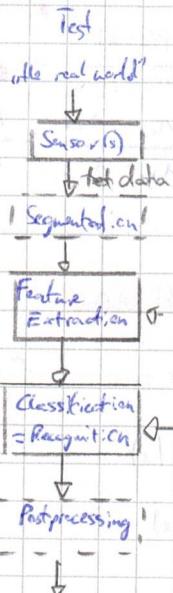


# Zusammenfassung Mustererkennung Kapitel 1: Introduction

Was ist ein Muster? Anordnung von Formen, Teilen oder Elementen oder etwas, das eine Klasse oder Typ repräsentiert

Typische Schritte der Mustererkennung:



Feature Extraction:

Finde eine niedrig-dimensionale Beschreibung eines Musters, die unabhängig von Größe, Rotation, Translational und robust gegenüber Rauschen ist

Merkmalvektoren sollten innerhalb einer Klasse nah beieinander sein und zwischen Klassen weit auseinander liegen

Kapitel 2: Bayesian Decision Theory

$$\text{Bayes: } P(s=i|x) = \frac{p(x|s=i) P(s=i)}{p(x)} \Rightarrow \text{Posterior} = \frac{\text{Likelihood} \cdot \text{Prior}}{\text{Evidence}}$$

$$\text{Max-prior-Erkennung: } s^* = \underset{i \in S}{\operatorname{argmax}} [P(s=i)]$$

$$\text{Max-Likelihood-Erkennung: } s^* = \underset{i \in S}{\operatorname{argmax}} [p(x|s=i)]$$

$$\text{Maximum A Posteriori (MAP) Decision / Bayes decision rule: } s^* = \underset{i \in S}{\operatorname{argmax}} P(s=i|x)$$

$$\text{Bayes Fehler: } P^* = P(\text{error}|x) = 1 - \underset{i \in S}{\operatorname{max}} [P(s=i|x)] \rightarrow \underline{\text{M. Fehler Fehler ist minimal}}$$

$$\text{Winfelde: } P(\text{error}) = 1 - \frac{1}{N}$$

$$\text{Prior: } P(\text{prior}) = 1 - \underset{i \in S}{\operatorname{max}} [P(s=i)] \rightarrow \text{immer mindestens so gut wie Winfelde, maximal so gut wie Bayes}$$

Risiko: Aktionen haben Kosten, die in Entscheidung mit einzbezogen werden sollten

$$R(a=j|x) = \sum_{i \in S} \lambda(a=j|s=i) P(s=i|x) \quad \lambda(a=j|s=i): \text{Loss function. Wicke ist Aktion } a=j, \text{ wenn } s \text{ in Wahrheit } i \text{ ist.}$$

$$\text{Bayes decision rule for actions: } a^* = \underset{j \in A}{\operatorname{argmin}} [R(a=j|x)]$$

## 2-Klassen-Probleme

Likelihood-Ratio:  $LR(x) = \frac{p(x|s=1)}{p(x|s=0)} \rightarrow \Theta \Rightarrow \text{Decide for } s=1, \text{ else } s=0$

Möglichkeiten für  $\Theta$ :  $\Theta = 1 \rightarrow \text{Maximum Likelihood Entscheidung (ML)}$

$$\Theta = \frac{p(s=1)}{p(s=0)} \rightarrow \text{MAP}$$

$$\Theta = \frac{\lambda_{11} - \lambda_{21}}{\lambda_{21} - \lambda_{11}} \cdot \frac{P(s=1)}{P(s=0)} \rightarrow \text{Bayes risk decision}$$

## Discriminant Functions / Trennfunktionen

Bayes Klassifikator mit Trennfunktionen:

$$s^* = \arg \max_{i \in S} [g_i(x)] \quad a^* = \arg \max_{i \in A} [g_i(x)]$$

Allgemein:  $g_i(x) = -R(a=i|x)$

Minimierer Fehler Fall:  $g_i(x) = P(s=i|x)$

Beispiele für  $g_i(x)$ :

$$g_i(x) = p(x|s=i) \cdot P(s=i) \quad g_i(x) = \log p(x|s=i) + \log P(s=i)$$

Mögliche Operationen:  
ohne Klassifikation  
zu verändern

$$g_i(x) \rightarrow \alpha g_i(x), \alpha > 0$$

$$g_i(x) \rightarrow \alpha + g_i(x), \alpha \in \mathbb{R}$$

$$g_i(x) \rightarrow f(g_i(x)) \quad f \text{ linear steigende Funktion}$$

### 2-Klassen-Fall:

typischerweise eine einzelne Trennfunktion:  $g(x) = g_1(x) - g_2(x)$

$$\rightarrow s^* = 1 \text{ if } g(x) > 0 \quad s^* = 2 \text{ if } g(x) < 0$$

$$g(x) = P(s=1|x) - P(s=0|x) \approx \log \frac{P(s=1|x)}{P(s=0|x)} = LLR(x) + \log \frac{P(s=1)}{P(s=0)}$$

### Decision boundary / Entscheidungsgrenze:

$$g(x) = 0$$

$$-\frac{1}{2} \left( \frac{x-\mu}{\sigma} \right)^2$$

(log-)likelihood-ratio

### Univariater Gauß:

$$p(x) = N(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$-\frac{1}{2} \frac{(x-\mu)^T (x-\mu)}{\sigma^2}$$

### Multivariater Gauß:

$$p(x) = N(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \cdot e^{-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu)}$$

### Gaussisches Mixture Modell (GMM)

$$p(x) = \sum_{k=1}^{\infty} c_k \cdot N(x; \mu_k, \Sigma_k) \approx \sum_{k=1}^K c_k \cdot N(x; \mu_k, \Sigma_k) \quad \sum_{k=1}^K c_k = 1$$

### Missing Features

Hinzunehmen über gute Features

### Noisy Features

Definition: „channel model“  $p(x_{62}|x_{92})$

### (Bayesian) Belief Networks

Möglichkeit, expert (prior) knowledge in Entscheidungsprozess einzubringen

→ Statt Graphstrukturen, deren  $\mu$  und  $\Sigma$  wir oft nicht kennen, Graph, dessen Verbindungen wir kennen

Binary classification ( $N=2$ ):

True Positive Rate (Hit Rate):  $TPR = \frac{TP}{TP+FN} = \text{Recall}$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Accuracy}: ACC = \frac{TP+TN}{TP+FP+TN+FN}$$

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

false acceptance rate

$$\text{Error Rate}: ER = 1 - ACC$$

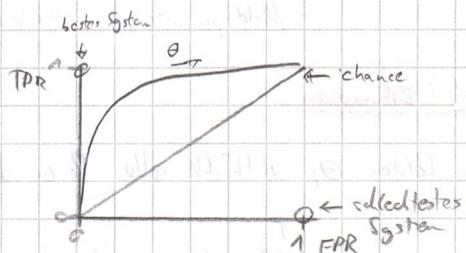
Erinnerung:  $g(x) = LLR(x) + \log \frac{P(s=1)}{P(s=0)}$   $\rightarrow ER, TPR, FPR, FNR, \dots$  alle von  $\theta$  abhängig

$$\theta_{\text{Bayes}} \triangleq \text{Minimum von } ER(\theta)$$

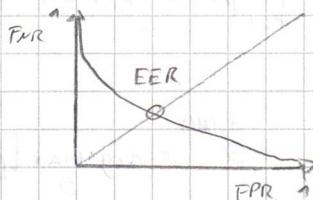
$$ER = FNR \cdot P(s=1) + FAR \cdot P(s=0)$$

Receiver Operating Characteristic (ROC)

plottet TPR als Funktion von FPR( $\theta$ ),



oder FNR als Funktion von FPR( $\theta$ )

Detection Error Tradeoff

plottet ebenfalls  $FNR=f(FPR)$ , aber mit transformierten Achsen, sodass Geradienteilung zu gerader Linie führt

Precision / Recall - Plot

$$\text{Precision} = f(\text{Recall}(\theta))$$

$\rightarrow$  Beste Precision ergibt seltenste Recall und andersherum

F-measure:

$$F_1 = 2 \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$g(x) - \log \frac{P(s=1)}{P(s=0)} = LLR(x) < \theta$$

$$LLR(x) = \log \left( \frac{P(x|s=1)}{P(x|s=0)} \right)$$

Training eines Classifiers

Training eines Bayes-Classifiers: Schätzen der Priors und Likelihoods

Supervised: Trainingsmaterial liegt mit Labels vor  $\rightarrow$  folgende Sache erfordert alle supervised

Unsupervised: Trainingsdaten liegen ohne Label vor

Prior-Schätzung: entweder einfach oder weggelassen

Likelihood-Schätzung: - Parametrisch (wissen Form der Likelihood, z.B. Gauß)

- Nicht-parametrisch  $\rightarrow$  wir machen keine Annahme über pdf-Form

Parametric Estimation

Parameter-Mödell  $\Theta_i$ : enthält alle R zum schätzenden Parameter z.B.  $p(x|\theta_i) \sim N(x; \mu_i, \Sigma_i)$

$$p(x = o_i | \Theta_i) = \prod_{t=1}^T p(x = o_{i,t} | \Theta_i) \quad \Rightarrow \Theta = \{\mu_i, \Sigma_i\}$$

Wie gut modelliert PDF mit  $\Theta_i$  die Trainingsdaten  $O_i$ .

Maximum Likelihood:  $\hat{\Theta}^{(ML)} = \arg \max_{\Theta} [p(\Theta | \Theta)]$

$$\hat{\Theta}^{(ML)} = \arg \max_{\Theta} [LL(\Theta)] \quad \text{mit Log-Likelihood } LL(\Theta) = \sum_{t=1}^T \log(p(x=o_{i,t} | \Theta))$$

Maximum Likelihood Parameter Estimation

Vorgehensweise: 1.) Entscheide für ein Modell, z.B. Multivariate Gauß

2.) Besorge Trainingsmaterial

3.) Suche über Parameterraum, durch Sache des maximalen LL.

$\hookrightarrow$  z.B. durch partielle Ableitung und Nullsetzen

Beispiel Multivariate Gaußfunktion als PDF:

$$\hat{\mu} = \underbrace{\frac{1}{T} \sum_{t=1}^T o_t}_{\text{Sample mean}} \quad \hat{\Sigma} = \frac{1}{T-1} \sum_{t=1}^T \underbrace{(o_t - \hat{\mu})(o_t - \hat{\mu})^T}_{\text{Sample covariance}}$$

GMM: Parameter-Suche via Ableitung nicht lösbar  $\rightarrow$  EM-Algorithmus

Kullback Leibler (KL) divergence:

$$D(p_{\text{true}} \| p) = \int_{\mathbb{R}^d} p_{\text{true}}(x) \ln \frac{p_{\text{true}}(x)}{p(x)} dx$$

Berechnet Ähnlichkeit zwischen zwei pdfs.

## Bayesian Parameter Estimation

$\theta$  immer noch unbekannt, aber Prior  $p(\theta)$  bekannt

$$\text{Likelihood } p(x|s=1) = \int_{\Theta} p(x|\theta) \cdot p(\theta|s=1) d\theta$$

$$\frac{p(\theta|s=1)}{\text{posterior}} = \frac{p(s=1|\theta) \cdot p(\theta)}{p(s=1)} = \frac{1}{C} \cdot \frac{p(s=1|\theta) \cdot p(\theta)}{\text{Likelihood} \quad \text{prior}}$$

$$\text{Posterior-Update rekursiv: } p(\theta|s_i) = \frac{1}{C_i} \cdot p(\theta|s_{i-1}) \cdot p(s_i|\theta)$$

$$i=1, 2, \dots, T$$

Für wenige Daten ist rekursiver Bayes besser als ML, für viele Daten oft ähnliche Ergebnisse.  $\rightarrow$  für adäquaten Prior

## Expectation-Maximization (EM) Algorithmus

Unteranderem: Lösung für ML-Parameter-Schätzung für GMMs

Ansatz:  $LL(\theta|s) - LL(\theta'|s)$  für  $\theta$  maximieren

2 sich abwechselnde Schritte: Expectation-Step (Erwartungswert berechnen)  
Maximization-Step (Maximum bestimmen)

$\rightarrow$  konvergiert, aber nicht zwingend gegen globales Maximum

EM für GMM:

Initialisierung: Welche Daten zufällig Gruppen zuordnen für jede Menge ML-Schätzung und  $c_k^{(0)} = \frac{T}{K}$

INITIALIZE  $\theta^{(i=0)}$ ,  $i=1$

Do  $i \leftarrow i+1$

FOR  $k=1, 2, 3, \dots, K$

$$c_k^{(i+1)} = \frac{1}{T} \sum_{t=1}^T P(u_t=k | \alpha_t, \theta^{(i)})$$

$$\mu_k^{(i+1)} = \frac{\sum_{t=1}^T P(u_t=k | \alpha_t, \theta^{(i)}) \cdot \alpha_t}{\sum_{t=1}^T P(u_t=k | \alpha_t, \theta^{(i)})}$$

$$\Sigma_k^{(i+1)} = \frac{\sum_{t=1}^T P(u_t=k | \alpha_t, \theta^{(i)}) \cdot (\alpha_t - \mu_k^{(i+1)}) (\alpha_t - \mu_k^{(i+1)})^T}{\sum_{t=1}^T P(u_t=k | \alpha_t, \theta^{(i)})}$$

E und M-Schritt in einer

UNTIL  $i=i_{\max}$  or convergence

$$P(u_t=k | \alpha_t, \theta^{(i)}) = \frac{c_k^{(i)} \cdot N(\alpha_t | \mu_k^{(i)}, \Sigma_k^{(i)})}{\sum_{k=1}^K c_k^{(i)} \cdot N(\alpha_t | \mu_k^{(i)}, \Sigma_k^{(i)})}$$

Nonparametric: wir nehmen keine Struktur für die Verteilung an

Histogramm-Methode: einfachster Weg Verteilung zu schätzen

Probleme: wieviele Bins?

Schätzen der PDF über Volumina:  $p_n(x) = \frac{k_n/n}{V_n}$

$k_n$ : number of samples in  $V_n$   
 $n$ : total number of samples  
 $V_n$ : some interval/area/volume centered around  $x$

$V_n$  small: High resolution, aber auch Rauschen

$V_n$  high: geglättete PDF

$\rightarrow$  entweder  $V_n$  oder  $k_n$  festlegen, abhängig von  $n$ , der anderen ist dann von den Daten abhängig

1) Volumen fixed  $V_n = V_1 / \sqrt{n} = h_n$  2) # samples fixed to  $k_n = \sqrt{n}$

Kernel-Based Methods (z.B.  $V_n = V_1 / \sqrt{n}$ )

Definieren Kernel zum Zählen der Samples im Volumen, z.B.  $\varphi(u) = \sum_{i=1}^n \begin{cases} 1 & \text{if } f_{d=1..d}(x_i) \leq 0.5 \\ 0 & \text{else} \end{cases}$

$$k_n = \sum_{i=1}^n \varphi\left(\frac{x - x_i}{h_n}\right)$$

Verschiedene Kernel möglich, aber sie müssen alle selbstne PDF sein.

Kernel density estimation

$$p_n(x) = \frac{1}{n} \cdot \frac{k_n}{V_n} = \frac{1}{n} \sum_{i=1}^n \varphi_n(x - x_i)$$

$$\varphi(u) \geq 0$$

$$\int \varphi(u) du = 1$$

auf jeden muss Kernen mit  $n$  abnehmen, ob nicht

zu schnell, dann:  $\lim_{n \rightarrow \infty} V_n = 0$  und  $\lim_{n \rightarrow \infty} n \cdot V_n = \infty$

Mögliche Kernel: - Uniform Kernel

$$\Rightarrow V_n = V_1 / \sqrt{n} \text{ oder } V_n = V_1 / n^{1/d}$$

- Multivariate Gaußkernel

$$\varphi(u) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{\|u\|^2}{2}}$$

- Generalized Multivariate kernel

$$\text{imang case} \Rightarrow p_n(x) = \frac{1}{n} \sum_{i=1}^n \frac{1}{V_n} \varphi_n\left(\frac{x - x_i}{h_n}\right)$$

Für  $n \rightarrow \infty$ :  $p_n(x) \approx p(x)$

$k_n$ -nearest-Neighbour PDF-Estimation (z.B.  $k_n = \sqrt{n}$ )

z.B.  $k_n = k_1 \cdot \sqrt{n} \rightarrow$  Volumen wird angepasst

- kleine  $k_n$ : - PDF wird „spitzig“ - Risiko von Überfitting

- große  $k_n$ : - glatte PDF - Risiko detaillierte Entscheidungsgerüste zu verlieren

$n \rightarrow \infty$ : konstante PDF  $p_n(x) \approx p(x)$

Konvergenz:  $\lim_{n \rightarrow \infty} k_n = \infty \quad \lim_{n \rightarrow \infty} k_n/n = 0$

$$\rightarrow \text{z.B. } k_n = k_1 \cdot \sqrt{n}$$

## Zusammenfassung Mustererkennung

a posteriori:

$$p_n(x) = \frac{k_n/n}{V_n} \rightarrow p_n(x, s=i) = \frac{k_n^{(i)}/n}{V_n}$$

$$\rightarrow P_n(s=i | x) = \frac{k_n^{(i)}}{k_n} = \frac{\text{number of samples in } V_n \text{ belonging to class } i}{\text{number of samples in } V_n}$$

→ for minimum error-rate decide for most represented class in volume.

Idee: da it even simpler: decide just for nearest neighbour.  $s^* = \arg \min_i D(x, c_i)$

→ Generalisierung: nicht nur nächster Nachbar, sondern k-nearest Neighbors.

$$s^* = \text{majority class}(D^{(k)}(x, 0))$$

$i_1, i_2, \dots, i_k$

### Metriken

Reflexivity:  $D(x, y) = 0$  nur wenn  $x=y$

Symmetry:  $D(x, y) = D(y, x)$

Triangle inequality:  $D(x, y) + D(y, z) \geq D(x, z)$

Minkowski metric:

$$D(x, y) = L_p(x, y) = \left( \sum_{j=1}^d |x_j - y_j|^p \right)^{\frac{1}{p}}$$

$p=1$ : Manhattan Metrik  $L_1$  norm

$p=2$ : Euklidische Metrik  $L_2$  norm

$p \rightarrow \infty$ : Maximum Metrik  $L_\infty$  norm

$$L_1(x, y) = \sum_{j=1}^d |x_j - y_j|$$

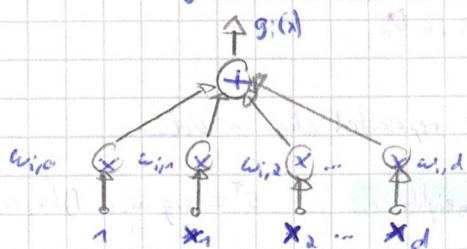
$$L_2(x, y) = \sqrt{\sum_{j=1}^d |x_j - y_j|^2}$$

$$L_\infty(x, y) = \max_j (|x_j - y_j|)$$

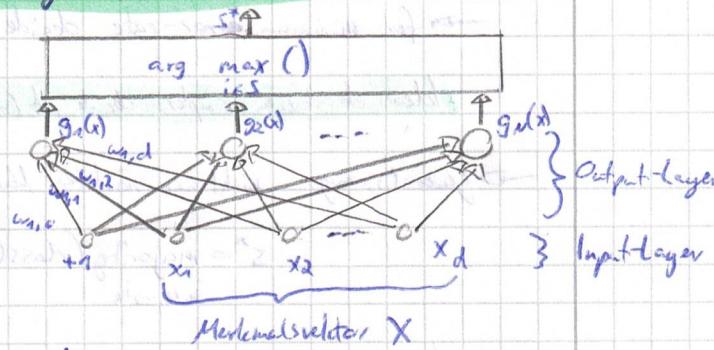
5. da: schätzen der Likelihood-Koeffizienten von Trennfunktionen zu berechnen.

Ziel: direkt Parameter einer linearen Trennfunktion finden

$$g_i(x) = w_i^T x + w_{i,0} = \sum_{j=1}^d w_{i,j} x_j + w_{i,0} \quad w_{i,0} = b_i \triangleq \text{Bias}$$



Single-layer neural network



### 2-Klassen-Fall:

Legt Hypothese  $g(x)=0$  in Raum. Entscheidende für Klasse, je nach Seite der Ebene, also  $g(x)>0$  oder  $g(x)<0$

### Mehrklassen-Fall:

wenn man es in mehrere 2-Klassprobleme zerlegt  $\rightarrow$  mehrdeutige Regeln

Es nicht machen!

Statt Klassen:  $s^* = \arg \max_{i \in S} g_i(x)$

$\rightarrow$  Simply connected convex decision regions

Bias  $w_{i,0}$  oft mit in den Gewichtsvektor:

$$g_i(x) = w_i^T x + w_{i,0} = a_i^T y$$

$$y = \begin{pmatrix} x \\ 1 \end{pmatrix}$$

Lineare Trennbarkeit besteht, wenn es einen Gewichtsvektor  $a$  gibt, der alle Samples korrekt klassifiziert

Einfacher Trick: Mappe alle Vektoren  $y_2 \rightarrow -y_2 = y'_2$  und  $y_1 \rightarrow y'_1$ . Suche nun Vektor  $a$ ,  $M=2$

der für alle  $y'$   $g(x) = a^T y' > 0$  erfüllt. Bei  $g(x) = a^T y' > b > 0$

$\Rightarrow$  Scharfe Trennlinie

### Basic Gradient Descent

Idee: Definiere statische Kostenfunktion  $J(a)$ , die alle Trainingsdaten erfasst und minimal wird, wenn  $a$  ist ein Lösungsvektor ist.

INITIALIZE  $a(0) = 0$ ,  $\theta = \epsilon$  (small number),  $\mu = 1$ ,  $k = -1$

DO:  $k = k + 1$

$$a(k+1) = a(k) - \mu \cdot \nabla J(a(k))$$

UNTIL  $\|a(k+1) - a(k)\| < \theta$

RETURN  $a(k+1)$

# Zusammenfassung Mustererkennung

## Batch Perceptron

Idee: Maximiere die Anteile  $a^T y \leq 0$  der falschklassifizierten Trainingsdaten,

$$\text{bzw. minimiere: } J_\alpha(a) = \sum_{y \in \mathcal{Y}_{\text{err}}} (-a^T y) \Rightarrow \nabla J_\alpha(a) = -\sum_{y \in \mathcal{Y}_{\text{err}(a)}} y'$$

$\rightarrow$  setz  $\nabla J_\alpha(a)$  in basic Gradient Descent ein.

keine smooth Kostefunktion

### Margin

Werte alle Trainingsvektoren mit  $a^T y \leq b(a)$  als fehlklassifiziert

Theoretisch: jeder Margin  $b(a) \geq 0$  erfüllt. Praktisch:  $b(a) = \mu(a) \cdot \left\| \sum_{y \in \mathcal{Y}_{\text{err}(a)}} y \right\|^2$

### Step Size

Theoretisch: jede Schrittgröße  $\mu(k) \geq 0$  konvergiert. Praktisch: Abfallende Schrittgröße wählen

$$\text{z.B.: } \mu(k) = \mu(0) / (k+1)$$

## MMSE Approaches

Idee: nicht nur misklassifizierte, sondern alle Trainingsvektoren und löse Lineare Gleichungen

$$a^T y_e = b_e > 0 \quad \Rightarrow \quad Y^T a = b \quad \text{mit } Y^T = (y_1^T, y_2^T, y_3^T, \dots, y_n^T)^T \quad a = (a_0, a_1, \dots, a_d)^T$$

$\rightarrow$  keine exakten Lösungen existieren,  $a$  ist überbestimmt

$\rightarrow$  Minimiere  $Y^T a - b$  durch Minimieren von:

$$J_S(a) = \frac{1}{2} \|Y^T a - b\|^2 = \frac{1}{2} \sum_{e=1}^E (a^T y_e - b_e)^2 \quad (\text{MMSE Kriterium})$$

$$\Rightarrow \nabla J_S(a) = Y^T (Y^T a - b) = \sum_{e=1}^E (a^T y_e - b_e) y_e$$

↑  
Pseudo inverse  
LMS

$\rightarrow$  Stetige Kostefunktion

## MMSE by Pseudo inverse

$$a = Y^+ b \quad \text{mit } Y^+ = (Y^T Y)^{-1} Y^T \quad \text{Pseudoinverse } Y^+$$

wenn  $b = 1$ , dann konvergiert für  $T \rightarrow \infty$  gegen Bayes Trennfunktion

- optimiert für  $T \rightarrow \infty$  nicht für minimalen Fehler
- funktioniert gut auf trennbaren und nicht-trennbaren Daten, generalisiert gut

## MMSE by LMS

$$\nabla J_s(a) = \sum_{t=1}^T (a^T y_t - b_t) y_t \quad \rightarrow \text{(ohne Summe) in Batch Perceptron einsetzen}$$

→ abfallendes  $\mu$  ist ein Muss

- glatte Kettfunktion

Geh so über alle Trainingsvektoren

Mehrklassen: - Batch-Perceptron anwendbar

MMSE müssen angepasst werden.

$$g_i(x) = a_i^T y = 1 \quad \text{für alle } y \in \mathcal{Y}; \quad i \in S = \{1, 2, \dots, N\}$$

$$g_i(x) = a_i^T y = 0 \quad \text{für alle } y \notin \mathcal{Y}_i$$

$$s^* = \arg \max_{i \in S} g_i(1)$$

ausführliche Erklärung der Schritte

2-Klassen-Problem: ( $N=2$ )

$$g(x) = w^T x + b = \phi^T y$$

Ausgang: Batch Perceptrons Lösung hängt von Initialwerten und bei LMS auch von Reihenfolge der Trainingssamples ab

Idee: Wähle aus der Menge aller möglichen Lösungen die Lösung aus, die die

Margen  $\Delta = \min_{t \in T} \left( \frac{|g(x_t)|}{\|w\|} \right)$  maximiert:  $\Delta \rightarrow \Delta_{\max}$ . Diese Lösung sollte am besten generalisieren.

### Zero-Error-Case

Einführen einer Merkmalstransformation, vererbt linear:  $\phi(x) = x$

$$g(x) = w^T \phi(x) + b$$

Einführen von Klassenzetzen:

$$g(x_i) > 0 \Rightarrow z_i = 1 \quad g(x_i) < 0 \Rightarrow z_i = -1$$

Distanz eines transformierten Datenvektors  $\phi(x_i)$  zur Hyperebene:

$$\|r\| = \frac{|g(x_i)|}{\|w\|} = \frac{z_i g(x_i)}{\|w\|}$$

$$\text{Margin } \Delta = \min_{t \in T} \|r\|$$

$$\Rightarrow \Delta_{\max} = \max_{w \in \mathbb{R}^d, b \in \mathbb{R}} \left( \min_{t \in T} \left( \frac{z_t g(x_t)}{\|w\|} \right) \right)$$

$$(w, b)_{\text{opt}} = \arg \max_{w, b} \left( \min_{t \in T} \left( \frac{z_t g(x_t)}{\|w\|} \right) \right)$$

$$\text{Minimierung unabhängig von } \|w\| \Rightarrow (w, b)_{\text{opt}} = \arg \max_{w, b} \left( \frac{1}{\|w\|} \min_{t \in T} (z_t g(x_t)) \right)$$

4 Schritte:

1) bestimmen, dass  $z_t (w^T \phi(x_t) + b) = 1$  für die Vektoren  $\phi(\tilde{x})$  am nächsten an der Entscheidungsebene gilt.  $\rightarrow$  Support-Vektoren

Damit gilt für alle  $x_i$ :  $z_i (w^T \phi(x_i) + b) \geq 1$

2) Es wird immer mindestens ein Support-Vektor existieren. Nachdem Maximalen sind es mindestens 2. Da das Minimum = 1 ist, muss dann nur

$$\frac{1}{\|w\|} \text{ maximiert werden, bzw. } \min_{w, b} \left( \frac{1}{2} \|w\|^2 \right) \text{ mit Constraint: } z_t (w^T \phi(x_t) + b) \geq 1$$

3) Einführen von Lagrange-Multiplikatoren:  $\lambda = (\lambda_1, \dots, \lambda_T)^T$  mit  $\lambda_t \geq 0$

$$\min_{w, b} L_p(w, b, \lambda) \quad \text{mit } L_p(w, b, \lambda) = \frac{1}{2} \|w\|^2 - \sum_{t \in T} \underbrace{\lambda_t \cdot (z_t (w^T \phi(x_t) + b) - 1)}_{= 0}$$

Partielle Ableitungen nach  $w$  und  $b$  und wieder einsetzen führt zu:

$$\max_{\lambda} L_p(\lambda) = \sum_{t \in T} \lambda_t - \frac{1}{2} \sum_{t \in T} \sum_{j \in T} \lambda_t \lambda_j z_t z_j k(x_t, x_j)$$

Kern-Funktion:

$$k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

$$\text{mit Constraint: } \sum_{t \in T} \lambda_t z_t = 0$$

Wenn Lagrange-Multiplikatoren dann bekannt sind, gilt:

SVM-Klassifikation:

$$g(x) = w^T \phi(x) + b = \sum_{j \in \mathcal{I}_{\text{sup}}} \lambda_j z_j k(x, x_j) + b$$

$$\xi = \frac{1}{|\mathcal{I}_{\text{sup}}|} \sum_{t \in \mathcal{I}_{\text{sup}}} \left( z_t - \sum_{j \in \mathcal{I}_{\text{sup}}} \lambda_j z_j k(x_t, x_j) \right)$$

Vorteil der SVMs: Zur Klassifikation werden nur die Support-Vektoren benötigt

Merkmalstransformation  $\phi(x)$  mappt Datenvektoren in einen hochdimensionalen Merkmalsraum  $\mathcal{H}$ , Hilbert-Space genannt.

$\phi(x)$  interessiert meist nicht, sondern kernel.

Beispiele: Linear:  $k(x, x') = x^T x'$  Polynomial:  $k(x, x') = (x^T x' + r)^p$

Radial Basis Function:  $k(x, x') = e^{-\gamma \|x-x'\|^2}$   $\rightarrow$  Gauß-Kernell  $\gamma = \frac{1}{2\sigma^2} > 0$   
(RBF)

Sigmoidal:  $k(x, x') = \tanh(\gamma x^T x' + r)$

Eigenschaften der SVMs:

- Optimierungsproblem ist konkav  $\Rightarrow$  jedes lokale Optimum ist auch global
- durch nicht-linearen Kernel sind auch linear nicht-trennbare Daten trennbar

### Relaxed Case

Auch Fehlklassifizierungen erlaubt  $\rightarrow$  Slack-Variablen  $\xi_t \geq 0 \quad t \in \mathcal{T}$

$\xi_t = 0 \rightarrow$  auf richtiger Seite der Margin befindet

$0 < \xi_t < 1 \rightarrow$  zwischen margin befindet und decision boundary

$\xi_t \geq 1 \rightarrow$  hinter der Decision boundary (Fehlklassifiziert)

SVMs sind alle Vektoren mit  $\xi_t \geq 0$  und  $\lambda > 0$

$\rightarrow$  neue Lagrange-Multiplikatoren:  $\mu_t \cdot \xi_t = 0$  und  $0 \leq \mu_t \leq C$

### SVM-Klassifikation (soft margin)

$C \hat{=} \text{Kostentoller}$

!

$$g(x) = \sum_{j \in \mathcal{I}_{\text{sup}}} \lambda_j z_j k(x, x_j) + b$$

neue Constraint:  $0 \leq \lambda_t \leq C$

Mehrklassen-Probleme:

SVM eigentlich nur für 2-Klassen-Probleme  $\rightarrow$  Training N-Klassif. "One-versus-the-

Normalisierung

- Trainings- (und Test-) Daten sollten normalisiert werden, damit einzelne große Vektoren keinen zu großen Einfluss haben

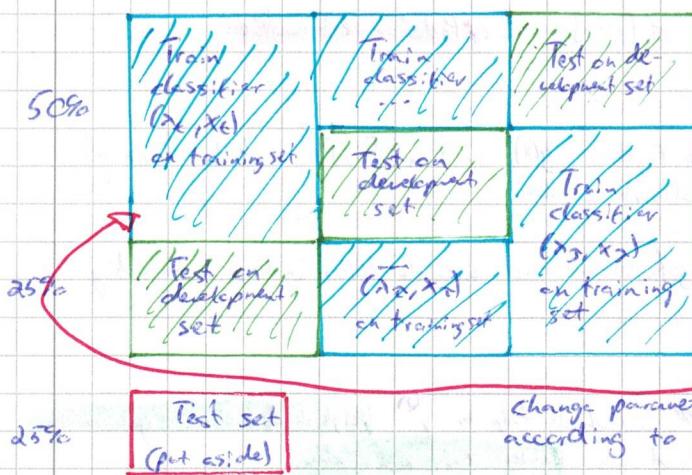
- wenn eine Klasse deutlich mehr Trainingsdaten hat  $\rightarrow$  Klassa-verschiedene Kostenfunktionen

- SVMs, First steps:
- 1) Apply linear scaling to the data
  - 2) try first with RBF-kernel
  - 3) Use 2D grid search to find best C and g.
  - 4) Train with optimal C and g. die SVM.
  - 5) Perform test on normalized test data

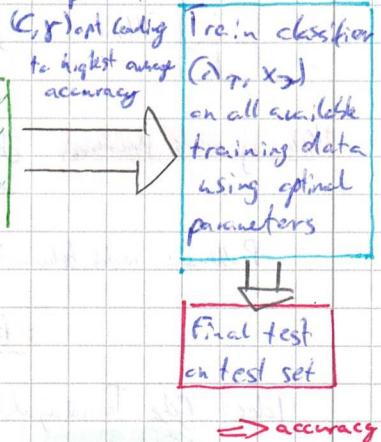
### K-fold Cross-Validation

Start with first guess for  $(C, g)$

Totally available  
data ( $x_1, x_2$ )



After grid search:  
Select parameters

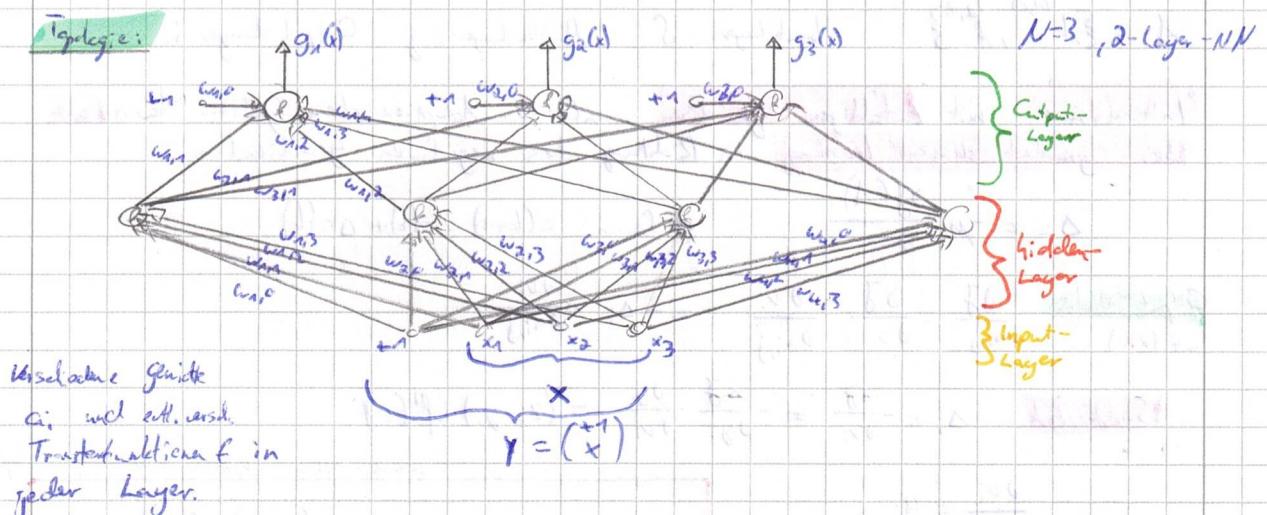


### Neural Networks

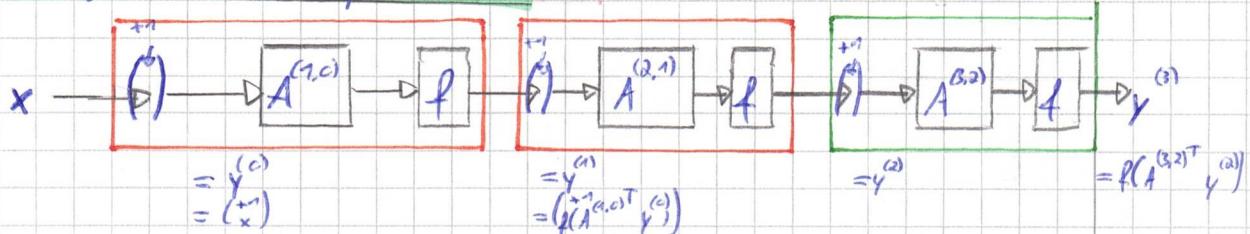
Bisher bei Single-layer-Networks: Lineare Transfunktion  $g(x) = a^T y$

Jetzt neu:

Nicht-lineare Transfunktion  $g(x) = f(a_i^T y)$



Multi-layer-Netzwerke in compact notation:



## Activation functions / transfer functions (f)



Linear:  $y_i := f(v_i) = v_i \in \mathbb{R}$  → Für Output-Layer von Funktions-Approximation



Sigmoid (tanh):  $y_i := \tanh(v_i) = 1 - \frac{e^{-v_i}}{1+e^{-v_i}} \in [-1; 1]$  klassische Wahl für hidden-Layer



Sigmoid (logistic):  $y_i := \frac{1}{1+e^{-v_i}} = \frac{1}{2}(1 + \tanh(\frac{v_i}{2})) \in [0; 1]$



ReLU (rectified linear unit):  $y_i := \max(v_i, 0)$  erhält schnelle Konvergenz und schnelle Berechnung → DeepLearning



Softmax:  $y_i := \frac{e^{v_i}}{\sum_{j=1}^n e^{v_j}} \in [0; 1]$  → Posterior estimation

Kolmogorov's Theorem: Jede kontinuierliche Funktion  $g(x)$  mit  $x \in [0; 1]^n$  kann mit 2-Layer-NN realisiert werden

Probleme: nicht bekannt, welche Aktivierungsfunktionen und wieviele Nodes in Layer notwendig

### Backpropagation

Idee: Gebe Trainingsdaten  $y^{(0)}$  in  $W$  und erhalte Outputs  $y^{(1)}$ . Vergleiche diese mit gewünschten Outputs  $z$ . Verändere Gewichte  $A$  um Fehler zu minimieren.

2-lag.-NN:

$$(x) \quad y_j^{(1)} = f\left(\underbrace{\sum_{d=0}^D a_{j,d}^{(1)} g_d}_{v_j^{(1)}}\right)$$

$$(xx) \quad y_i^{(2)} = f\left(\underbrace{\sum_{j=0}^n a_{i,j}^{(2)} \cdot y_j^{(1)}}_{v_i^{(2)}}\right) = g_i(x)$$

Kosten-Funktion:  $J(A) = \frac{1}{2} \sum_{i=1}^n (z_i - y_i^{(2)})^2 = \frac{1}{2} \|z - y^{(2)}\|^2$

„Mean square error“ (MSE)

$$A = \{A^{(1)}, A^{(2)}\} \quad \text{Input-Layer: } S \quad \text{Hidden-Layer: } j \quad \text{Output-Layer: } i.$$

Initialisiere mit zufälligen Gewichten  $a(k=0)$ . Adaptiere diese Gewichte dann wie bei gradient descent learning in Richtung des negativen Gradienten.

$$\Delta a = -\mu \frac{\partial J(A)}{\partial a} \quad \rightarrow \text{Dann } a(k+1) = a(k) + \Delta a(k)$$

Output-Schicht:  $\frac{\partial J}{\partial a_{i,j}} = \frac{\partial J}{\partial v_i} \frac{\partial v_i}{\partial a_{i,j}} = -\Delta_i \cdot \frac{\partial v_i}{\partial a_{i,j}}$

Sensitivität:  $\Delta_i = -\frac{\partial J}{\partial v_i} = -\frac{\partial J}{\partial y_i} \cdot \frac{\partial y_i}{\partial v_i} = (z_i - y_i) \cdot f'(v_i)$

Lernrate • Fehler • Ableitung d. Aktivierung • Input

$$\frac{\partial v_i}{\partial a_{i,j}} = g_j$$

$$\rightarrow \Delta a_{i,j} = \mu \cdot \Delta_i \cdot g_j = \mu \cdot (z_i - y_i) \cdot f'(v_i) \cdot g_j$$

# Zusammenfassung Mustererkennung

## Hidden-Schicht mit ( $x$ )

$$\frac{\partial J}{\partial a_{j,s}} = \frac{\partial J}{\partial g_j} \cdot \frac{\partial g_j}{\partial v_j} \cdot \frac{\partial v_j}{\partial a_{j,s}}$$

$$\begin{aligned}\frac{\partial J}{\partial g_j} &= \frac{\partial}{\partial g_j} \left[ \frac{1}{2} \sum_{i=1}^N (z_i - y_i)^2 \right] = -\sum_{i=1}^N (z_i - y_i) \cdot \frac{\partial g_j}{\partial g_j} = -\sum_{i=1}^N (z_i - y_i) \cdot \frac{\partial g_j}{\partial g_j} = -\sum_{i=1}^N (z_i - y_i) \cdot f'(v_j) \cdot a_{i,j} \\ &= -\sum_{i=1}^N a_{i,j} \cdot \Delta_i \quad \text{Satzweise: } \Delta_j = \frac{\partial J}{\partial g_j} \cdot \frac{\partial g_j}{\partial v_j} = f'(v_j) \cdot \sum_{i=1}^N a_{i,j} \cdot \Delta_i\end{aligned}$$

$$\frac{\partial g_j}{\partial v_j} = f'(v_j)$$

$$\frac{\partial v_j}{\partial a_{j,s}} = y_s$$

$$\Delta a_{j,s} = \mu \cdot \Delta_j \cdot y_s$$

## Backpropagation - Protokolle

### Stochastic Backpropagation

Gute in zufälliger Reihenfolge über alle Trainingsdaten innerhalb einer Epoche. Update genügte nach jedem Vektor. Überprüfe am Ende jeder Epoche Abbruchkriterium und beide deneingesetztes abrunden.

- Gut für große und/oder redundante Trainingsdaten

### Online Backpropagation

Trainiere nachander über Trainingsdaten und aktualisiere nach jedem Vektor die genügte. Keine zufällige Reihenfolge und keine Wiederholung

- kann genutzt werden um Netze online, also im Betrieb zu trainieren.

### Batch Backpropagation

Trainiere nachander in gegebener Ordnung. Ihr die Trainingsdaten und aktualisiere nach jedem Vektor alle Update-Terme der Gewichte. Update die Gewichte am Ende der Epoche.

- Gut für kleine und mittlere Trainingsdaten-Sets.

- auch Mini-Batch möglich. Mehrere kleinere Batchs in einer Epoche.

## Kostenfunktionen

Minimum squared error (MSE):  $J(\lambda) = \sum_{t=1}^T \sum_{i=1}^N \frac{1}{2} (z_{ti} - y_{ti}^{(a)})^2 = \sum_{t=1}^T \frac{1}{2} \|z_t - y_t^{(a)}\|^2$

→ Bestrafte grobe Fehler

Minimum absch. error (MAE):  $J(\lambda) = \sum_{t=1}^T \sum_{i=1}^N |z_{ti} - y_{ti}^{(a)}|$

→ Reicht die Dominanz grober Fehler (da diese nicht so wichtig für Entscheidungsergebnisse sind)

Minimum cross entropy (MCE):  $J(\lambda) = \sum_{t=1}^T \sum_{i=1}^N z_{ti} \ln(z_{ti}/y_{ti}^{(a)}) = -\sum_{t=1}^T \ln(y_{ti}^{(a)})$

→ Misst Distanz zwischen zwei Wahrscheinlichkeitsverteilungen (KL-Divergenz)

$$z_{ti} = \begin{cases} 1 & \text{if } s_t = i \\ 0 & \text{otherwise} \end{cases}$$

	Funktions-Approximation	Klassifikation	Klassifikation mit Postiors
Activation function	Sigmoid/ReLU hidden output	Sigmoid/ReLU hidden output	Sigmoid/ReLU hidden output
Input warter	$N^{(0)} = 1$	$N^{(0)} = N$	$N^{(0)} = N$
Target values	$z_{ti} = F(x_t)$	$z_{ti} = \begin{cases} 1 & \text{if } s_t = i \\ 0 & \text{otherwise} \end{cases}$	$z_{ti} = \begin{cases} 1 & \text{if } s_t = i \\ 0 & \text{otherwise} \end{cases}$
Cost function	MSE	MSE or MAE	MSE, MAE or MCE
Stop criterion	nondecreasing MSE	nondecreasing MSE, MAE or ER	nondecreasing MSE, MAE, MCE or ER

$$\text{kd: } \sum_{i=1}^N g_{ti}^{(a)} = 1 \quad 0 \leq g_{ti}^{(a)} \leq 1$$

## Posteriors

Möglich, Posteriors zu erhalten, wenn: - MSE (oder NC $\epsilon$ ) Kostenfunktion

- gängig Neuronen in Hidden-Layer
- unbegrenztes Training material
- Softmax im Output-Layer
- One-Hot-Training

→ Wenn Trainingsprior und echter Prior nicht übereinstimmen, dann ist es besser nach dem Training aus den Posteriors Likelihoods zu berechnen.

→ Posteriors werden in der Regel überschätzt → Overconfidence

## Training:

Zum Überprüfen der Qualität während des Trainings wird ein Validierungsdaten-Set verwendet  
- Stop-Kriterium darin: Kosten auf Validierungsset → early stopping

Dannen-Regel für optimale Parameteranzahl:

$$|A| \approx T/10$$

$T =$  Anzahl Trainingsdaten

$$\begin{aligned} 2-N &\rightarrow NN: \\ |A| &\approx 5 \cdot N \end{aligned}$$

## Normalisierung der Input-Vektoren

Merkmal, die größere Werte annimmt als andere, würde im Training schneller in der Kostenfunktion dominieren. → Merkmalsvektoren vorher normalisieren.

→ so stellen, dass jedes Merkmal im Merkmalsvektor mittelwertsfrei ist und Varianz 1 hat.

→ Skalieren Trainings-, Validierungs- und Test-Daten mit den gleichen Faktoren

## Batch Normalization

intend covariance shift verlangsamt Lernen. → Normale Merkmale nicht nur im Input-Layer, sondern auch in Hidden-Layer. Nutzen für Mini-Batches und normiere pro MB.

→ ermöglicht höhere Lernraten und ist zugleich Form der Regularisierung

## Initialisierung der Gewichte

Mit 0 initialisieren würde das Training der Gewichte verhindern.

↪ initialisieren mit gleichverteilten Zufalls-werten → Dieser auf richtige Range achten

## Momentum

wenn Kostenfunktion plateausiert ist, kann stochast. BP sehr langsam werden

→ Momentum: beschreibt frühere Gradienten in Verbindung mit e.w.

## Regularisierung / Weight Decay

Große Gewichte sind schlecht für Generalisierung. → führt Regularisierung ein, der die Gewichte nach Update wiederverringert

$$\Delta a_{jS} = a_{jS} - \mu \Delta a_{jS}$$

$$J(A) = J(A) + \frac{\epsilon}{2} a^2$$

## Hervorheben von Klassen / Unbalance DataSets

In manchen Anwendungen sind manche Klassen wichtiger als andere → führt importance weight in Kosten ein

$$J(A) = \sum_{i=1}^T \sum_{j=1}^N \beta_i \cdot J_{ij}$$

Kann auch gewichtet werden um ungewichtete Trainingssets aufzubauen, wenn Prior eigentlich gleich ist

# Zusammenfassung Mastererkennung

## Regularisierung

- ohne Regularisierung kommt es oft zu Overfitting auf Trainingsdaten, was zu schlechter Generalisierung auf Testdaten führt.

Bekannt:

- Early Stopping
- BP mit Momentum
- BP mit Weight Decay

## Dataset Augmentation

- Generieren „fake“ Daten  $\rightarrow$  mehr Trainingsschritte  $\rightarrow$  bessere Generalisierung
- Bilder z.B. in gedrehter Form mehrmals im Trainingsdatensatz einfügen
- Rauschen (hochrechte Energie, Mittelwert-frei) auf Trainingsdaten geben (jede Epoche neues Rauschen)

## Adding noise to the output targets

Manchmal sind Trainingsdaten falsch gelebtet  $\rightarrow$  füge (low power) Rauschen zu Zielen hinzu um Generalisierung zu verbessern

## $L_1$ und $L_2$ Parameter Regularisierung

$L_2$ : gleiche wie weight-decay  $J(A) = J(A) + \frac{\epsilon}{2} \|A\|_2^2$

$L_1$ :  $J(A) = J(A) + \epsilon \|A\|_1 = J(A) + \epsilon \sum_i |a_i|$



## Dropout

Verbessern der Generalisierung durch Kombinieren vieler verschiedener Netzwerke

$\rightarrow$  Wähle in jedem Batch nur eine zufällige Auswahl von Neuronen, ignoriere die anderen (Dropout). Die getauschten Gewichte werden in diesem Batch nicht trainiert

- wähle 10 bis 100-fach größere Lernrate oder sehr hohes Momentum  $\alpha \approx 0,95$  bis 0,99

- Gedachte Units im Test mit einziehen

## Multi-Task Learning

- 2 oder mehr Aufgaben aus gleichen Trainingsset zu trainieren? Trainiere nicht d komplexe Modelle, sondern einen gemeinsamen ersten Teil und 2 separate Hinterteile.  
 $\rightarrow$  shared parameters / parameter tying?

## Radial Basis Networks (RBN)

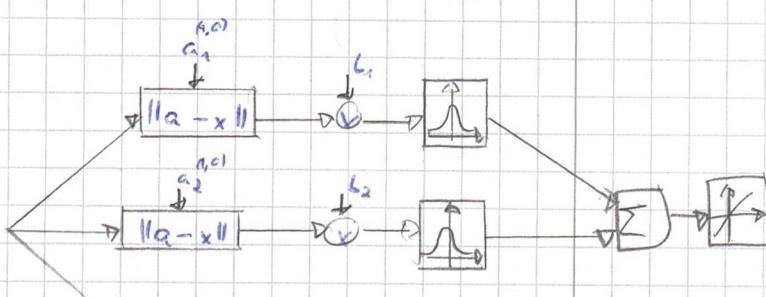
- für Funktionen-Approximation

- verwendet radial basis function (RBF) als Basisfunktion

$$\text{RBF-Aktivationsfunktion: } f(u_j) = e^{-\frac{\|x-a_j\|^2}{\sigma_j^2}}$$

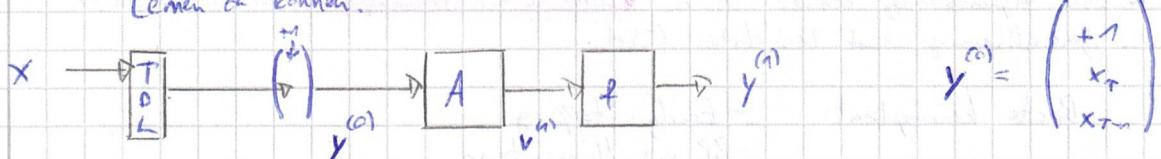
Trainieren deutlich schneller als feedforward NN mit linearer Basis-Funktion

$$g_i^{(1)} = f(\|a_i - x\| \cdot b_i) = f(u_i)$$



## Neuronale Netze mit Memory

Idee: Fügen einer tapped delay line (TDL) um zeitliche Zusammenhänge lernen zu können.



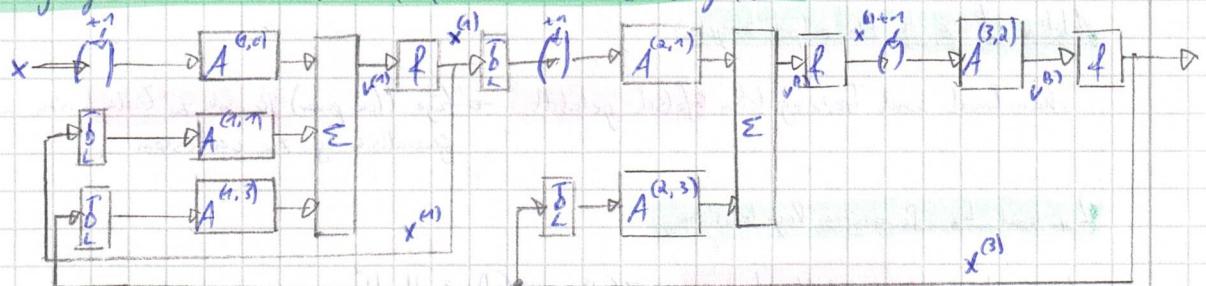
Kern zusammen mit Feedforward Netz zeitintensive short-term Strukturen lernen.

Spezialfälle:  $N=1$ , lineare Aktivierungsfunctionen  $\rightarrow$  Feedforward: FIR-Filter

Recurrent : IIR-filter

### Recurrent Neural Network (RNN)

Ausgang von Schichten wird auf früheren Schichten zurückgetragen.

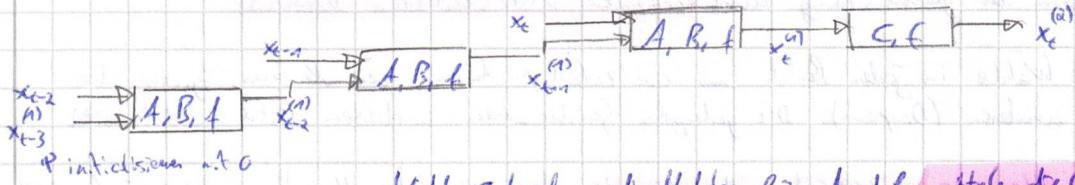


Minimales Delay für rückgetragene Inputs ist 1.

Training von RNNs:

### Backpropagation through time (BPTT)

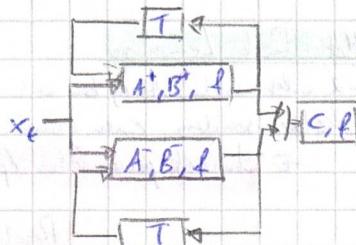
möglich mit Online-Backpropagation durch Abräumen der TDL bis zu gewisser Tiefe.



Wichtig: berechne die Updates für  $A$  und  $B$ , enttele die Updates, und wende sie dann erst auf  $A$  und  $B$  an - ansonsten kein stochastic BP möglich/sinnvoll.

### Bidirectional RNN (BRNN)

Trainieren Netz in positiver und negativer Richtung



# Zusammenfassung Mustererkennung

## Convolutional Neural Networks (CNN)

Manchmal ist gewisse Position eines Merkmalsvektors in der TFL un wichtig (Shift-invariance/translation invariance)

Lösung: weight tying.  $\rightarrow$  in der ersten Layer werden oft alle Merkmalsvektoren die gleiche Gewichte angewandt.

- Fully connected : Alle Neuronen innerhalb eines Merkmalsvektors haben eigene Gewichte, aber die versch. Merkmalsvektoren haben gleiche Gewichte
- Sparsely connected : jedes Neuron sieht nur einen Merkmalsvektor. Aber alle haben gleiche Gewichte

2D-CNN: basieren auf sparsely connected - Struktur. a: z.B. Kernel-Funktionen

- 1. Hidden-Layer erzeugt Feature Maps, z.B. Kanten detektion
- nach jeder convolutional Layer folgt normalerweise eine subsampling/pooling Layer.  
↳ keine Parameter, die weiter trainiert werden müssen
- Final-Layer:
  - immer fully connected.
  - Softmax für posteriors
  - SVM bei wenig Training material

## Trainingsprobleme bei RNN/BPTT und deep FF NNs

$$\prod_{j=1}^{T_0} F_{i,j}^{(r, \gamma-1)} \quad \text{mit } F_{i,j} = [f'(v_j) \cdot a_{i,j}]$$

$\rightarrow$  Backpropagierter Fehler neigt zu "Blow up" oder "Vanishing".

↳ oszilliert, instabil

↳ praktisch kein Lernen

- Betrifft RNNs durch das Abrollen und tiefe FF-Nets.

Lösung 1: 1. Pretraining der Gewichte als "restricted Boltzmann machine" (RBMs)

Schicht für Schicht.  $\Rightarrow$  benötigt tanh() als Aktivierungsfunktion

2. Dann BP mit diesen vorinitialisierten Gewichten

$\rightarrow$  durch Training mit unlabeled Data modellieren wir Einträge p(x). Diesen dann in BP zum Likelihood/Posterior untertrainieren

Lösung 2: Verwenden von ReLUs in Hidden Layer.  $\rightarrow f'(y)=1$  für positive y

$\rightarrow$  gut um Vanishing zu verhindern

- braucht kein Pretraining, zufällige Initialisierung möglich

- ReLU deutlich schnelleres Training als Sigmoid

- Risiko für Overfitting  $\rightarrow$  Regularisierung

### Lösung 3: Learning Residuals

#### Einführung von transparenten Bypass-Connections

→ BP ist besser dann Residuals ein trainieren, also Abrechnung von Bypass.

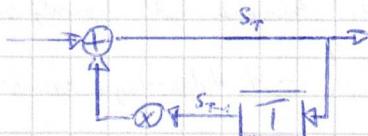
→ ohne Residuals performen sehr tiefe selbst auf Trainingsdaten schlechter als flachere NN. Mit Residuals sind auch sehr tiefe Netze trainierbar (652 im Papier)

### Lösung 4: Long-Short-Term-Memory (LSTM) Recurrent NNs

#### Einführung des Constant Error Cancell (CEC)

hinzufüllt zu CEC:

- Multiplikatives Input-Gate
- Multiplikatives Forget-Gate
- Multiplikatives Output-Gate



↳wendlicher Inputort → LSTM

Aufbau LSTM-Blocks S. 252

- LSTM lernen schnell, aber leider auch Fehler im Trainingsset ⇒ schlecht im Generalisieren

### Unsupervised Learning and Clustering

Unlabeled Data oft in großen Mengen vorhanden, Labeled Data ist teuer

- RBM pre-training
- autoencoder
- Clustering

### Anhöhlungsmaße

- Euklidische Distanz

- Kostefunktion MSE:  $J = \sum_{i=1}^n \sum_{x \in C_i} \|x - \mu_i\|^2$

### k-Means

- Platziere  $\mu$  der Cluster zufällig

- Setze  $\mu$  neu

→ ordne Datenpunkte den  $\mu$  zu

→ wiederhole, bis  $\mu$  nicht mehr ändert

### Hierarchisches Clustering

Agglomerative Approach: start: - jeder Datenpunkt ist eigenes Cluster

- kombiniere immer Cluster mit niedrigster Distanz, bis Anzahl an Clustern erreicht

Metriken: Minimum of sample distances

$$D(C_i, C_j) = \min_{x \in C_i, x' \in C_j} \|x - x'\|$$

Minimum of cluster means,

$$D(C_i, C_j) = \min_{\mu_i, \mu_j} \|\mu_i - \mu_j\|$$

Minimum of cluster sample dists

$$D(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i} \sum_{x' \in C_j} \|x - x'\|$$