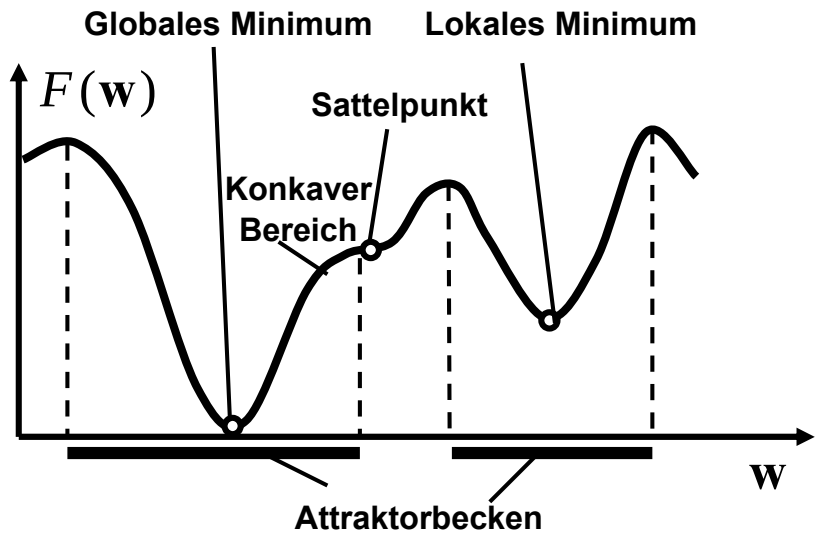
Nichtkonvexe Optimierungsprobleme

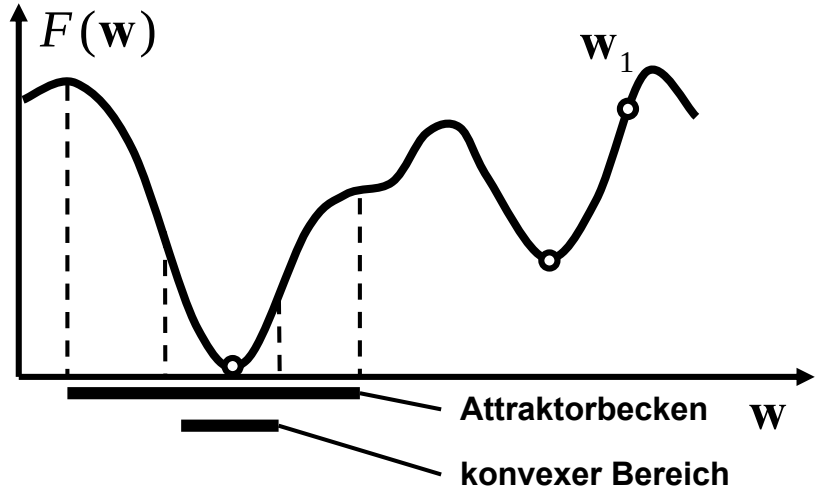
Mögliche Eigenschaften:

- **Konkave Bereiche**
- **Sattelpunkte**
- **Lokale Minima**
- **Jedes Minimum hat einen Einzugsbereich -> Attraktorbecken**
- **Nahe eines Minimums ist das Problem konvex**

Optimierung:

- **I. d. R. wird ein lokales Minimum gefunden**
- **Im Attraktorbecken: Gradientenabstieg, Liniensuche funktionieren**
- **Im konvexen Bereich: Newtonverfahren funktioniert**





- **<u>Möglichkeit:</u> Wiederholter Gradientenabstieg**

  -- Wähle Startparameter $\mathbf{w}_0$

  -- Finde Minimum im jeweiligen Attraktorbecken

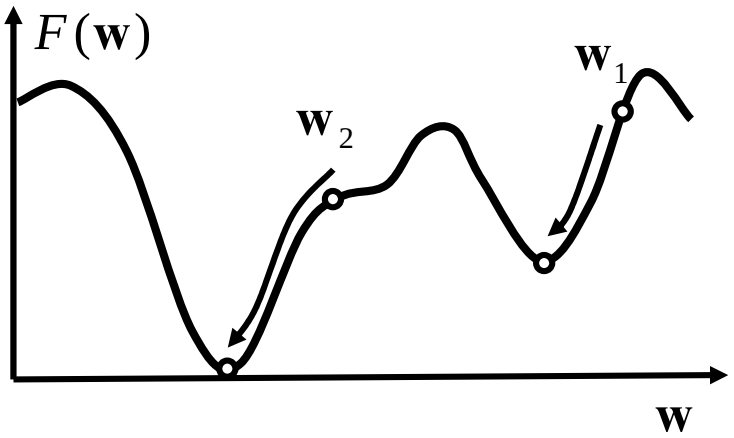
  -- Nach vielen Durchgängen wähle das Minimum mit kleinstem F

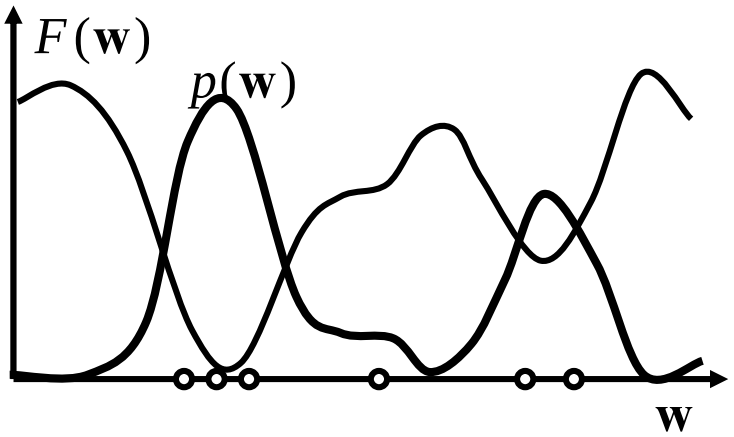
- **<u>Probleme:</u>**

  -- Keine Garantie, das globale Minimum zu finden

  -- Fluch der Dimensionen

  -- Kein Stopkriterium





Wichtiger: Globale Suchstrategien

<u>Monte Carlo Verfahren</u>

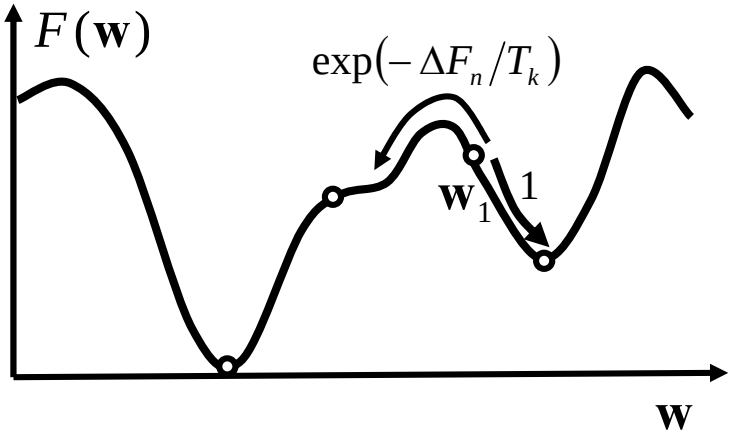
-- Versuch, Stichprobe $\{\mathbf{w}_0,\mathbf{w}_1,...\}$ aus der unbekannten Verteilung $p(w) = \exp(-F(\mathbf{w}))$ zu ziehen

-- Prinzip: Wähle zufällig $\mathbf{w}$, bewerte diese mit Hilfe von F, Akzeptiere proportional zu p

-- Beispiele: Importance sampling, Metropolis Algorithmus

-- Anwendung: Numerische Integration, Globale Optimierung

Simulated Annealing



- **Idee: Betreibe anfangs Importance Sampling, schließe dann System in tiefstes Minimum ein**
- **Prinzip: Lasse anfangs auch ungünstige Sprünge zu, um lokalen Minima zu entkommen**
- **Algorithmus**
  1. **Wähle hohe „Pseudotemperatur“** T_0
  2. **Wähle Zufalls-Sprung** $\Delta\mathbf{w}$**, berechne** $\Delta F_n = F(\mathbf{w}_n + \Delta\mathbf{w}) - F(\mathbf{w}_n)$
  3. $\Delta F_n < 0 \Rightarrow \mathbf{w}_{n+1} = \mathbf{w}_n + \Delta\mathbf{w}$
     $\Delta F_n \geq 0 \Rightarrow \mathbf{w}_{n+1} = \mathbf{w}_n + \Delta\mathbf{w}$ **mit Wahrscheinlichkeit** $p_n = \exp\left(-\frac{\Delta F_n}{T_k}\right)$
  4. **Gehe** N **mal zurück zu 2**
  5. **Erniedrige die Temparatur** $T_{k+1} < T_k$ **gehe zu 2, bis Enddtemparatur** T_K **erreicht**
- **Annealing Schedule gegeben durch** $T_0, T_K, N,$ **Abkühlverfahren**
- **Globale Minimierung garantiert durch (zu langsame) logarithmische Abkühlung**
- **In der Praxis z.B:** $T_0 \approx (F_{\max} - F_{\min})_{est}$, $T_{k+1} = \alpha T_k$, $0.9 \leq \alpha \leq 0.99$, $N = 100d$, $\exp(-|\Delta F| / T_K) << 1$

Perceptron, Kern-Trick und Support Vector Machine

- **Das Perceptron (Wiederholung)**
- **Kernel-Klassifikation**
- **Large-Margin Klassifikatoren**
- **Support Vector Machine**

Das Perceptron

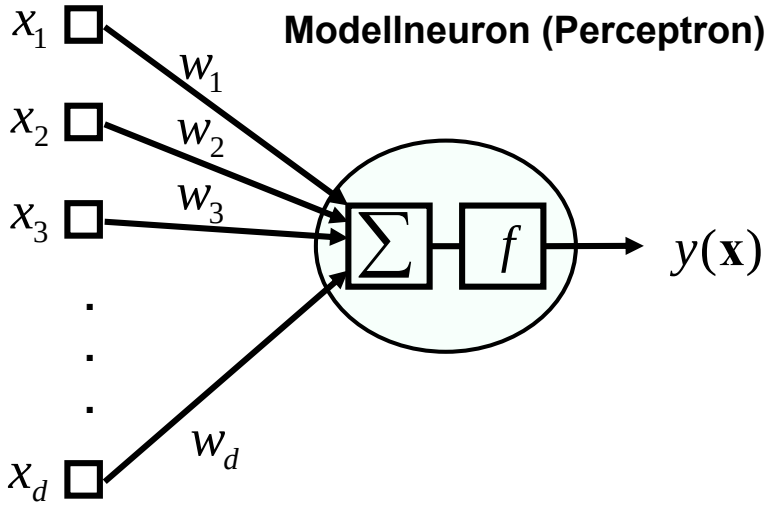
Binäre Klassifikation mit dem Perceptron

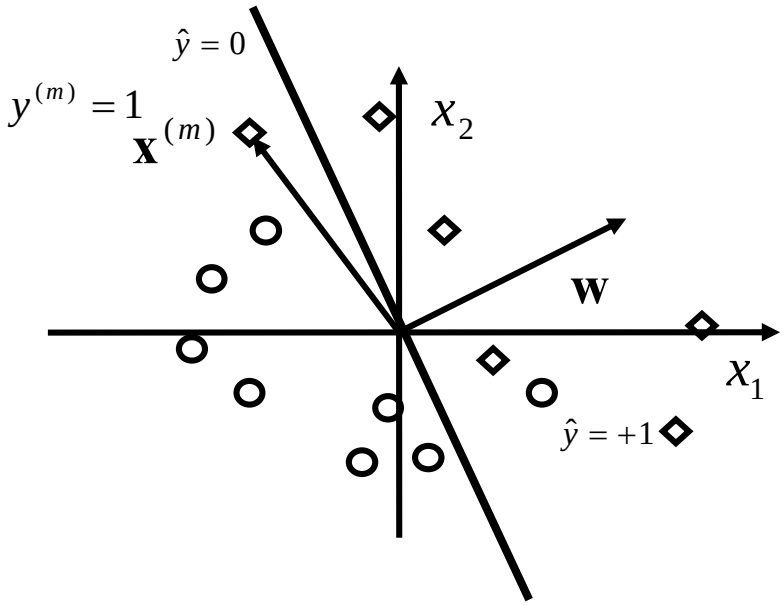
- **Neuronale Struktur:**
  Das binäre Modell-Neuron

$$y(\mathbf{x}) = \Theta(\mathbf{w}^T \mathbf{x} + b)$$

- **Wiederholung:**
  Das Perceptron kann als binärer Klassifikator dienen
  -- Geg: Daten in zwei Klassen, $y = 1, 0$
  -- Ges: Klassifikationsregel als Fkt. von $\mathbf{x}$
  -- Def: $x_{d+1} = 1,\ w_{d+1} = b \Rightarrow y = \Theta(\mathbf{w}^T \mathbf{x})$
- **Fehlerfunktion (Bsp): Gewichteter Falsch-Klassifikationsfehler**
- **Korrekte Klassifikation: Kein Fehler**
  Falsch-Klassifikation: positiver Fehler

$$F_P(w) = -\sum_{m=1}^{M} (y^{(m)} - \hat{y}(\mathbf{x}^{(m)}))\mathbf{w}^T \mathbf{x}^{(m)}$$





- **<u>Perceptron-Lernregel:</u> Gradientenabstieg**

$$\Delta \mathbf{w} = -\eta \nabla F_P(\mathbf{w}) = \eta \sum_{m=1}^{M} (y^{(m)} - \hat{y}(\mathbf{x}^{(m)}))\, \mathbf{x}^{(m)}$$ **(Batch-Modus)**

Online-Lernregel:

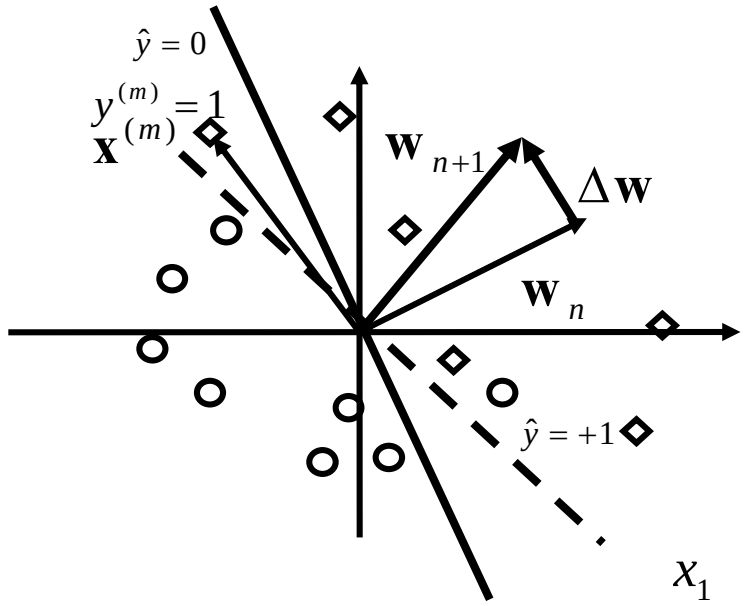
1. Wähle beliebiges $\mathbf{w}_0$

2. Wähle Muster m

3: $\mathbf{w}_{n+1} = \mathbf{w}_n + \eta(y - \hat{y}(\mathbf{x}^{(m)}))\, \mathbf{x}^{(m)}$

(d.h. tue nichts bei korrekter Klassifikation, biege $\mathbf{w}$ **bei Falsch-Klassifikation)**

4: Bis beste Klassifikation gehe zu 2



- **<u>Bem:</u>**

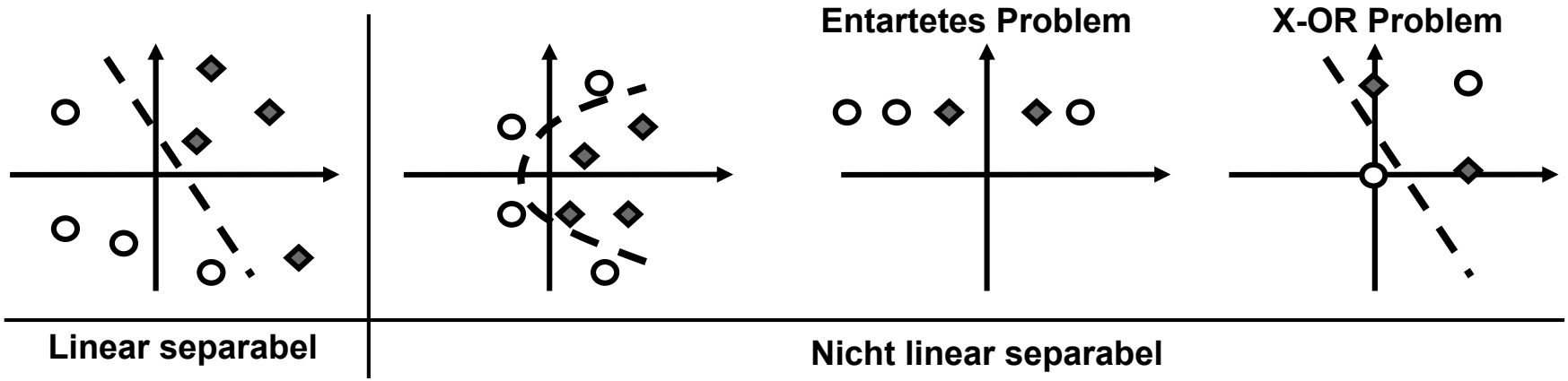
  -- Für linear separable Probleme <u>Konvergenzgarantie</u> in endl. vielen Schritten

  -- Funktioniert nur für linear separable Probleme

  -- Erweiterbar auf kontinuierliche Outputs

Lineare Separabilität:

Klassen können durch Hyperebene vollständig getrennt werden



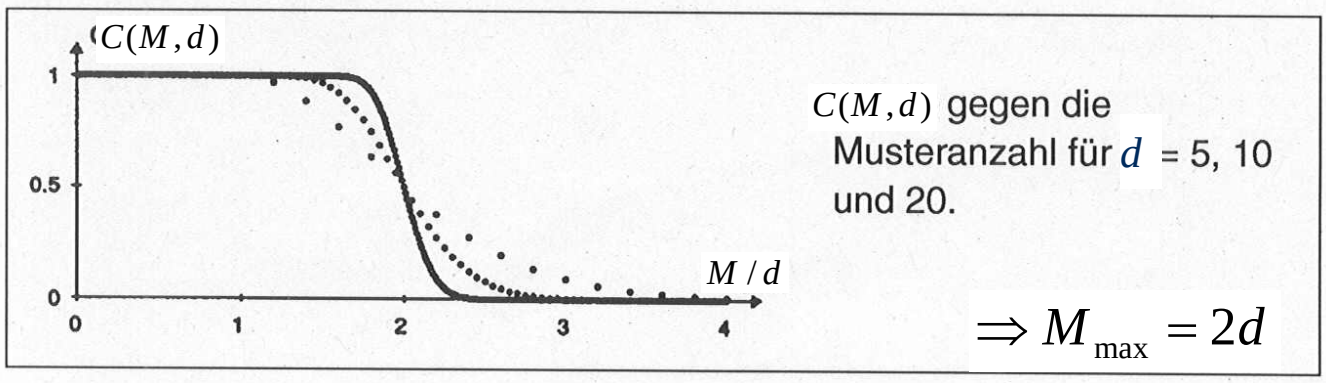
- **Wie wahrscheinlich sind M Muster in d Dimensionen linear separabel?**

  -- Wähle zufällig M d-dimensionale Muster

  -- Verteile zufällig Klassenlabels

  -- Bestimme Wa. für lin. Separabilität

  -- Erg. f. hohe Dimensionen:

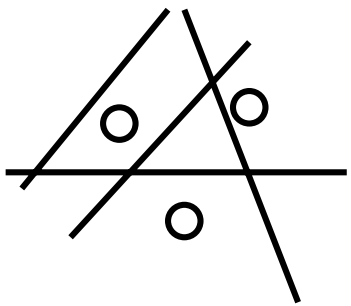


Lineare Separabilität und Modell-Komplexität

Lineare Klassifikation entspricht einer bestimmten Modellkomplexität
-- Wie komplex sind die Funktionen, die ein solches Modell implementieren kann?
--> „Zerschmettern“ von M Datenpunkten: Fähigkeit, alle 2^M möglichen Funktionen zu implementieren.

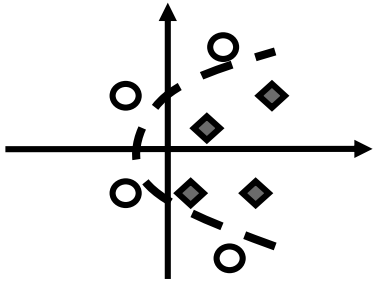
Def: Vapnik-Chervonenkis-Dimension (VC-Dimension) eines Modells:
Größtes M, das das Modell zerschmettern kann

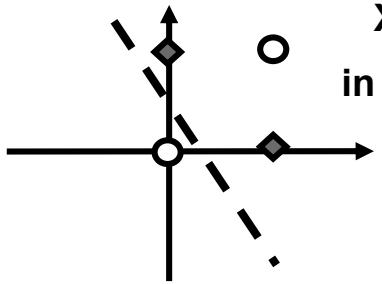
VC-Dimension e. linearen Klassifikators in d Dimensionen: $d+1$



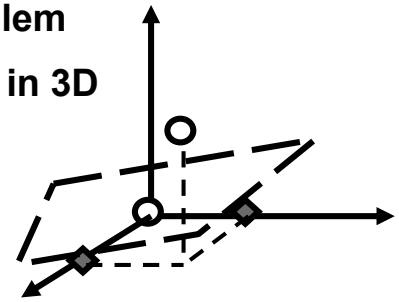
Klassifikation nicht linear separabler Probleme

Nichtlineare Klassifikation ... oder ... Lineare Klassifikation in höherer Dimension





X-OR Problem
in 2D...

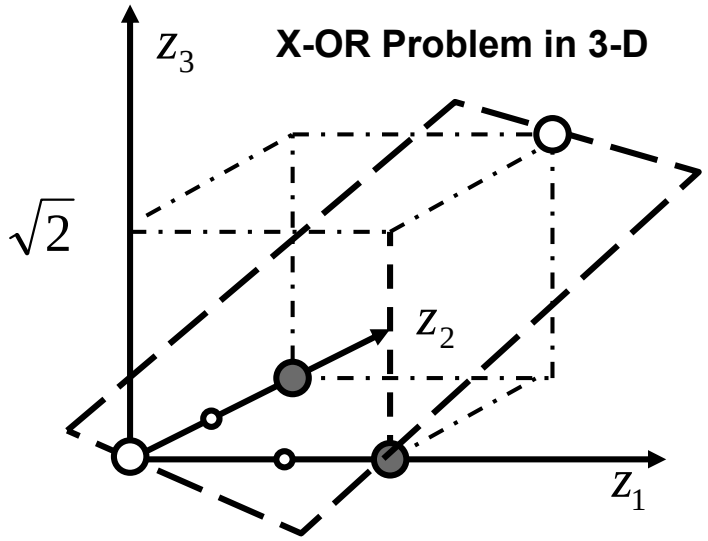


in 3D

Der Kern-Trick

Kernel-Klassifikation: Ein illustrierendes Beispiel

- **Projektion des X-OR Problems in einen 3-D Eigenschafts-Raum („feature space“).**
  Def: $\phi : \mathbf{x} \to \phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$
- **Erfolgreiche lineare Klassifikation im transformierten Raum: z.B.** $y = \Theta(\mathbf{v}^T\mathbf{z} + b)$
  mit $\mathbf{v} = (1,1,-\sqrt{2}), \quad b = -1/2$
- **Beob. 1: Separierende Hyperebene in 3-D entspricht nichtlinearer Klassifikation (hier: separierender Ellipse) in 2-D. Denn:**

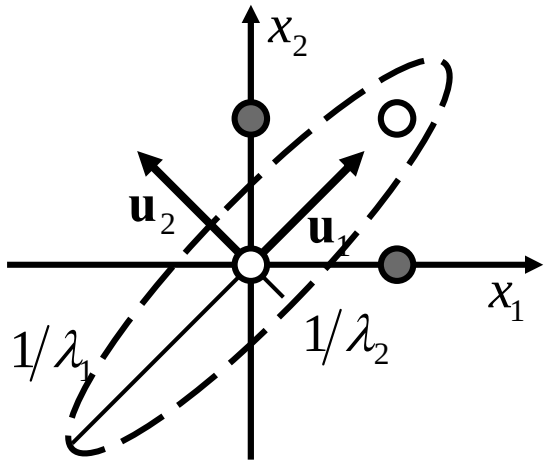


$$\mathbf{v}^T\mathbf{z} = v_1x_1^2 + v_2x_2^2 + \sqrt{2}v_3x_1x_2 = \mathbf{x}^T\begin{pmatrix} v_1 & v_3/\sqrt{2} \\ v_3/\sqrt{2} & v_2 \end{pmatrix}\mathbf{x} = \mathbf{x}^T\mathbf{V}\mathbf{x}$$

$\mathbf{V}^T = \mathbf{V} \Rightarrow$ **orthogonale Eigenvektoren** $\mathbf{V}\mathbf{u}_i = \lambda_i\mathbf{u}_i, \; i = 1,2$

$$\mathbf{x} = \sum_i \alpha_i\mathbf{u}_i \Rightarrow \mathbf{x}^T\mathbf{V}\mathbf{x} = const = \sum_{i,j}\alpha_i\alpha_j\mathbf{u}_i^T\mathbf{V}\mathbf{u}_j = \sum_i \lambda_i\alpha_i^2$$

$\sum_i \lambda_i\alpha_i^2 = const.$ **Ellipsengleichung mit Hauptachsen** $\mathbf{u}_1, \mathbf{u}_2$



- **Beob 2: Skalarprodukt im Feature-Raum läßt sich einfach im Originalraum (2-D) berechnen. Betrachte dazu die Abb:** $\mathbf{v} = \phi(\mathbf{w}), \quad \mathbf{z} = \phi(\mathbf{x})$

  Im Fall des obigen Beispiels :

  $\Rightarrow \mathbf{v}^T\mathbf{z} = \phi(\mathbf{w})^T\phi(\mathbf{x}) = w_1^2x_1^2 + w_2^2x_2^2 + 2w_1x_1w_2x_2 = (w_1x_1 + w_2x_2)^2 = (\mathbf{w}^T\mathbf{x})^2 \equiv k(\mathbf{w},\mathbf{x})$

  $k(\mathbf{x},\mathbf{x}')$ **heißt Kernel (Kern).** $\phi(\mathbf{x})$ **heißt Kernel-Abbildung**

- **Beob. 3: Die Klassifikation läßt sich allein durch Berechnung des Kernels durchführen**

  Kernel Trick:

  Aus jedem Algorithmus, der durch Skalarprodukte formuliert ist, läßt sich das SP durch einen positiv definiten Kernel ersetzen und so ein alternativer Algorithmus formulieren.

  Kernel-Klassifikation:

  - **Definiere einen Kernel** $k(\mathbf{x},\mathbf{x}')$
  - **Formuliere das Klassifikationsproblem unter Verwendung des Kernels:** $y = \Theta(k(\mathbf{w},\mathbf{x}) + b)$

Bem:

-- Hohe VC-Dim (Modellmächtigkeit) durch hohe Dimension von $\phi(\mathbf{x})$

-- Trotzdem effiziente Berechnung durch Kernel $\phi(\mathbf{x})^T\phi(\mathbf{x}') = k(\mathbf{x},\mathbf{x}')$

-- Für das X-OR Problem (s.o.): $\mathbf{v} =: \phi(\mathbf{w}), \quad \mathbf{w} = (1,-1), \quad b = -1/2, \quad y = \Theta((\mathbf{w}^T\mathbf{x})^2 + b)$

Beispiele für Kernel-Funktionen

- **Polynomialer Kernel**
  Kernel: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^l \quad , l = 2,3, \ldots$
  Kernel-Abbildung: $\phi(\mathbf{x}) = \sum_{j_1=1}^{d} \ldots \sum_{j_l=1}^{d} x_{j_1} \ldots x_{j_d}$ **Raum aller Produkte v. Grad** l
  Dimension d. Feature-Raums: $(d+l-1)!/(l!(d-1)!)$ **Bsp:** d=100, l =5**: Dimension =** $9.2\ 10^7$

- **Inhomogener Polynom-Kernel**
  Kernel: $k(\mathbf{x}, \mathbf{x}') = (1+\mathbf{x}^T \mathbf{x}')^l \quad , l = 2,3, \ldots$
  Kernel-Abbildung: **Raum aller Produkte vom Grad höchstens** l

- **Gauss-scher radiale Basisfunktionen-Kernel**
  Kernel: $k(\mathbf{x}, \mathbf{x}') = \exp(-(\mathbf{x}-\mathbf{x}')^T (\mathbf{x}-\mathbf{x}') / 2\sigma^2)$
  Dimension des Feature-Raums: Unendlich, wegen $\exp(x) = \sum_{j=0}^{\infty} x^j / j!$

- **Sigmoider Kernel**
  Kernel: $k(\mathbf{x}, \mathbf{x}') = \tanh(\kappa(\mathbf{x}^T \mathbf{x}') + \vartheta)$
  Dimension des Feature-Raums: Unendlich, wegen $\tanh(x) = \dfrac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$

Mercer-Theorem und Konstruktion von Kernel-Abbildungen

Frage: Geg. Kernel-Funktion k, wie kann ich den zugehörigen Kernel-Raum finden, d.h., Abbildung finden, in dem k als Skalarprodukt agiert?

- **Mercer-Theorem:**
  geg: Symmetrischer, positiv definiter Kernel, also
  - -- $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$
  - -- $\int k(\mathbf{x}, \mathbf{x}') f(\mathbf{x}) f(\mathbf{x}') d^d\mathbf{x}\ d^d\mathbf{x}' \geq 0$ **für alle integrierbaren Funktionen** $f(\mathbf{x})$

  Dann: k hat ein orthogonales Eigenspektrum sowie eine EW-Zerlegung

  $$\int k(\mathbf{x}, \mathbf{x}')\psi_j(\mathbf{x}')d^d\mathbf{x}' = \lambda_j\psi_j(\mathbf{x}),\ j = 0,...J,\ \lambda_j \geq 0$$ **(Exakt: $J = \infty$)**

  $$k(\mathbf{x}, \mathbf{x}') = \sum_{j=0}^{J} \lambda_j \psi_j(\mathbf{x})\psi_j(\mathbf{x}')$$

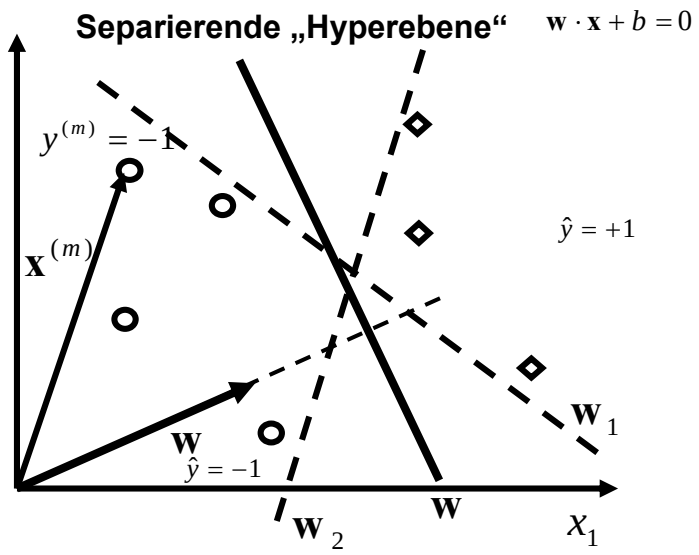
  Die zugehörige Kernel-Abbildung ist

  $$\phi(\mathbf{x}) = \sum_{j=0}^{J} \sqrt{\lambda_j}\psi_j(\mathbf{x})$$

- **Bem: Auch unendl. dim Probleme können durch endl. Summen angenähert werden**

„Large-Margin“-Klassifikatoren

Betrachte wieder linear separable Probleme



ObdA: Gehe über zu $y \in \{0,1\} \rightarrow y \in \{-1,1\}$

- **Beobachtung: Es existieren viele Lösungen (w,b) mit**

  $y^{(m)} = \mathrm{sgn}(\mathbf{w}^T \cdot \mathbf{x}^{(m)} + b)$, **also**

  $y^{(m)}(\mathbf{w}^T \cdot \mathbf{x}^{(m)} + b) > 0, \ \forall m = 1, \ldots M$

- **Idee: Wähle diejenige Lösung mit maximalem Abstand zu den Datenpunkten: „Large Margin“**

Warum?: Eindeutige Lösung, robusteste Lösung, kleinste VC Dimension (Generalisierung) effizient lösbar (quadratisches Programm), führt zu Support Vector Machine

- **Kanonische Form des Klassifikators**

Linear separables Problem: $\exists$ **Lösg** (w‘,*b*‘) $y^{(m)}(\mathbf{w}'^T \cdot \mathbf{x}^{(m)} + b') > 0, \Rightarrow \geq \mu > 0, \quad \forall m = 1, \ldots M$

also mit $\mathbf{w} := \frac{1}{\mu}\mathbf{w}', \ b := \frac{1}{\mu}b'$ $\boxed{\Rightarrow y^{(m)}(\mathbf{w}^T \cdot \mathbf{x}^{(m)} + b) \geq 1 \ \forall m = 1, \ldots M}$ **„=1“ => Margin**

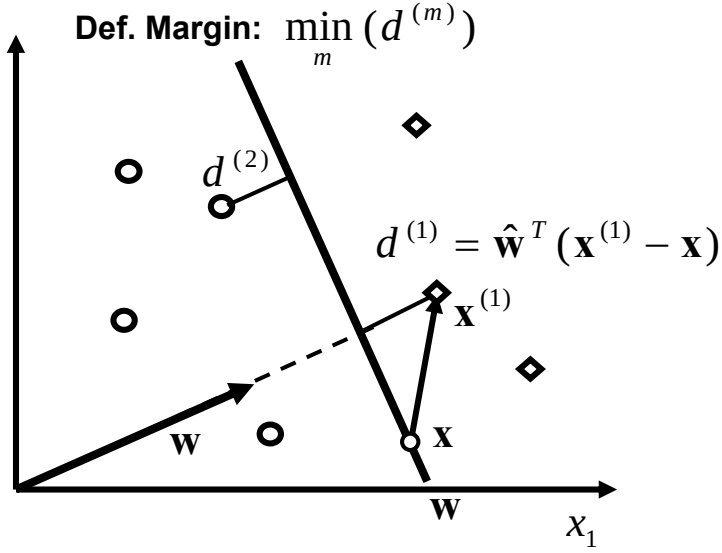
- **Margin eines Hyperebenen-Klassifikators:**

$$\mathrm{mg}(\mathbf{w}) = \min_m \min_{\mathbf{x}} \{ \| \mathbf{x}^{(m)} - \mathbf{x} \| \, \big| \, \mathbf{w}^T \mathbf{x} + b = 0 \}$$

In der kanonischen Form gilt:

$$\mathrm{mg}(\mathbf{w}) = \min_m \left| \frac{\mathbf{w}^T \mathbf{x}^{(m)} - \mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|} \right| = \min_m \left| \frac{\mathbf{w}^T \mathbf{x}^{(m)} + b}{\|\mathbf{w}\|} \right| =$$

$$= \min_m \frac{y^{(m)} (\mathbf{w}^T \mathbf{x}^{(m)} + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$



Intuition: Large-Margin Ebene durch die nächstliegenden Datenpunkte bestimmt

- **Large Margin:**

  -- Minimiere $\|\mathbf{w}\|^2$

  -- aber behalte kanonische Form bei: Also

$$\mathbf{w}^{LM} = \arg\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2$$ **unter den Randbedingungen**

$$\Rightarrow -(y^{(m)} (\mathbf{w}^T \cdot \mathbf{x}^{(m)} + b) - 1) \leq 0, \quad \forall m = 1, \ldots M$$

=> Optimierungsproblem mit Rdbed: Lagrange-Fkt

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{m=1}^{M} \alpha^{(m)} y^{(m)} (\mathbf{w}^T \cdot \mathbf{x}^{(m)} + b) - 1) =$$ **Sattelpunkt**

$$\alpha^{(m)} \geq 0$$

Support-Vektor-Machine

- **Lösung des Optimierungsproblems bedingt:**

-- $\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0 \Rightarrow \boxed{\mathbf{w} = \sum_{m=1}^{M} \alpha^{(m)} y^{(m)} \mathbf{x}^{(m)}}$ => $\mathbf{w}$ = **Linearkomb. der Trainingsdaten**

-- **KKT Bedingung:** $\alpha^{(m)}(y^{(m)}(\mathbf{w}^T \mathbf{x}^{(m)} + b) - 1) = 0, \quad m = 1, \ \ldots \ , M$

=> **Entweder** $\mathbf{x}^{(m)}$ **liegt am margin:** $y^{(m)}(\mathbf{w}^T \mathbf{x}^{(m)} + b) = 1, \ \Rightarrow \alpha^{(m)} > 0$ **erlaubt**

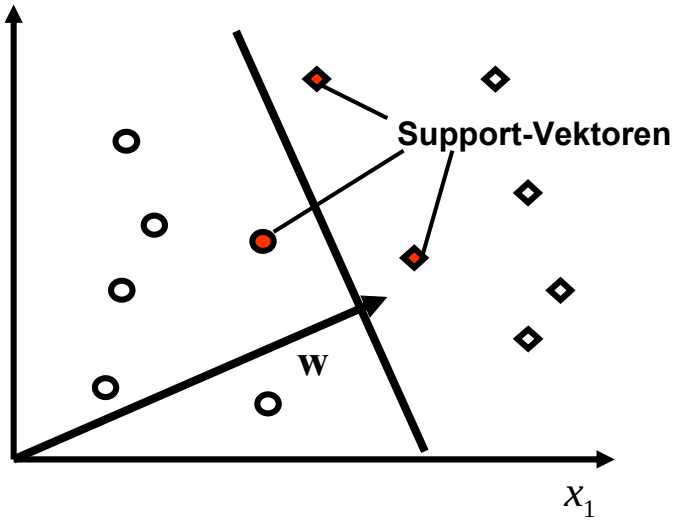
$\mathbf{x}^{(m)}$ **= Support-Vektor**

=> **Oder** $\alpha^{(m)} = 0$ => **Datenpunkt trägt nichts zum Parametervektor** $\mathbf{w}$ **bei**

-- Gesuchter Parametervektor ist eine
Linearkombination von Supportvektoren!
ist vollständig durch die Support-Vektoren bestimmt

-- $\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0 \Rightarrow \boxed{\sum_{m=1}^{M} \alpha^{(m)} y^{(m)} = 0}$

-- **Support-Vektoren liegen auf beiden Seiten der Ebene**



- **Duales Problem:**

-- Eliminiere $\mathbf{w}$ und b durch Gleichungen für α

$$\mathbf{w} = \sum_{m=1}^{M} \alpha^{(m)} y^{(m)} \mathbf{x}^{(m)} \qquad \sum_{m=1}^{M} \alpha^{(m)} y^{(m)} = 0$$

$$\frac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}\sum_{m,n=1}^{M} \alpha^{(m)}\alpha^{(n)} y^{(m)} y^{(n)} \mathbf{x}^{(m),T}\mathbf{x}^{(n)}$$

$$-\sum_{m=1}^{M} \alpha^{(m)}(y^{(m)}(\mathbf{w}^T\mathbf{x}^{(m)} + b) - 1) = \underbrace{-\sum_{m=1}^{M} \alpha^{(m)} y^{(m)} \mathbf{w}^T\mathbf{x}^{(m)}}_{= -\sum_{m,n=1}^{M} \alpha^{(m)}\alpha^{(n)} y^{(m)} y^{(n)} \mathbf{x}^{(m),T}\mathbf{x}^{(n)}} - b\underbrace{\sum_{m=1}^{M} \alpha^{(m)} y^{(m)}}_{=0} + \sum_{m=1}^{M} \alpha^{(m)}$$

$$\Rightarrow W(\boldsymbol{\alpha}) = \sum\nolimits_m \alpha^{(m)} - \frac{1}{2}\sum\nolimits_{m,n} \alpha^{(m)}\alpha^{(n)} y^{(m)} y^{(n)} \mathbf{x}^{(m)T}\mathbf{x}^{(n)} = \min$$

Mit RB: $\alpha^{(m)} \geq 0, m = 1,\ldots,M, \quad \sum_m \alpha^{(m)} y^{(m)} = 0$

Duales Problem

-- Löse „duales“ Optimierungsproblem bezüglich α : Quadratisches Programm

-- Die Entscheidungsfunktion wird zu

$$\hat{y}(\mathbf{x}) = \mathrm{sgn}(\mathbf{w}^T \cdot \mathbf{x} + b) \quad \rightarrow \quad \hat{y}(x) = \mathrm{sgn}(\sum\nolimits_{m=1}^{M} \alpha^{(m)} y^{(m)} \mathbf{x}^{(m)T}\mathbf{x} + b)$$

- **Support-Vector-Machine und Kernel-Trick :**

  -- Beobachtung: Das duale Problem läßt sich rein durch Skalarprodukte formulieren

  -- Kerneltrick anwendbar!!

  -- Bei nicht-separablen Problemen, löse:

$$W(\boldsymbol{\alpha}) = \sum_m \alpha^{(m)} - \frac{1}{2}\sum_{m,n} \alpha^{(m)}\alpha^{(n)} y^{(m)} y^{(n)} k(\mathbf{x}^{(m)}, \mathbf{x}^{(n)}) = \min$$

Duales Problem

Mit RB: $\alpha^{(m)} \geq 0, m = 1,...,M, \quad \sum_m \alpha^{(m)} y^{(m)} = 0$

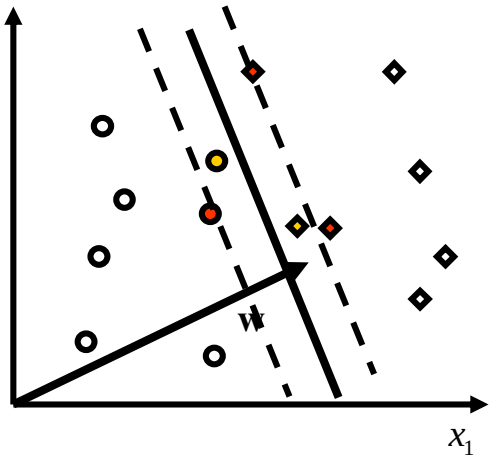
-- Die Entscheidungsfunktion wird zu $\hat{y}(x) = \mathrm{sgn}(\sum_{m=1}^{M} \alpha^{(m)} y^{(m)} k(\mathbf{x}^{(m)}, \mathbf{x}) + b)$

- **Behandlung verrauschter Probleme: Soft-Margin Klassifikatoren**

  -- Erlaube gelegentliche Verletzung des Margin

  -- Lerne „Lockerungsvariablen“ $\xi^{(m)} \geq 0$ **mit**

$$y^{(m)}(\mathbf{w}^T \mathbf{x}^{(m)} + b) \geq 1 - \xi^{(m)}, \quad \text{aber} \quad \sum_{m=1}^{M} \xi^{(m)} = \min$$



Bayesianische Netze (Bayes-Belief Netze)

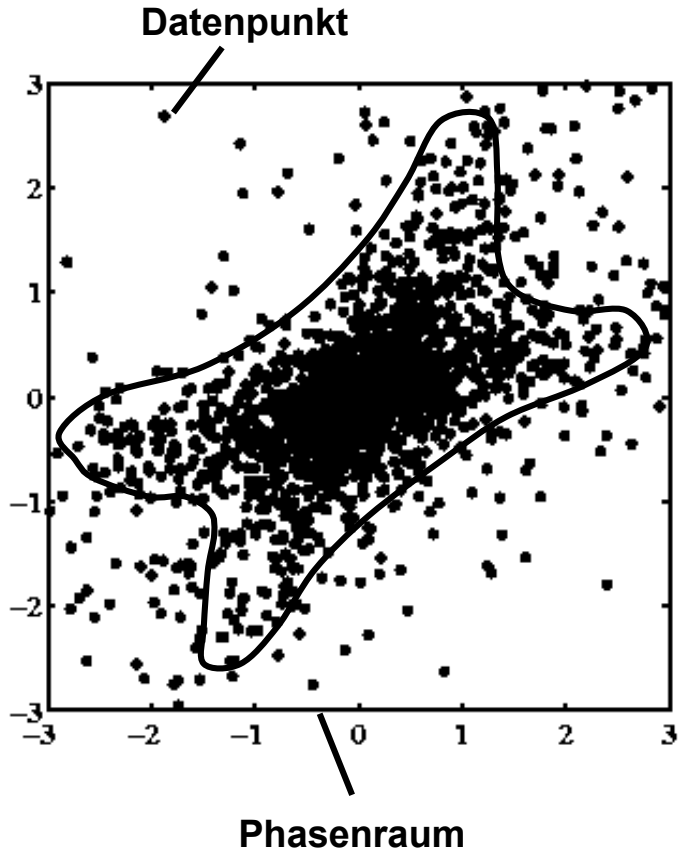
- **Bayes-Belief Netze und Dichteschätzung**
- **Gerichtete Graphen und Parameter**
- **Strukturlernen**

- **Folien: Mathäus Dejori**

Dichteschätzung

Bemerkungen

- **Schätzung der unterliegenden zusammengesetzten Wahrscheinlichkeitsdichte $p(\mathbf{x})$ aus den Daten**
- **Kenntnis der Dichte bedeutet vollständige statistische Charakterisierung!**
- **Charakterisierung der Struktur darin (z.B. Form Abhängigkeiten, Trends....)**



Aber:

- **Fluch der Dimensionen, komplexe Probleme ...**

Gesucht:

- **Einfachst mögliche Darstellung, die teilweise bzw. bedingte Unabhängigkeiten ausnützt**

Graphical Model

- **Combines probablity theory and graph theory**
- $p(\mathbf{X})$ **is defined in terms of an undirected or directed graph *G***
    - ***V*** **set of nodes (= components of random vector)**
    - ***E*** **set of edges (= dependencies between components of random vector)**
- **Probabilistic model**
- **Handles two problems in statistics:**
    - **uncertainty**
    - **complexity**

Bayesian Network

- **Consider decomposition of probability density functions:**

$$p(x_d, x_{d-1}, \ldots, x_2, x_1) =$$

$$= p_d(x_d \mid x_{d-1}, \ldots, x_1)\, p_{d-1}(x_{d-1} \mid x_{d-2}, \ldots, x_1) \ldots p_2(x_2 \mid x_1)\, p_1(x_1)$$

- **A Bayes-Net describes the underlying probablity distribution $p(\mathbf{X})$ as a set of conditional probabilities:**

$$p(\mathbf{x}) = P(x_1, .., x_d) = \prod_{i=1}^{d} P(x_i \mid Pa_i)$$

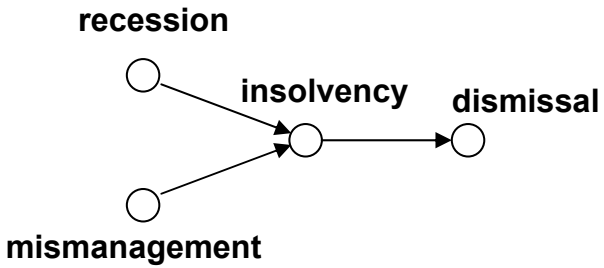
- **Each variable only depends on its parents!**
  Examples: Independent components, Markov chains

A Bayes-Net...

- **Belongs to the class of graphical models**
- **Consists of two parts: a graph structure *G* and a set of parameters Θ.**
- ***G* is a direct acyclic graph (DAG)**

Graph structure *G*

- ***G* is a direct acyclic graph (DAG)**
  - **edges are assigned with a direction**
  - ***G* contains no loops**
- **Each node encodes a variable**
- **An edge describes a conditional dependency between two variables (causality!!)**

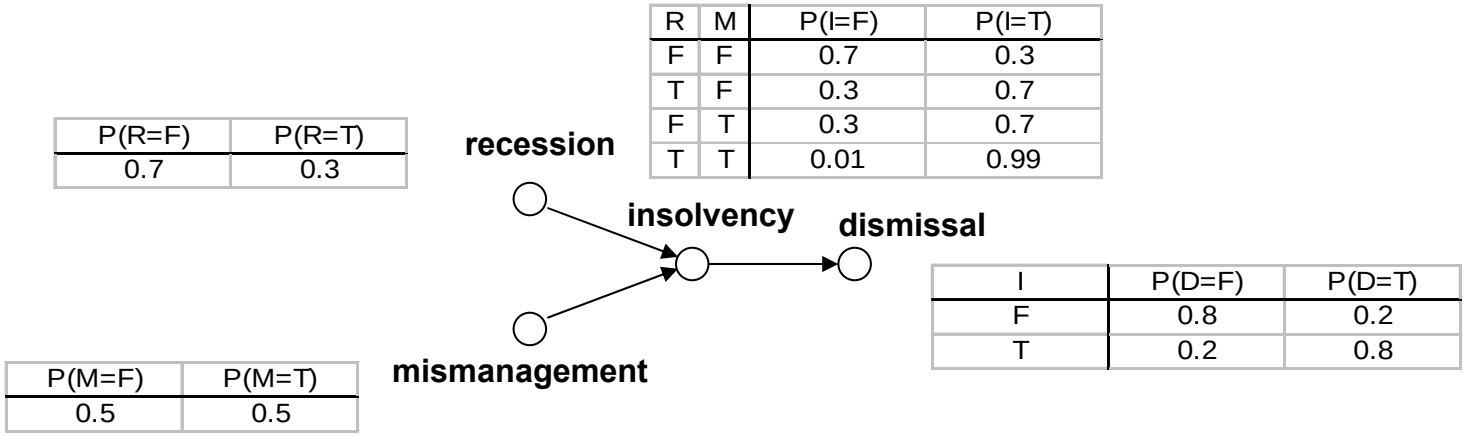


$$\mathbf{X} = \{recession, mismanagem\ ent, insolvency, dismissal\}$$

$$P(\mathbf{X}) = P(recession)P(mismanagem\ ent)P(insolvency \mid recession, mismanagem\ ent)P(dismissal \mid insolvency)$$

Parameters Θ

- **Θ encodes the conditional probability distribution (CPD) of each node**
- **Given a multinomial distribution, each CPD can be represented by a table**
- **Parameter Learning: Approximate table entries by relative frequencies**



P(R=F)	P(R=T)
0.7	0.3

R	M	P(I=F)	P(I=T)
F	F	0.7	0.3
T	F	0.3	0.7
F	T	0.3	0.7
T	T	0.01	0.99

I	P(D=F)	P(D=T)
F	0.8	0.2
T	0.2	0.8

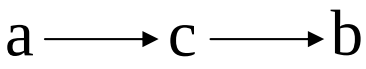
P(M=F)	P(M=T)
0.5	0.5

- **Belief Propagation: Calculation of posterior probabilities given some evidence**

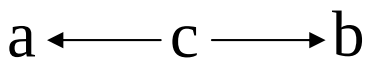
D-separation

Two variables *a* and *b* are d-separated if for all paths between *a* and *b* there is an intermediate variable *c* such that either

- ***c* is a node of a serial or divergence connection and its state is known**

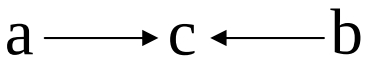


$$a \perp b \mid c$$



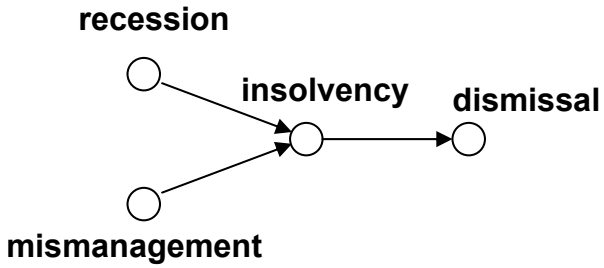
$$a \perp b \mid c$$

- ***c* is a node of converging connection, called *collider*, and neither c nor any other of its descendants is known**



$$a \perp b \mid \varnothing$$

D-separation (example)



- **Given insolvency, dismissal does not depend on recession anymore**

$$P(dismissal \mid insolvency, recession) = P(dismissal \mid insolvency)$$

- **Recession and mismanagement are d-separated given insolvency is unknown**

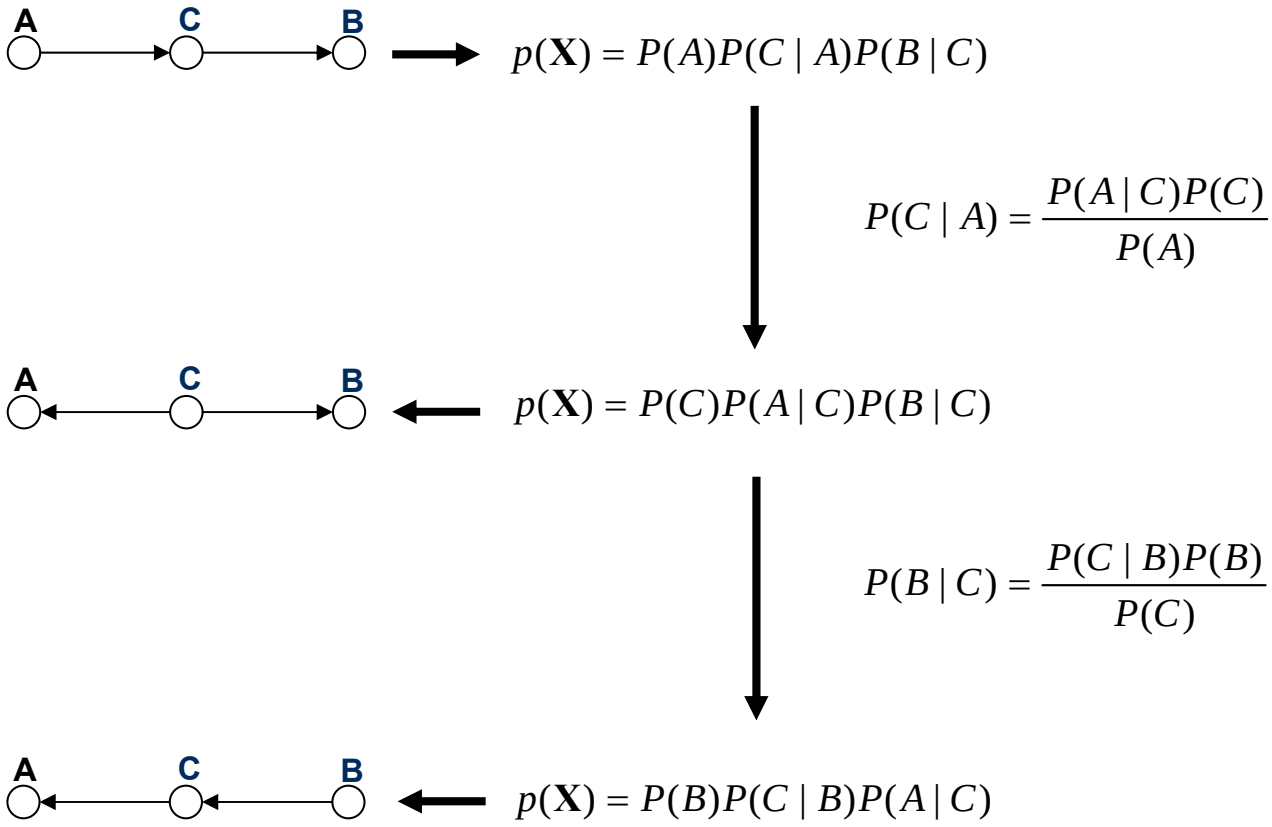
$$P(recession, mismanagement) = P(recession)P(mismanagement)$$

- ***Explaining away*: Knowledge about insolvency and recession influence our belief about mismanagement!**

$$P(mismanagement = T \mid recession = T, insolvency = T) \neq P(mismanagement = T \mid insolvency = T)$$

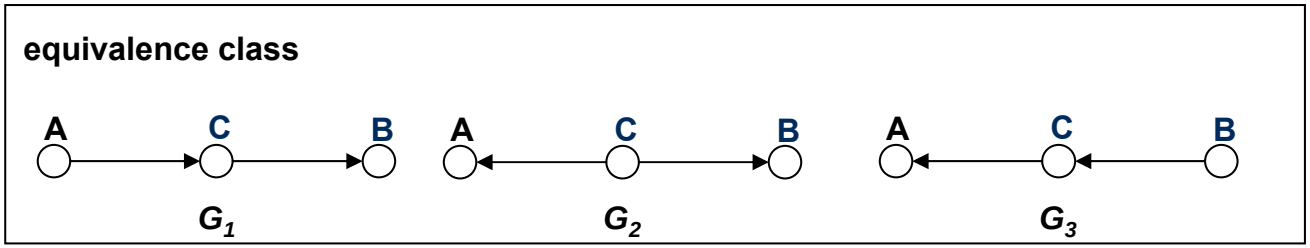
Structure-equivalence

• **Problem with edge-directions: The joint probability represented by a graph structure can equally be represented by another one.**



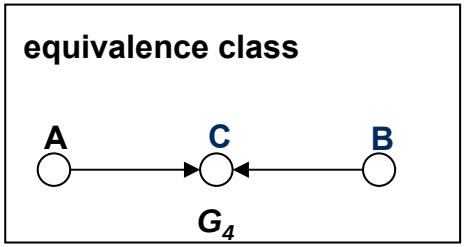
Structure-equivalence (condt)

DAGs belong to the same equivalence class if they have the same skeleton and the same set of colliders



$$A \perp B \mid C$$

Edges cannot be interpreted as an association between a cause and a consequence

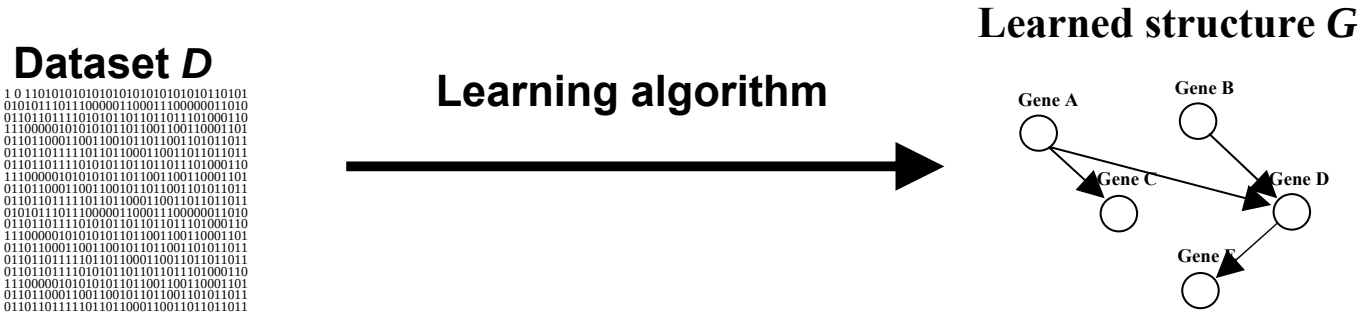


$$A \perp B \mid \varnothing$$

Edges can be interpreted as causal relationships

Structure learning task

- **Given a training set *D* of N independent instances of $X = \{ x^1,....,x^N \}$, find a network *G* that best matches *D.***



Scoring functions

• To evaluate the goodness of fit of a network with respect to the dataset, a statistically motivated *scoring function* S assigns a score S(G) to the graph G.

• Goal: Find the structure with the best score S(G|D), given the dataset D.

Frequentist way

Maximize the likelihood of the data:

$$S(G) = P(D \mid G, \Theta^{ML})$$
$$= \prod_{l=1}^{N} p(x^l \mid G, \Theta^{ML})$$

Bayesian score

S(G) is proportional to the posterior probability of a network structure given the data:

$$S(G) = P(G \mid D) = \frac{P(D \mid G)P(G)}{P(D)}$$

$$P(D \mid G) = \int P(D \mid \Theta, G)P(\Theta \mid G)d\Theta$$
$$= \prod_{l=1}^{N} \int p(x^l \mid D_l, G, \Theta)\, p(\Theta \mid D_l, G)d\Theta$$

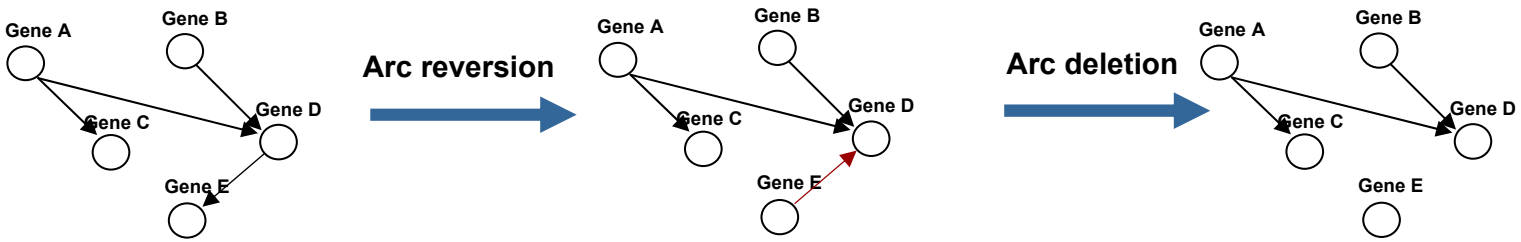
Local search strategy

- **Score can be factored into a product of functions that depend only on a node and ist parents (see slide #4)**

$$S(G \mid D) = \prod_i S_{local}(X_i, Pa_i \mid D)$$

- **Change one arc at each move and evaluate the gains made by this change**

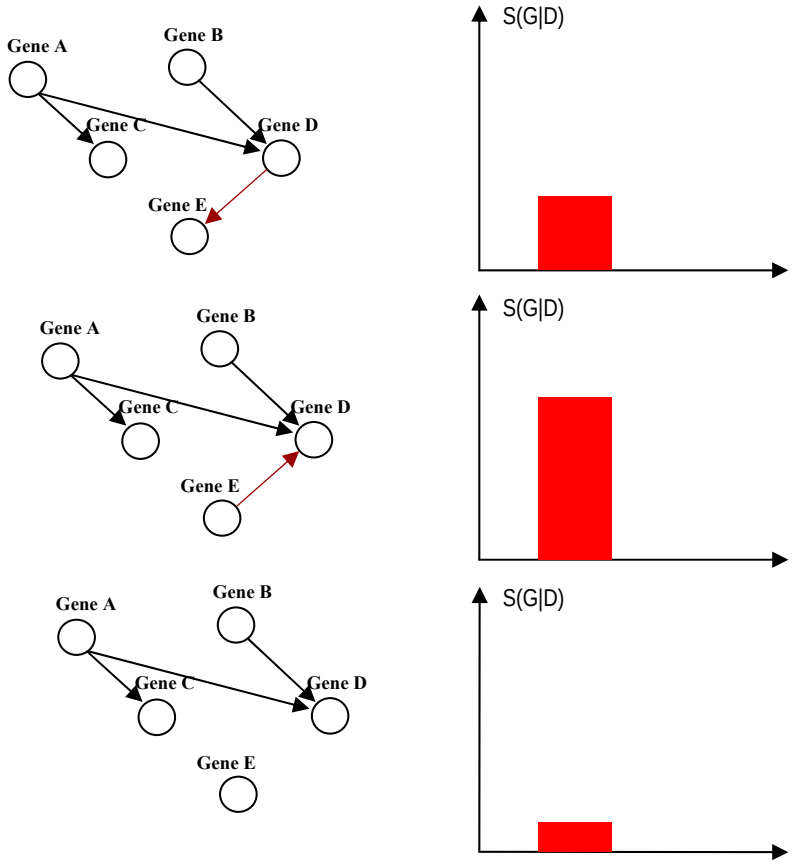
Initial structure G **neighboring structures G`**



If an arc to X_i is added or deleted, only Score(X_i| Pa_i) needs to be evaluated.
If an arc is reversed, only *Score(X_i|Pa_i)* and *Score(X_j|Pa_j)* need to be evaluated.

Searching the best local structure

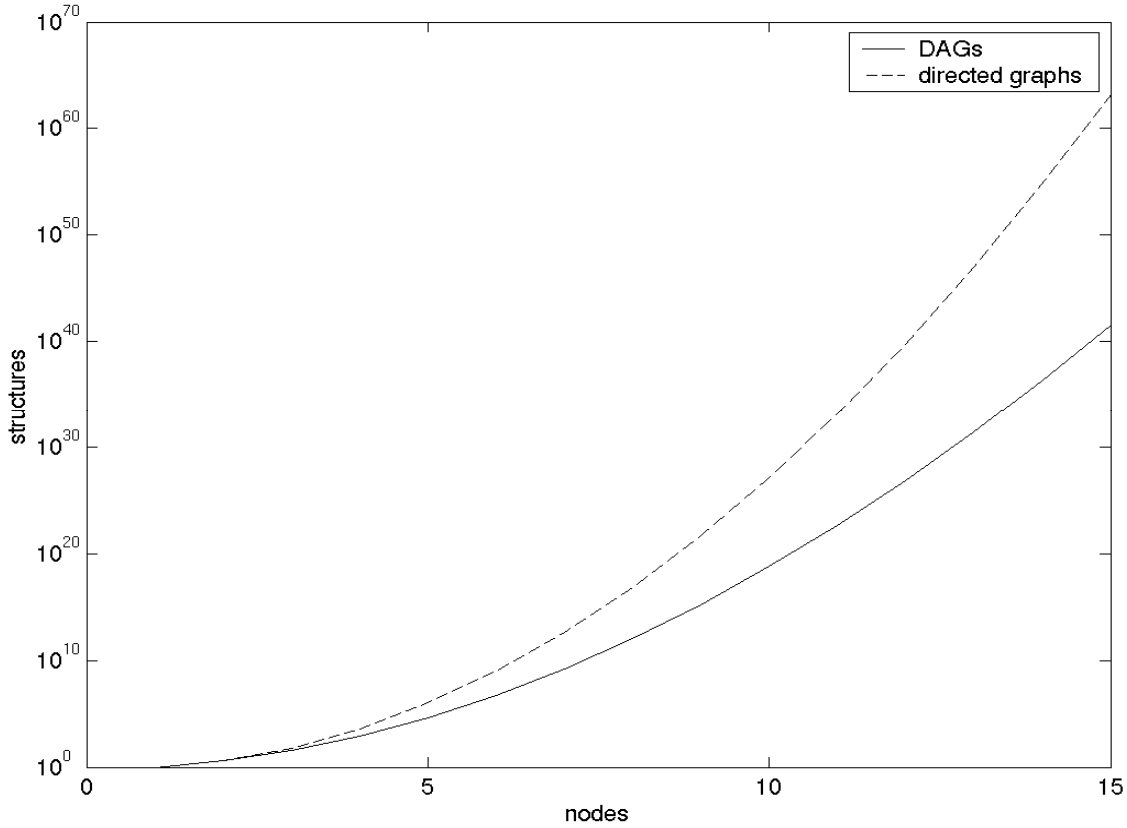
Find the structure with the best score S(G|D) by searching through the space of neighbooring structures



Local structure with the best score!

Search space

The number of possible structures grows super-exponentially with the number of variables:



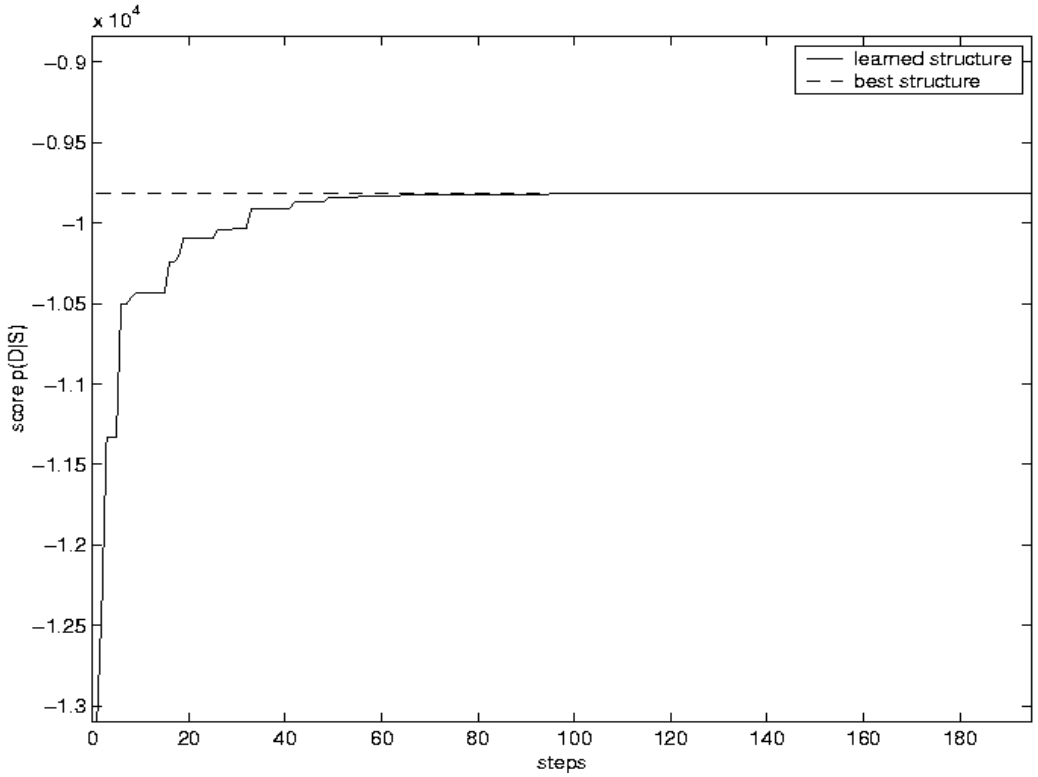
The problem of finding the best model is known to be NP-hard, so we have to use heuristic methods

Greedy Hill-climbing

Accept a change only if it increases the scoring function

Greedy hill climbing

Choose G somehow
While not converged
For each G' in nbd(G)
Compute Score(G')
$G^* = argmax_{G'} Score(G')$
If $Score(G^*) > Score(G)$
$G := G^*$
else converged=true



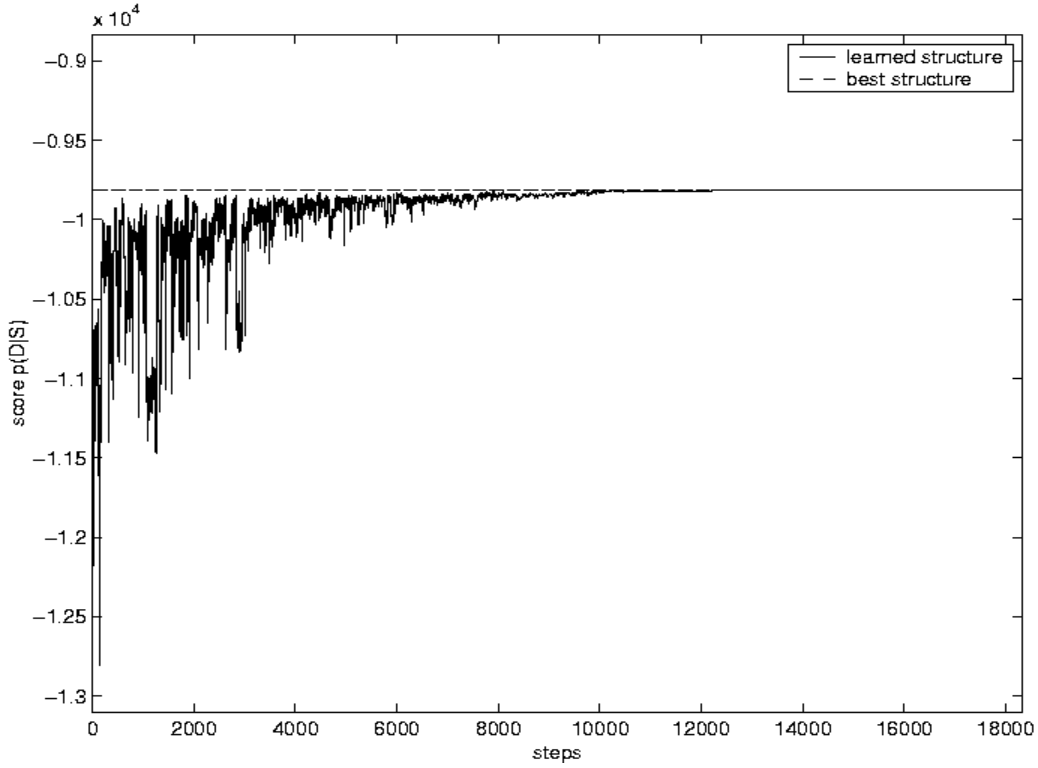
Problem: Optimization procedure can get stuck at local optima

Simulated Annealing (SA)

Changes improving the score are always accepted.
Changes decreasing the score are accepted or rejected with a finite probability

Simulated annealing

Initialize T
Choose G somehow
While T > T_{min}
For each G' in nbd(G)
Compute Score(G')
$G^* = argmax_{G'}$ Score(G')
calculate Δ := Score(G^*)-Score(G)
If Δ >0 or with p=exp(Δ/T)
G := G^*
reduce T (e.g T=0.9*T)



SA is not guaranteed to reach global optimum, since computational limits

Summary

- **A Bayesian network is a graphical representation of probability distributions**
- **It provides a compact and intuitive representation**
- **Learn the structure out of the data**
    - **Discover structural propertys of the domain**
    - **More information than simply testing for correlation**
- **Handle prior knowledge to guide learning procedure**
- **Edges can be interpreted in a causal way**

References

- **S. Lauritzen. "Graphical Models", Oxford. 1996.**

  The definitive mathematical exposition of the theory of graphical models

- **J. Whittaker. "Graphical Models in Applied Multivariate Statistics", Wiley. 1990.**

- **Kevin Murphy´s tutorial. 1998.**

  A Brief Introduction to Graphical Models and Bayesian Networks

  http://www.ai.mit.edu/~murphyk/Bayes/bayes.html