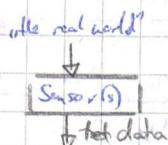


Zusammenfassung Mustererkennung Kapitel 1 Introduction

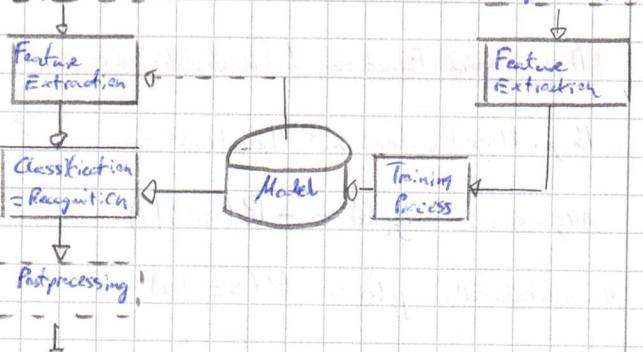
Was ist ein Muster? Anordnung von Formen, Teilen oder Elementen oder etwas, das eine Klasse oder Typ repräsentiert

Typische Schritte der Mustererkennung:

Test



Training



Feature Extraction:

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

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

Kapitel 2 Bayesian Decision Theory

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

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

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

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

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

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

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

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

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

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

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

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

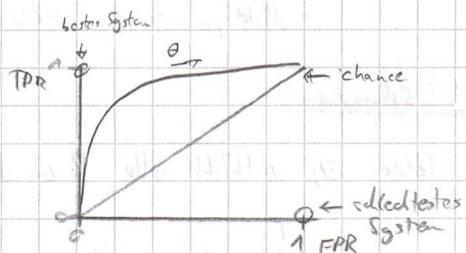
Erinnerung: $g(x) = LLR(x) + \log \frac{P(s=1)}{P(s=0)}$ $\rightarrow ER, TPR, FPR, FNR, \dots$ alle von θ abhängig

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

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

Receiver Operating Characteristic (ROC)

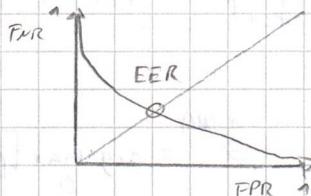
plottet TPR als Funktion von FPR(θ),



oder FNR als Funktion von FPR(θ)

$$EER = FAR = FNR = \text{Equal Error Rate}$$

optimales θ abhängig von Anwendung

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:

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

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

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

$$\text{AUC} = \int_{-\infty}^{\infty} \text{TPR} dFPR$$

$$= \int_{0}^{1} \text{TPR} d(1-FPR)$$

$$= \int_{0}^{1} \text{TPR} dFNR$$

$$= \int_{0}^{1} \text{TPR} d(1-P(s=1))$$

$$= \int_{0}^{1} \text{TPR} d(1-P(s=0))$$

$$= \int_{0}^{1} \text{TPR} dP(s=1)$$

$$= \int_{0}^{1} \text{TPR} dP(s=0)$$

$$= \int_{0}^{1} \text{TPR} dP$$

$$= \int_{0}^{1} \text{TPR} dP$$

Bayesian Parameter Estimation

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

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

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

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

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

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 θ 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

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

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_∞ norm

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

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

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

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

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 Maxima 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:

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

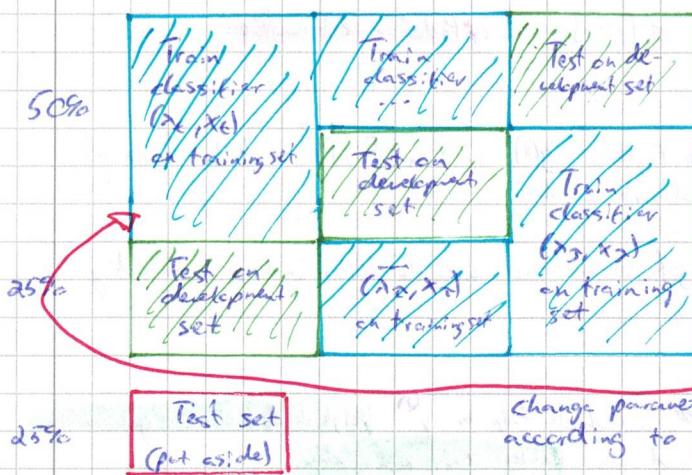
$$\text{mit Constraint: } \sum_{t \in T} \lambda_t z_t = 0$$

- 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

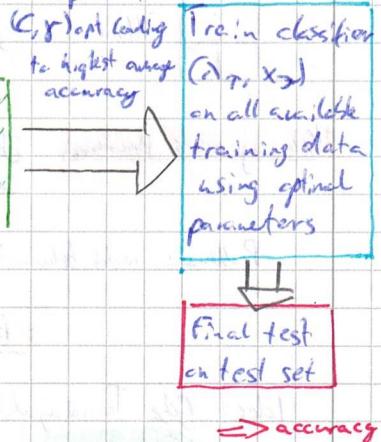
K-fold Cross-Validation

Start with first guess for (C, g)

Totally available
data (x_1, x_2)



After grid search:
Select parameters

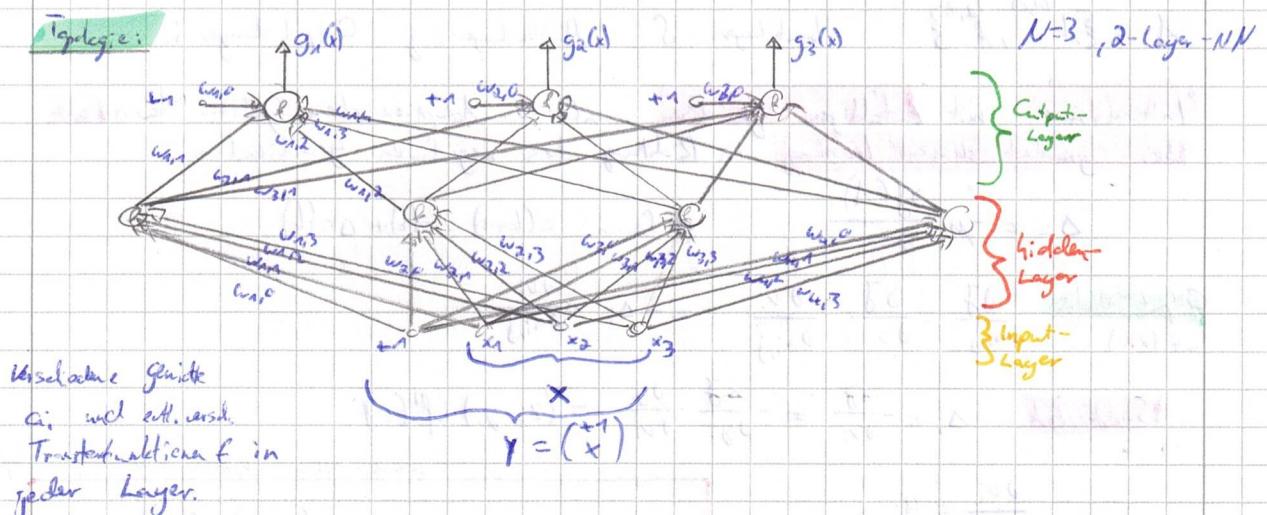


Neural Networks

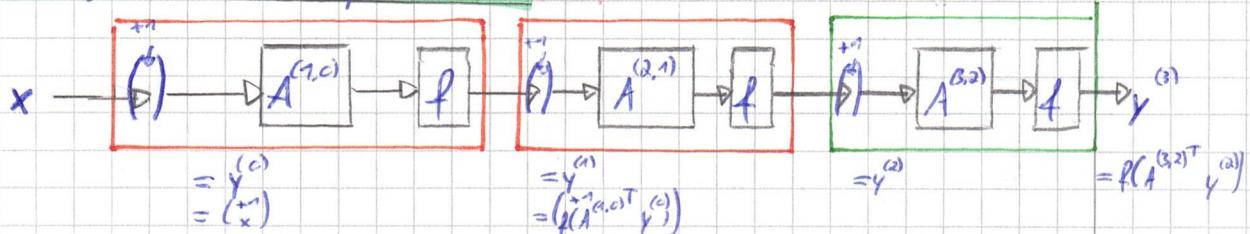
Bisher bei Single-layer-Networks: Lineare Transfunktion $g(x) = a^T y$

Jetzt neu:

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



Multi-layer-Networks in compact notation:



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 nacheinander 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 nacheinander 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 absolute 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/wieviel | $N^{(0)} = 1$ | $N^{(0)} = N$ | $N^{(0)} = N$ |
| 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 |

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

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)

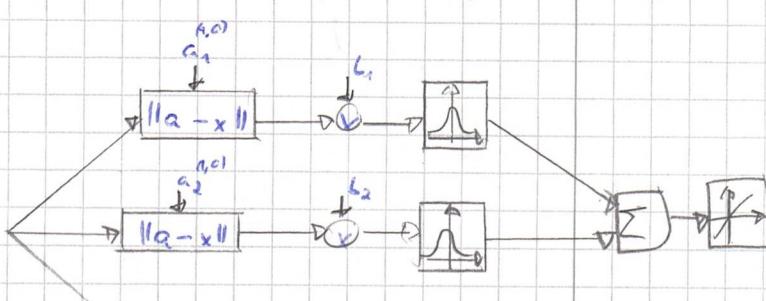
- für Funktionen-Approximation

- 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

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



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