HMM-basierte Online Handschrifterkennung — ein integrierter Ansatz zur Text- und Formelerkennung

Vom Fachbereich Elektrotechnik
der Gerhard-Mercator-Universität - Gesamthochschule Duisburg

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

genehmigte Dissertation

von

Andreas Kosmala

aus Herne

Referent: Prof. Dr.-Ing. habil. G. Rigoll

Korreferent: Prof. B. Hosticka Ph.D.

Tag der mündlichen Prüfung: 11. Dezember 2000

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter im Fachgebiet Technische Informatik des Fachbereichs Elektrotechnik an der Gerhard-Mercator-Universität Duisburg.
Dem Leiter des Fachgebietes Herrn Prof. Dr.-Ing. habil. Gerhard Rigoll gilt mein besonderer Dank für die Anregung zu dieser sehr interessanten Themenstellung, für die wertvollen Diskussionen und für die gewährten wissenschaftlichen Entfaltungsmöglichkeiten.
Prof. Ph.D. Bedrich Hosticka (Fraunhofer-Institut IMS, Duisburg) danke ich für die Übernahme des Korreferats und die kooperative Zusammenarbeit.
Danken möchte ich auch meinen Kollegen für die fundierten Diskussionen und die vielfältige Mithilfe. Insbesondere möchte ich mich bei Frau Dipl.-Ing. Anja Brakensiek für die Durchsicht dieser Arbeit und die zahlreichen Anmerkungen bedanken.
Ferner danke ich all den Versuchspersonen, die im Rahmen der Datenakquisition zu einer beträchtlichen Sammlung von Handschriftdaten beitrugen und damit viele Experimente und Entwicklungen ermöglicht haben.
Schließlich möchte ich mich bei meiner lieben Frau Antje und meinen Kindern Sophia, Jan Philipp und Niklas für deren Verständnis und deren stete Unterstützung - auch in sehr arbeitsintensiven Phasen - bedanken.
Ihnen sei diese Arbeit gewidmet.

Duisburg, im Dezember 2000

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der verwendeten Formelzeichen

$\mathbf{a}$	Kontrollpunkt einer Bézierkurve oder eines B-Splines
$\vec{B}$	Bézierkurve
b	Ausgabeverteilung eines HMM-Zustandes
C	Buchstabenfolge
c	Buchstabe, Zeichen, Graphem
d	Abstandsmaß zwischen HMM-Zuständen
d_C	Abstandsmaß zwischen Clustern von HMM-Zuständen
F	Fenstergröße
F_{BM}	Bitmap-Fenster
$\mathcal{F}$	Zeitfenster einer Trajektorie
f_i	Aktivierungsfunktion des i-ten Neurons
$\mathbf{f}$	Abtastpunkt
g	binäre Bitmap
g'_k	Ausschnitt aus g um k-ten Abtastpunkt
h_i	Hilfsfunktionen
H	Entropie
h	Histogramm
I	Transinformation
$\mathcal{I}$	Informationsgehalt
i, j, J	allgemeine Zählvariablen
K	Länge einer Schriftprobe in Abtastpunkten
k	Zeitdiskreter Index nach der Neuabtastung der Stifttrajektorie
L	Likelihood
l	allgemeine Zählvariable
$\hat{l}$	Vorgabewert für die Abtastvektorlänge
$\vec{m}$	Merkmalsvektor
M	Dimensionalität

m	(skalares) Merkmal
m_x, m_y	Mittelpunkt eines umschreibenden Rechtecks
N	Kontexttiefe
N^-	Rückweisungsschwelle für die Größennormalisierung
N_Φ^-	Rückweisungsschwelle für die Slantkorrektur
N_C	Codebuchgröße
N_c	Zeichenvorrat
N_m	Anzahl der Komponenten eines Merkmalsvektors
n	allgemeine Zählvariable
$o(\cdot)$	Soft-Max Funktion
p	Wahrscheinlichkeitsdichtefunktion
$\tilde{p}$	Stiftdruck
$Q(\cdot)$	Kullback-Leibler Distanz
$\mathcal{Q}$	Menge von Fragen
q	Frage zur Aufspaltung eines Clusters
R	Matrix der Bernstein-Operatoren
$r_{\mathbf{v}}$	Radius des Abtastpunktes $\mathbf{v}$
r	Bernstein-Operator
$\vec{S}_3$	Kubischer B-Spline
S	Folge von HMM-Zuständen
S^*	optimale HMM-Zustandsfolge
$\mathcal{S}$	Menge von HMM-Zuständen (Cluster)
s	HMM-Zustand
t	Zeit
U	Eigenwertmatrix
V	Folge von Abtastvektoren
$\vec{v}$	Abtastvektor
$\mathbf{v}_0, \mathbf{v}_1$	Anfangs- bzw. Endpunkt eines Abtastvektors
W	Wortfolge, bestehend aus Worten w
W^*	optimale Wortfolge
w	Wort
X	beobachtete Folge von Merkmalsvektoren
x_l, x_r	linke und rechte Begrenzung des umschreibenden Rechtecks
x	Kartesische x-Komponente der Stiftposition
$\vec{x}$	allgemeiner Merkmalsvektor der Beobachtungsfolge X
Y	Folge von VQ-Codewörtern
y_{BL}	Position der Basislinie
y_{KL}	Position der Kernlinie

y_o, y_u	obere und untere Begrenzung des umschreibenden Rechtecks
y_n	(diskretes) VQ-Codewort
y	Kartesische y-Komponente der Stiftposition
Z, z	Indiziert Merkmalsstrom bzw. Codebuch
$\alpha(k)$	Winkel des k-ten Abtastvektors
β	Lernparameter
γ_s	Frequentierung des HMM-Zustandes s
ΔT	Abtastintervall
η	normalisierter, kubischer B-Spline
θ	Zeilenneigungswinkel
Λ	Parametersatz des Sprachmodells
λ	Parametersatz der HMM
λ_{VQ}	Parametersatz des Vektorquantisierers
$\vec{\lambda}_i$	i-ter Codebuchvektor des Vektorquantisierers
Φ	Schriftneigungswinkel
$\phi_{\mathbf{v}}$	Winkel des Radiusvektors von **v** zur x-Achse
$\|\cdot\|_2$	Euklid'sche Norm

Verzeichnis der verwendeten Abkürzungen

BM	Bitmap
CAGD	Computer Aided Geometric Design
DCT	Diskrete Cosinus Transformation
DFT	Diskrete Fourier Transformation
dpi	Geometrische Auflösung (dots per inch)
FB	Forward-Backward (Algorithmus)
HMM	Hidden Markov Modell
HSE	Handschrifterkennung
INRIA	Institut National de Recherche en Informatique et en Automatique
JPG/JPEG	Joint Photographic Experts Group
LRTD	Left-Right-Top-Down
ME	Merkmalsextraktion
ML	Maximum Likelihood
MLP	Mutli-Layer Perzeptron
MMI	Maximale Transinformation (Maximum Mutual Information)
MPEG	Moving Picture Experts Group
NA	Neuabtastung
NN	Neuronales Netz
nN	Nächster Nachbar
OCR	Optical Character Recognition
OFR	Optical Formula Recognition
PDA	Personal Digital Assistant
PIM	Personal Information Manager
RBF	Radiale Basisfunktionen
TB	Tree-based (Clustering)
VQ	Vektorquantisierung
WD	Schreiberabhängig
WDF	Wahrscheinlichkeitsdichtefunktion
WI	Schreiberunabhängig

WTA	Winner-Takes-All
WYSIWYG	What-you-see-is-what-you-get Textverarbeitungsprogramme

Kapitel 1

Einleitung

Als Kommunikationsform und Erweiterung des menschlichen Gedächtnisses hat die geschriebene Sprache in all ihren Formen über viele Jahrhunderte hinweg die Entwicklung der Kulturen in entscheidender Weise beeinflußt. Seit der Erfindung des Buchdrucks verlagerte sich die Erstellung von Schriften zunehmend vom manuellen in den maschinellen Bereich. Dieser Prozeß wird scheinbar durch die starke Verbreitung leistungsfähiger Computer und deren Vernetzung zunehmend beschleunigt. Aufgrund dieser Entwicklungen wird nun - an der Schwelle zum dritten Jahrtausend - zunehmend die Frage nach der zukünftigen Bedeutung der Handschrift aufgeworfen.

Die skizzierten Entwicklungen mögen einerseits den Schluß nahelegen, dass die Nutzung von Handschrift als Kommunikationsmittel in Zukunft nicht mehr relevant sein wird. Andererseits, ebenfalls bedingt durch die technologischen Entwicklungen, scheint sich aber eine weitere Nutzungsform der Handschrift zu erschließen. Insbesondere die zunehmende Miniaturisierung von Computern, bei gleichzeitig zunehmender Rechenleistung, wecken den Bedarf nach fortschrittlichen Mensch-Maschine-Schnittstellen. Die Nutzung einer Tastatur erscheint bei Miniaturisierungsgraden heutiger PDA (personal digital assistants) oder Mobiltelefonen nicht mehr zweckmäßig. Darum rücken natürlichere Kommunikationsformen zwischen Mensch und Maschine stärker ins Blickfeld. Zu solchen Kommunikationsformen zählt zweifellos die gesprochene Sprache, wie auch die Handschrift. Dabei stellt sich offensichtlich weniger die Frage ob besser Sprache *oder* Handschrift verwendet werden sollte, sondern wie sich gesprochene Sprache *und* Handschrift ergänzend kombinieren lassen.

Während sich die gesprochene Sprache als Eingabemodus größerer Textmengen anbietet, scheint die Handschrift z. B. besonders geeignet für die Eingabe persönlicher Notizen.

Weiterhin stellt sich die Realisierung einer robusten Spracherkennung in Umgebungen mit starken Hintergrundgeräuschen als problematisch dar. Auch hier gilt die Handschrift als ergänzender Eingabemodus zur gesprochenen Sprache. Darüber hinaus sind einige Dokumentbestandteile, wie z. B. Skizzen oder Formeln, unter Verzicht auf Tastatur und Maus

sinnvoller Weise nur mittels eines Stiftes einzugeben.
Wie auch immer die speziellen Anwendungen für eine Handschrift-basierte Mensch-Maschine-Schnittstelle aussehen, langfristig werden die mit der Schnittstelle verbundenen Erkennungsalgorithmen drei wesentliche Forderungen erfüllen müssen:

- Die Erkennungssysteme sollten schreiberunabhängig arbeiten, oder aber im schreiberabhängigen Modus mit begrenztem Datenmaterial trainierbar sein.
- Das Erkennungssystem sollte über einen möglichst großen Wortschatz verfügen.
- Es sollten keine besonderen Bedingungen an den Schreibstil gestellt werden. Das System sollte durch den Benutzer mit seinem persönlichen Schreibstil bedienbar sein. Derzeitige verfügbare Produkte, mit einem fest definierten, speziellen Zeichensatz, können lediglich als Übergangslösung angesehen werden.

Wenngleich das äußerst facettenreiche Problem der Handschrifterkennung nicht in allen Details und mit absoluter Erkennungssicherheit gelöst werden kann, so sollen doch im Rahmen dieser Arbeit einige Lösungsvorschläge erarbeitet und präsentiert werden, die in die Richtung der oben genannten Anforderungen nach robuster und schreibstilunabhängiger Erkennung zielen.
Die für den Aufbau der Systeme akquirierten Trainings- und Testbeispiele orientieren sich konsequent an diesem Ziel. Hierbei werden die Testpersonen vor der Datenaufnahme nicht instruiert in besonderer Weise zu schreiben. Es findet keine Unterscheidung zwischen gedruckter Handschrift, vollständig verbundener (kursiver) Schrift oder diversen Mischtypen statt.
Wie in Abb. 1.1 dargestellt, bildet die Verarbeitungskette bestehend aus Vorverarbeitung, Merkmalsextraktion, Modellebene und Erkennung die Grundstruktur des Gesamtsystems. Die vorliegende Arbeit mit der Aufteilung der Kapitel bildet diese Struktur nach.
In Kapitel 2 werden somit zunächst die wesentlichen Vorverarbeitungsmethoden erklärt. Dies umfasst im wesentlichen die Glättung der Rohdaten, sowie die Normalisierung der Schrift. Die Normalisierung wird durchgeführt, um vorab die Variabilität der Handschrift bezüglich der Schrift- und Zeilenneigung, sowie der Schriftgröße einzuschränken.
In Kapitel 3 werden einige Verfahren zur Merkmalsentnahme untersucht. Um später die Modellparameter zuverlässig schätzen zu können, ist es das Ziel der Merkmalsextraktion, aus der Handschrift mit möglichst wenig Parametern die relevante Information zu extrahieren.
Kapitel 4 und 5 betreffen die Modellierungsaspekte. Während in Kapitel 4 die Frage nach der optimalen Modellierungsform beantwortet wird, werden in Kapitel 5 Vorschläge zur Strukturverknüpfung kontextabhängiger Modelle präsentiert. Bei kontextabhängigen Modellen handelt es sich um multiple Modelle für ein Zeichen. In Abhängigkeit der möglichen Kombinationen der Nachbargrapheme entsteht eine große Anzahl solcher kontextabhängigen Mo-

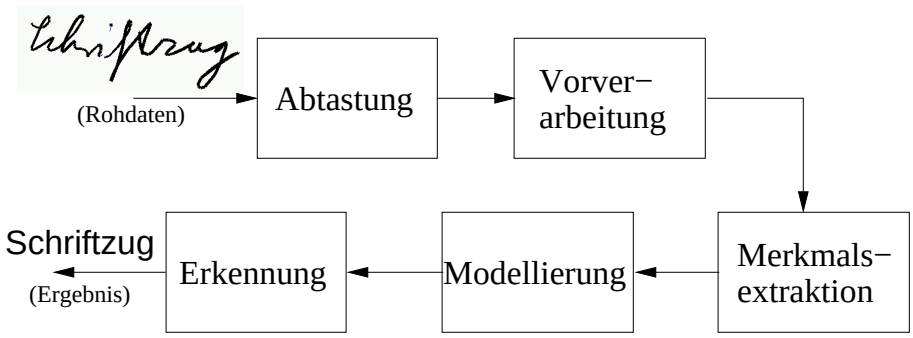


Abbildung 1.1: Systemübersicht

delle. Die Trainierbarkeit dieser Modelle wird schließlich durch die geschickte Verknüpfung von Modellparametern ermöglicht.

Bei verbundener, oder zumindest stückweise verbundener Handschrift, ist für die Erkennung die Angabe der erlaubten Wörter und der darin enthaltenen Buchstabenfolge in Form eines Lexikons notwendig. Mit dem in Kapitel 6 vorgestellten Ansatz ist es möglich diese Restriktion weiter zu reduzieren. Das feste Lexikon und die darin starr vorgegebenen Buchstabenfolgen werden durch eine statistische Beschreibung ersetzt. Die Ausdehnung der statistischen Beschreibung auf einen hinreichend großen Bereich erfordert dabei allerdings spezielle Dekodierungsalgorithmen.

Die Kapitel 2 und 3 gelten als allgemeingültig und sind als weitgehend unabhängig von der Zielanwendung und von der Funktionsweise des verwendeten Erkennungsansatzes zu verstehen. Kapitel 4 und insbesondere Kapitel 5 beziehen sich auf die Hidden Markov Modell basierte Texterkennung mit großem Vokabular, wobei in Kapitel 6 die Restriktion des festen Lexikons weiter reduziert wird. Kapitel 7 schließlich zeigt ein weiteres interessantes Anwendungsszenario für die Online-Handschrifterkennung. Hierbei handelt es sich um die Erkennung und Verarbeitung handgeschriebener Formeln. Die in Kapitel 7 beschriebenen Verfahren setzen auf den zuvor beschriebenen Algorithmen auf. Darüber hinaus bedarf es für die Analyse der zweidimensionalen Struktur von Formeln weiterer Schritte, die in diesem Kapitel beschrieben werden. Das System wird schließlich zu einem interaktiven, handschriftlichen Formeleditor erweitert.

Die Ergebnisse werden gesondert zu den einzelnen Kapiteln diskutiert. Eine Zusammenfassung schließt die Arbeit ab.

Kapitel 2

Vorverarbeitung

Unter technischen Aspekten, handelt es sich bei der Handschrift um ein hoch-komplexes Signal zur Informationsübertragung. Jedes handgeschriebene Wort, oder auch nur ein handgeschriebener Buchstabe sind Unikate. Die genaue manuelle Reproduktion, selbst der eigenen Schrift, scheint ausgeschlossen. Dies fällt um so mehr auf, je eingehender die Handschrift mit maschinellen Methoden bearbeitet wird. Die Faktoren, die schließlich das Gesamtergebnis beeinflussen, sind dabei außerordentlich vielfältig.
Die größten Unterschiede sind sicherlich bei variierenden Schreibern auszumachen. Jede Handschrift verfügt über gewisse Eigenschaften, die sie unverwechselbar machen.
Doch auch unter der Voraussetzung, dass die von einem Handschrifterkennungssystem erfassten Handschriftdaten von nur einem Schreiber produziert werden, ergeben sich zahlreiche Randbedingungen, die auf das Resultat einwirken. Dies können sein:

- Der Zeitpunkt der Produktion. Insbesondere bei jüngeren Menschen kann sich das Schriftbild über einen längeren Zeitraum stark verändern.
- Die persönliche Verfassung des Schreibers. Eine verletzte Schreibhand z. B. kann großen Einfluß auf die Handschrift haben.
- Der Betreff und die Zielperson. Das Schriftbild einer für eigene Zwecke gedachten schnellen Notiz weicht möglicherweise ab von dem Schriftbild in handgeschriebenen Nachrichten für andere.
- Schreibgerät und Unterlage beeinflussen das Schriftbild.
- Der Kontext. Als Kontext läßt sich sowohl die weiter als auch enger gefasste Umgebung der aktuellen Schreibposition definieren. Die weitere Umgebung, und damit verbunden auch die Platzverhältnisse, können sich auf die Schrift auswirken. Im engeren Sinne kann eine Kontextabhängigkeit auch durch benachbarte Buchstaben bedingt sein.

- Datenerfassungsgerät. Als technische Komponente beeinflußt das benutzte Erfassungsgerät als unmittelbare Schnittstelle zwischen Mensch und Maschine wesentlich die Qualität der erfassten und zu verarbeitenden Daten (Abtastverfahren, Abtastrate , Auflösung).

Auf eine automatische Schrifterkennung wirken sich Veränderungen der genannten Einflußgrößen natürlich störend aus. Ein 'a' sollte stets als ein solches erkannt werden - und dies möglichst robust gegenüber äußeren Einflüssen. Der Erkennungsalgorithmus sollte so unabhängig wie möglich von den genannten Randbedingungen funktionieren. Die Eliminierung, oder zumindest die Minimierung irrelevanter Eigenschaften ist die Aufgabe der Vorverarbeitung. Die Vorverarbeitung betrifft in diesem Zusammenhang die in den folgenden Abschnitten beschriebene Neuabtastung und die Neigungs- und Größen-Normalisierung der Schrift.

2.1 Neuabtastung

Mit der Online-Erfassung von Handschriftdaten und der damit gegebenen zeitlichen Abfolge der Abtastvektoren ergeben sich einerseits Vorteile, wie beispielsweise die Unterstützung der Zeichensegmentierung durch den Stiftdruck, andererseits wird mit der Online-Abtastung auch unerwünschte und irrelevante Information erfasst.
Durch die Verwendung von WACOM-Boards für die Datenerfassung sind dies in expliziter Form zum einen die Stiftführung bei negativem Stiftdruck - also bei abgehobener Mine. Bei dem Verlauf dieser virtuellen Segmente (obere Zeile in Abb. 2.2) kann davon ausgegangen werden, dass diese keinen Beitrag für eine Unterscheidung verschiedener Buchstaben liefern.
Zum anderen ist durch die zeitlich äquidistante Abtastung der Rohdaten implizit die Schreibgeschwindigkeit in den Rohdaten enthalten. Die zeitlich konstante Abtastrate resultiert bei zwei optisch identischen Kurvenverläufen die mit unterschiedlicher Stiftgeschwindigkeit erzeugt wurden in deutlich abweichenden Abtastsequenzen.
Um diese unerwünschten Effekte auszuschalten, bietet sich eine Neuabtastung (NA) der Rohdaten an [Wey96, Kas95]. Die Neuabtastung überführt die Folge von Abtastpunkten, die zu $n \cdot \Delta T$ zeitlich äquidistant abgetastet wurden in eine Folge räumlich äquidistanter Abtastpunkte:

$$\vec{v}(n \cdot \Delta T) = (x_{Anf}, y_{Anf}, x_{End}, y_{End}, \tilde{p})(n \cdot \Delta T) \xrightarrow{NA} (x_{Anf}, y_{Anf}, x_{End}, y_{End}, \tilde{p})(k) = \vec{v}(k) \tag{2.1}$$

Die Abtastvektoren $v(n \cdot \Delta T)$, die durch die abgetasteten Anfangs- und Endpunkte in x- und y-Richtung, wie auch durch den abgetasteten Stiftdruck $\tilde{p}$ gegeben sind, werden mit

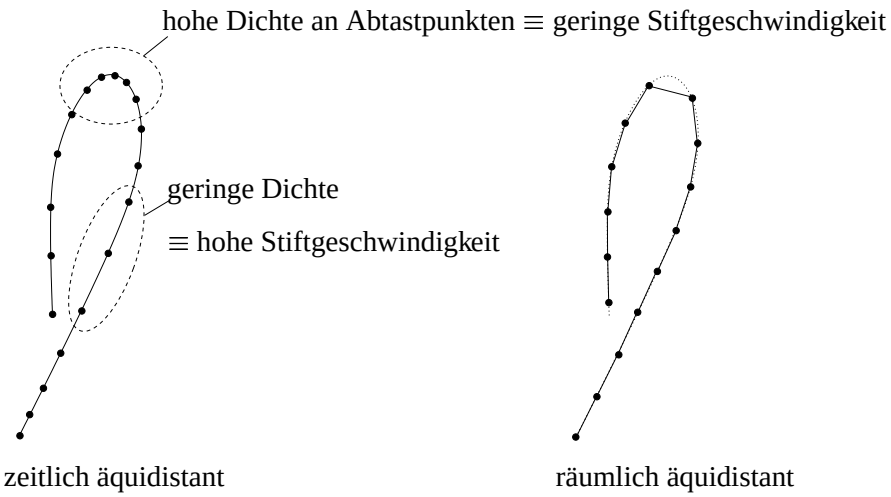


Abbildung 2.1: Neuabtastung - prinzipielle Wirkungsweise

der Neuabtastung in der Form zusammengefasst oder ggf. aufgeteilt, dass der Abstand zwischen Anfangs- und Endpunkten $\sqrt{(x_{End}(k) - x_{Anf}(k))^2 + (y_{End}(k) - y_{Anf}(k))^2}$ der Vektoren $\vec{v}(k)$ einem vorzugebenden, konstanten Wert $\hat{l}$ entspricht. Abb. 2.1 verdeutlicht diese Vorgehensweise. Abb. 2.2 zeigt einige konkrete Beispiele, die mit verschiedenen Abtastvektorlängen $\hat{l} = 2, 5, 10, 20$ erzeugt wurden, zusammen mit der zugrunde liegenden Originalsequenz. Die obere Zeile in Abb. 2.2 gibt die auf oben beschriebene Weise neuabgetasteten Vektorsequenzen wieder. Die dünnen Linien zeigen die ebenfalls erfassten Stiftbewegungen zwischen den Segmenten bei abgehobener Mine.

Zum einen zeigt sich in Abb. 2.2 der z. T. chaotische Verlauf der virtuellen Segmente (bei abgehobenem Stift), zum anderen zeigt sich aber auch, dass bei steigender Abtastvektorlänge die Formeigenschaften der Zeichen doch stark verfälscht werden. Dies gilt um so mehr bei insgesamt eher kleinen Zeichen. Abhilfe für das erste Problem schafft eine lineare Interpolation der virtuellen Segmente mit Abtastvektoren der konstanten Länge $\hat{l}$.

Zur Lösung des zweiten Problems werden die realen Teilsequenzen (bei aufgesetztem Stift) zunächst nach lokalen Extremwerten in x- und y-Richtung durchsucht. Diese Extremwerte werden dann verwendet, um eine Unterteilung der realen Teilsequenzen in Fragmente vorzunehmen. Schließlich wird die Neuabtastung so durchgeführt, dass die verwendete Abtastvektorlänge für das aktuelle Fragment optimiert wird. D. h. die Anzahl der Abtastvektoren in einem Fragment, und damit deren Länge ist so zu wählen, dass sie der geforderten Solllänge $\hat{l}$ möglichst weit angenähert wird. Die untere Zeile in Abb. 2.2 zeigt dazu einige Beispiele, ebenfalls bei variierter Abtastvektorlänge. Es zeigt sich klar, dass die chaotisch geformten virtuellen Segmente linearisiert wurden, wie auch die Formerhaltung bei längeren Abtastvektoren verbessert wurde. So läßt sich nicht nur die Konsistenz der Daten verbessern, insgesamt wird mit der Neuabtastung auch eine Reduktion der Datenmenge um das zwei- bis dreifache erreicht.

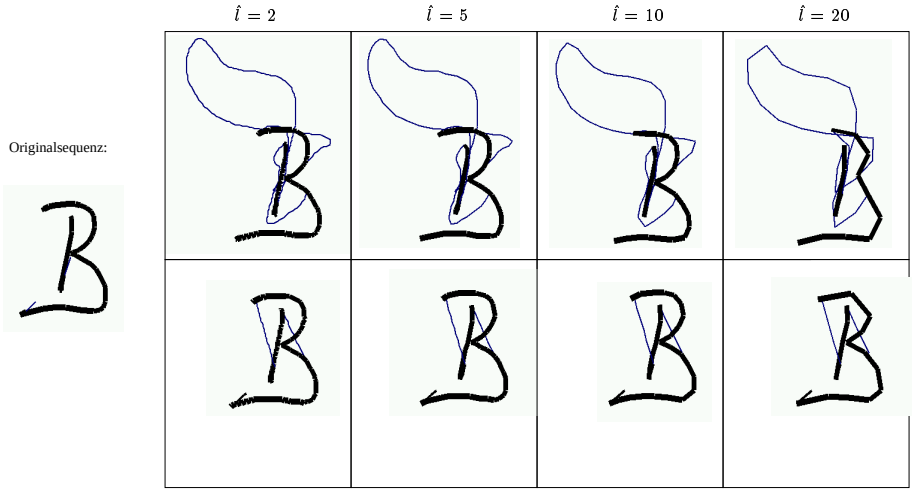


Abbildung 2.2: Neuabtastung mit verschiedenen Vektorlängen

2.2 Normalisierung

Neben der Abgleichung der Stiftgeschwindigkeit und der Linearisierung virtueller Segmente, sollten vor der eigentlichen Modellierung weitere Schritte zur Steigerung der Datenkonsistenz unternommen werden.
Für eine robuste Handschrifterkennung sind aufgrund der hohen Schriftvariabilität zwischen verschiedenen Schreibern große Mengen an Trainingsbeispielen erforderlich um möglichst alle Schrifttypen zu erfassen. Abgesehen von der Stiftgeschwindigkeit läßt sich die Handschrift unter Modellierungsaspekten anhand von vier wesentlichen Merkmalen charakterisieren:

- Schriftgröße,
- Schriftneigung (Slant),
- Zeilenneigung (Skew),
- Die Formeigenschaften der einzelnen Zeichen (Font).

Die Aufgabe der Handschriftnormalisierung ist die Reduktion dieser Freiheitsgrade, um auch mit einer relativ kleinen Trainingsdatenbasis und damit mit weniger komplexen Modellen eine robuste Erkennung zu realisieren. Während die Fontvarianz erst auf der Modellebene aufgelöst werden kann, ist es bereits in frühen Vorverarbeitungsschritten möglich, die Handschrift bezüglich Größe und Neigungswinkel zu normalisieren.
Die Normalisierung der Handschrift ist speziell in schreiberunabhängigen Systemen von Bedeutung. In schreiberabhängigen Systemen werden schreiberspezifische Charakteristika wie Schrifthöhe oder Slant-Winkel auf der Modellebene mittrainiert. Eine Normalisierung ist in

schreiberabhängigen Systemen daher nur erforderlich, wenn vorab bereits klar ist, dass die Testbedingungen von den Trainingsbedingungen abweichen.

2.2.1 Zeilenneigung - Skew

Eine in der Vorverarbeitung korrigierbare Abweichung stellt im allgemeinen auch die Zeilenneigung dar. Zur Korrektur einer Zeilenneigung finden sich in der Literatur zwei prinzipielle Ansätze. Der erste Ansatz basiert auf der Berechnung linearer Regressionsgeraden [Sch95]. Die berechneten Regressionsgeraden approximieren obere und untere Wendepunkte der Trajektorie. Die Steigungen der ermittelten Geraden sind ein Maß für den Winkel der Zeilenneigung. Der zweite Ansatz stützt sich auf die Analyse von Richtungshistogrammen [Sun97] um den Zeilenneigungswinkel zu finden. Ist der Zeilenneigungswinkel θ_0 gefunden, läßt sich die Vektorsequenz durch eine Rotation um den Winkel $-\theta_0$ korrigieren.
Bei einem Abstand $r_{\mathbf{v}}(k)$ der Abtastpunkte $\mathbf{v}(k) = (x, y)(k)$ zu einem frei wählbaren Referenzpunkt und einem Winkel $\varphi_{\mathbf{v}}(k)$ des Radiusvektors zur x-Achse ergibt sich die rotierte Abtastsequenz $\mathbf{v}'(k) = (x'(k), y'(k)) = r_{\mathbf{v}}(k)[\cos(\varphi_{\mathbf{v}}(k) - \theta_0), \sin(\varphi_{\mathbf{v}}(k) - \theta_0)]$. Sinnvoll ist es für die weitere Verarbeitung, den Rotationsradius einzuschränken. Als Referenzpunkt kann darum beispielsweise der Anfangspunkt des 0-ten Abtastvektors $\mathbf{v}_0(0)$ gewählt werden, um den sämtliche Vektoren $\vec{v}(k)$ gedreht werden.
Eine interessante Alternative zu den oben erwähnten Ansätzen bietet sich für die Normalisierung von Offline-Daten mit der Untersuchung von Integralprojektionen [Cot97]. Übertragen auf Online erfasste Handschriftdaten, läßt sich eine Projektion aus den Häufigkeiten der Abtastpunkte in den einzelnen Pixelreihen ermitteln. Betrachtet wird dazu ein hinreichend großer Bereich zwischen y' und y'' in y-Richtung der für die auszuführenden Rotationen genügend Raum bietet. Liegen in der i-ten Pixelreihe n_i Abtastpunkte, so läßt sich die Projektion, bzw. relative Häufigkeit

$$h(y) = \frac{n_y}{\sum_{i=y'}^{y''} n_i} \tag{2.2}$$

für die Pixelzeile y angeben.
Abb. 2.3 zeigt an einem Beispiel den Zusammenhang zwischen dem Zeilenneigungswinkel und den entsprechenden y-Projektionen. Horizontale Ausrichtungen spiegeln sich in Verteilungen mit geringer Varianz und klar erkennbaren Maxima wieder. Bei schief verlaufenden Zeilen sind die Verteilungen der Abtastpunkte deutlich flacher. Als Bewertungsmaß für die unterschiedlichen Verteilungen drängt sich damit die Entropie

$$H = -\sum_{i=y'}^{y''} h(i) \log h(i) \tag{2.3}$$

der Verteilung $h(i)$ auf.

Um schließlich die Zeilenneigung einer unbekannten Schriftprobe zu ermitteln, ist die Vek-

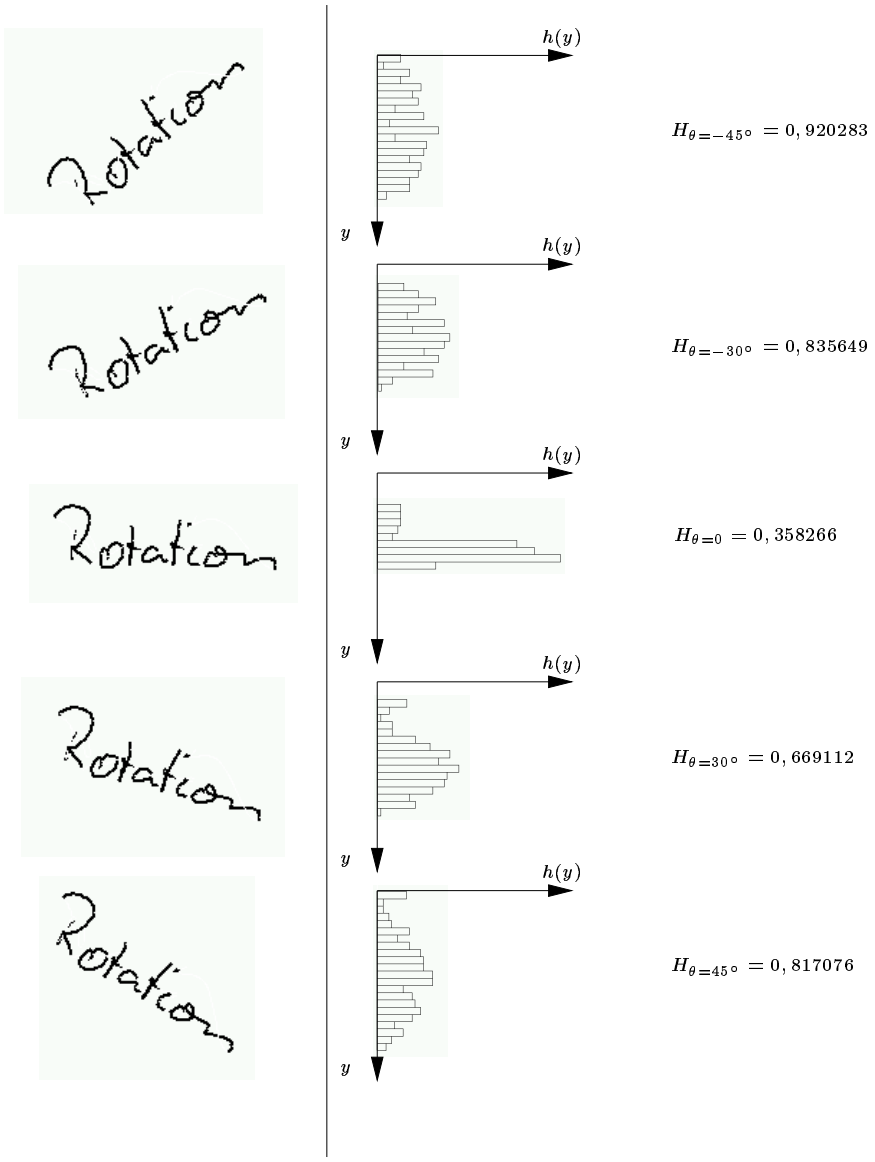


Abbildung 2.3: Zeilenrotation

torfolge in einem bestimmten Intervall jeweils um einen kleinen Betrag $\Delta\theta$ zu drehen. Zu jedem Drehwinkel wird die Projektion ermittelt und die zugehörige Entropie. Abb. 2.4 zeigt dazu das Beispiel einer schiefen Schriftprobe und den Entropieverlauf der y-Projektion in Abhängigkeit des Rotationswinkels. Der Neigungswinkel von ca. $-18°$ läßt sich klar am Minimum des Entropieverlaufes ablesen. Die abschließende Zeilenneigungskorrektur kann durch eine Rotation der Originalsequenz um $+18°$ vorgenommen werden. Bei einer hinreichend großen Anzahl von Abtastpunkten erweist sich dieses Verfahren als recht präzise. Es läßt sich dann eine Genauigkeit von $\pm 1°$ erreichen.

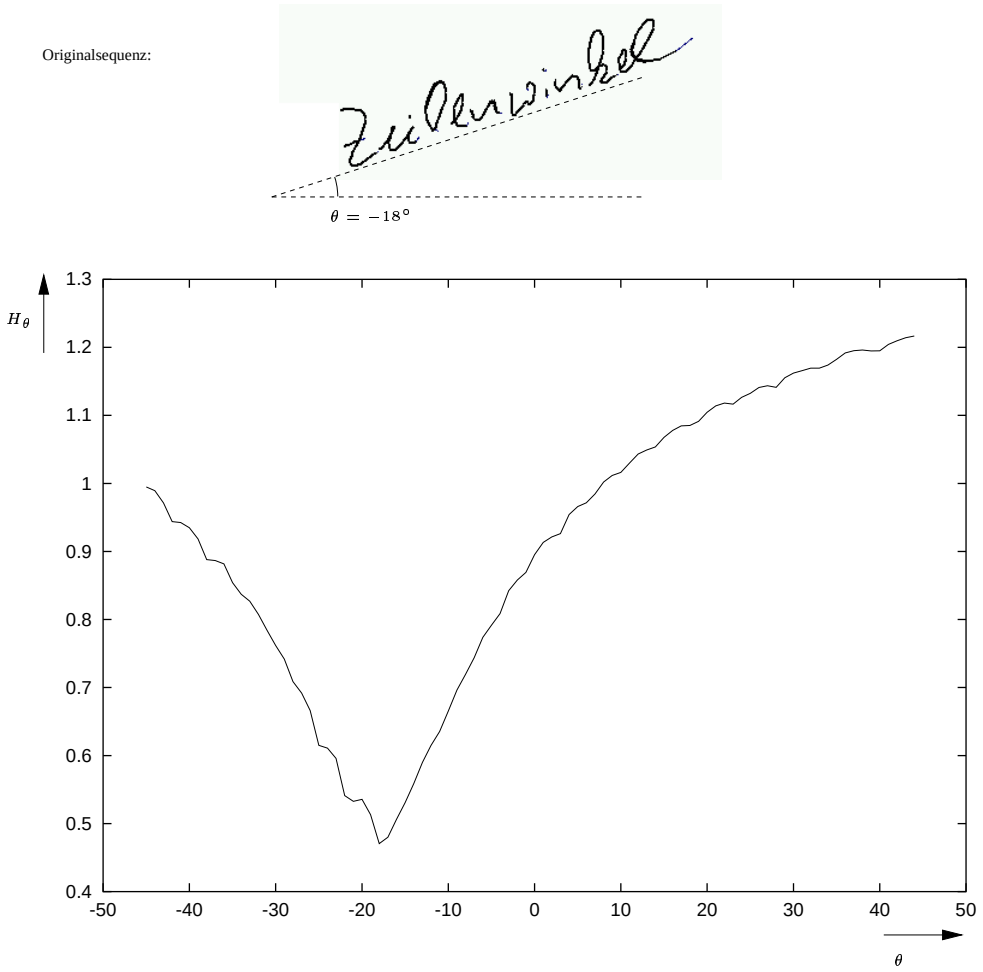


Abbildung 2.4: Entropie der Integralprojektion als Funktion des Zeilenneigungswinkels

2.2.2 Schriftneigung - Slant

Für die Normalisierung der Schriftneigung (Online und Offline) wird bisher fast ausschließlich auf die Analyse von Richtungshistogrammen gesetzt [Sun97]. In Anlehnung daran wird in [Sch95] ein Verfahren zur Slant-Minimierung beschrieben, welches eine mittlere Segmentorientierung verwendet. Die Ergebnisse in [Sch95], sowie eigene Untersuchungen zeigen jedoch, dass eine mittlere Orientierung, oder auch das Maximum der Orientierungsverteilung keine stabile Slant-Abschätzung erlaubt. Ein Grund dafür ist sicherlich, dass die verwendeten Vektorsequenzen, bedingt durch ihre teilweise begrenzten Längen, keine aussagekräftige Stichprobe für das Aufstellen von Richtungshistogrammen darstellen. Auf eine Normalisierung der Schriftneigung muß in der Vorverarbeitung daher oftmals verzichtet werden.

Betrachtet man das Problem der Schriftneigung näher, läßt sich wiederum die Frage stellen, was überhaupt ein eindeutiges Merkmal für aufrechte oder geneigte Schrift darstellt. Dies sind zweifellos die näherungsweise vertikalen Stiftbewegungen bei aufgesetzter Mine (pos. Stiftdruck). Diese Bewegungen entziehen sich allerdings einer direkten Analyse durch Rich-

tungshistogramme.

Eine indirekte Methode wäre wiederum die Projektion der Vektorsequenz. Nun erfolgt die Projektion jedoch auf die x-Achse. Schrift mit exakt vertikaler Ausrichtung der einzelnen Segmente müßte dabei durch Verteilungen auffallen, die besonders bei auftretenden Oberlängen zu hohen Peaks in den Projektionen führen. Geneigte Zeichen hingegen, die teilweise noch benachbarte Buchstaben überdecken, sollten im Gegensatz dazu eher verschmierte Maxima in den Projektionen ausbilden.

Diese Überlegungen werden anhand von Abb. 2.5 bekräftigt. Dort ist der Entropieverlauf in

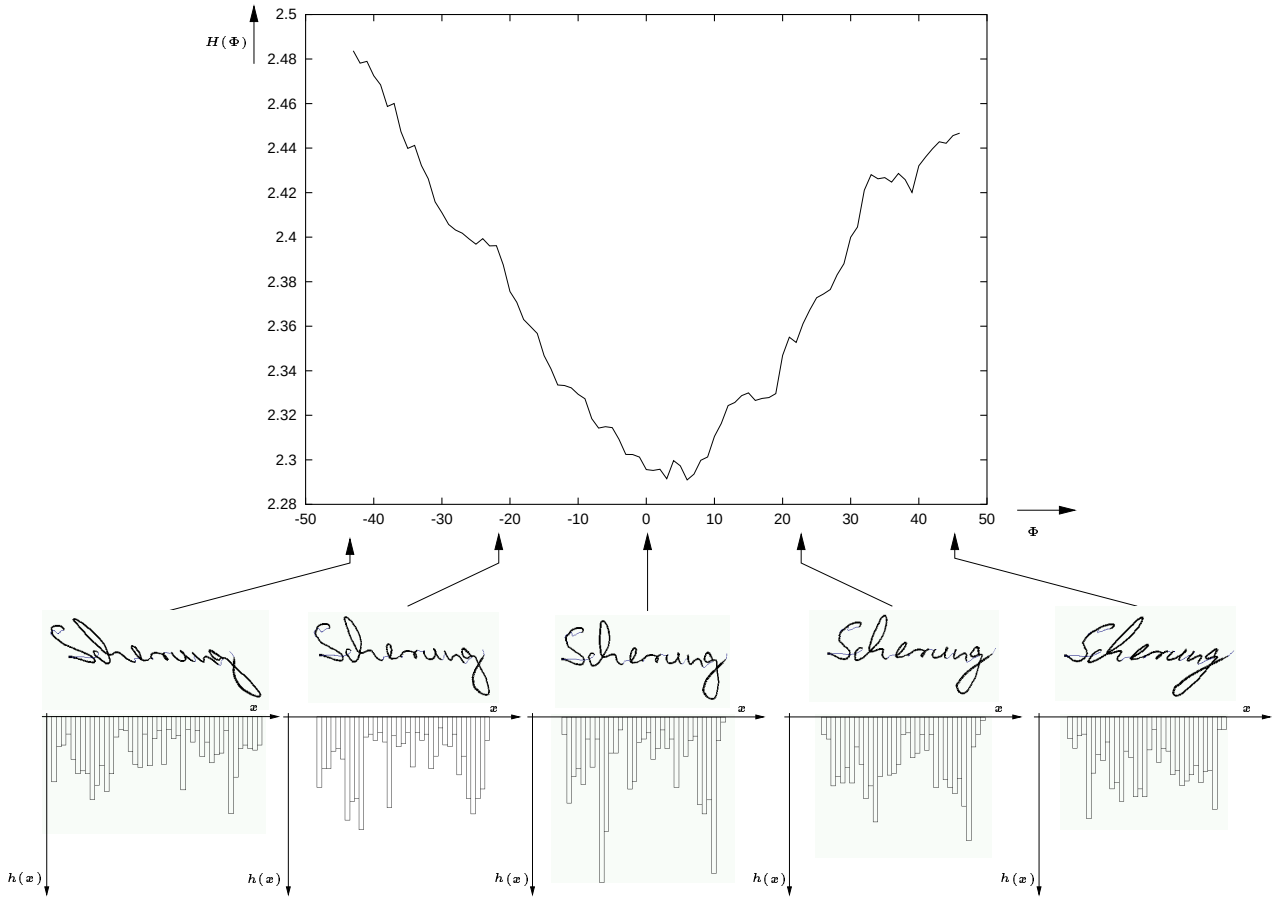


Abbildung 2.5: Entropieverlauf als Funktion des Scherungswinkels

Abhängigkeit des Schriftneigungswinkels gezeigt, wie auch das Schriftbild mit zugehörigen x-Projektionen für einige ausgewählte Neigungswinkel. Es zeigt sich ein Entropieminimum bei vollständig aufgerichteter Schrift. Wenngleich das Minimum in absoluten Werten nicht so deutlich von den Werten bei extremer Neigung abweicht, wie es bei der Zeilenrotation der Fall ist, scheint der Entropieverlauf ein geeignetes Maß für die Erkennung des Schriftneigungswinkels darzustellen.

Ist der Schriftneigungswinkel nun bekannt, so kann durch eine Scherung um diesen Winkel die Vektorsequenz $\vec{v}(k)$ zu $\vec{v}'(k)$ korrigiert werden. Die Scherung wird entsprechend den in

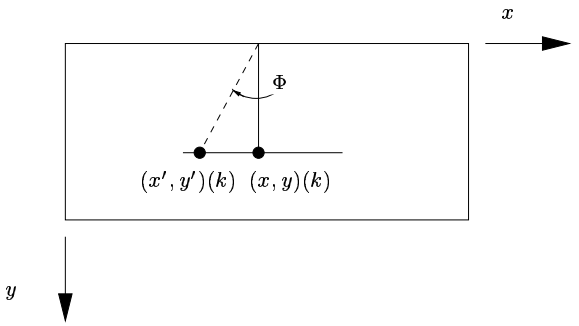


Abbildung 2.6: Vorzeichenkonvention für die Schriftscherung

Abb. 2.6 skizzierten Größen und Vorzeichen durchgeführt.
Von der Scherung bleiben die y-Komponenten der Abtastpunkte unberührt, d. h. es gilt $y'(k) = y(k)$. Die x-Komponenten der Abtastpunkte $x(k)$ werden um $-y(k) \cdot \tan(\Phi)$ korrigiert, sodass sich ein $x'(k) = x(k) - y(k)\tan(\Phi)$ ergibt. Da die Scherung in x-Richtung von der vertikalen Position y abhängt, sollte auch hier statt der x-Achse eine Referenzachse in der nahen Umgebung der Schrift gewählt werden.
In einigen Grenzfällen erweist es sich als schwierig einen exakten Neigungswinkel anzugeben. Dies kann der Fall sein, wenn die Schriftneigung zwischen verschiedenen Wörtern einer Wortfolge, oder sogar innerhalb eines Wortes variiert, oder aber bei sehr kurzen Worten mit entsprechend wenig Abtastpunkten. In solchen Fällen läßt sich beobachten, dass sich weit auseinanderliegende, nahezu gleichwertige Entropieminima einstellen (jeweils untere Kurven in Abb. 2.7). In den in Abb. 2.7 gezeigten Beispielen würde ein solches Nebenminimum sogar zu einer falschen Winkelschätzung führen.
Bei näherer Ursachenforschung fällt dabei auf, dass sich kritische Nebenminima dann bilden, wenn möglichst viele obere Wendepunkte in vertikaler Linie mit unteren Wendepunkten liegen. Dies wiederum liegt darin begründet, dass sich an den Wendepunkten aufgrund der zeitlich äquidistanten Abtastung die Abtastpunkte überdurchschnittlich stark kumulieren (geringere Stiftgeschwindigkeit). Die vertikale Überlagerung oberer und unterer Wendepunkte stellt jedoch nicht zwangsweise den optimalem Fall vollständig aufgerichteter Buchstaben dar. Es ist dabei sogar möglich, dass sich obere Wendepunkte des einen Buchstaben über untere Wendepunkte eines benachbarten Buchstabens schieben. Die Folge sind natürlich unerwünschte starke Schriftneigungen, die u. U. noch drastischer ausfallen als bei den zugrunde liegenden Originaldaten.
Abhilfe für dieses Problem sollte eine vorab durchgeführte Neuabtastung schaffen. Die jeweils oberen Kurven der beiden Beispiele in Abb. 2.7 wurden auf diese Weise erzeugt. Sie scheinen dies zu bestätigen. Kritische Nebenminima treten dort deutlich weniger auf. Darüber hinaus sind die Kurvenverläufe $H(\Phi)$ bei den neuabgetasteten Daten glatter als bei den Rohdaten.

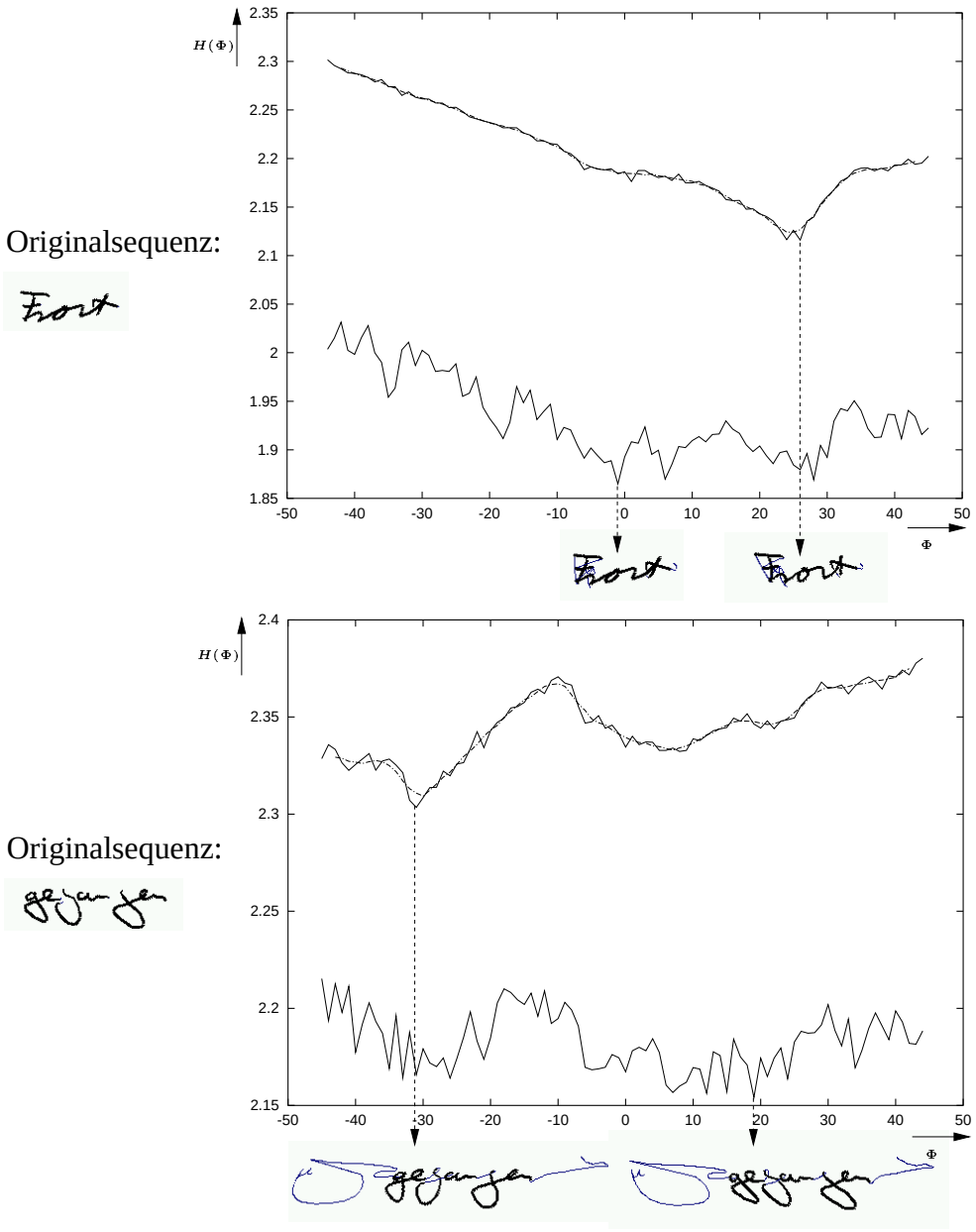


Abbildung 2.7: Beispiele für nicht eindeutig schätzbare Slant-Winkel

Einerseits sollte also die Neuabtastung nach der Normalisierung erfolgen um die Konstanz der Abtastvektoren nicht durch geometrische Transformationen zu stören, andererseits scheinen sich genau dann bessere Ergebnisse einzustellen, wenn die Normalisierung auf neuabgetasteten Daten durchgeführt wird. Als Lösung für dieses Dilemma kann daher vorgeschlagen werden, in einem ersten Schritt die Rohdaten mit sehr kleiner Vektorlänge abzutasten. Damit wird eine homogenisierte Abtastdichte erreicht, ohne signifikante Verluste hinnehmen zu müssen. Anschließend stellt der zweite Schritt die eigentliche Normalisierung dar, während mit dem dritten Schritt eine letzte Neuabtastung zur Datenreduktion und Linearisierung virtueller Segmente durchgeführt wird. So lassen sich gute Ergebnisse bei der Normalisierung mit dem Ziel konstanter Abtastvektorlängen kombinieren.

Zusätzlich zu der vorausgehenden Neuabtastung kann die Entropie $H(\Phi)$ mittels Bildung eines gleitenden Durchschnitts geglättet werden, was die Minimumsuche im Entropieverlauf noch robuster gestaltet (strichpunktierte Kurven in Abb. 2.7). Die Störung durch nahe benachbarte Minima (wie im oberen Beispiel in Abb 2.7) ist damit minimiert.

2.2.3 Größennormalisierung

Ein weiterer Freiheitsgrad, der mit der Normalisierung eingeschränkt werden sollte, ist die Schriftgröße. Für eine einheitliche Skalierung der Schriftgröße lassen sich die Abmessungen unterschiedlicher Bereiche der Schrift heranziehen. Diese Bereiche werden allgemein durch die vier in Abb. 2.8 eingezeichneten Linien abgegrenzt.

Ein Bereich, den alle Buchstaben, Wörter und Wortfolgen gemeinsam haben, ist der Bereich

Abbildung 2.8: Bereichsunterteilung der Handschrift

zwischen Basislinie und Kernlinie, die sog. Kernhöhe. Eine Approximation dieser beiden Linien und eine anschließende Skalierung der gesamten Vektorfolge auf eine vorgegebene Kernhöhe ist daher anzustreben.

Insbesondere bei der Online-Handschrifterkennung bietet sich die Nutzung der oberen und unteren Wendepunkte zur Approximation der Basis- bzw. Kernlinie an. Nachdem davon ausgegangen werden kann, dass keine Zeilenneigung mehr vorliegt, ließen sich die betreffenden Linien durch horizontale Regressionsgeraden durch die Wendepunkte annähern.

Interessant ist in diesem Zusammenhang ein Blick auf die Häufigkeiten mit denen Wörter mit Ober- oder Unterlängen auftauchen. Dazu wurde ein Text mit 26 Mio. Wörtern ausgewertet. Das Ergebnis ist, dass lediglich 1,8 Mio. Wörter (7 %) weder Ober- noch Unterlängen aufweisen. 7,2 Mio. Wörter (27,6 %) verfügen über Unterlängen. 23,5 Mio. Wörter (90,2 %) enthalten Oberlängen [1]. Diese Untersuchung belegt, dass als einzig zuverlässiges Referenzmaß zur Größennormalisierung die Kernhöhe anzusehen ist.

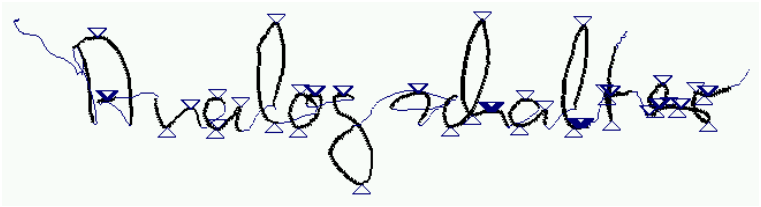
Eine einfache Regression durch alle oberen bzw. unteren Wendepunkte allein kann damit nicht zum Erfolg führen. Als Ergebnis erhielte man bei Einzelworten in 27,6 % (90,2 %) der Fälle Schätzungen für Basislinien (Kernlinien), die zwischen der echten Basislinie (Kernlinie) und der Grundlinie (Oberlinie) verlaufen. Bei Wortfolgen würden sich diese Effekte

[1] Die Angaben für Ober- oder Unterlängen sind nicht exklusiv, d. h. die gezählten Wörter mit Oberlängen können durchaus auch Unterlängen enthalten und umgekehrt.

noch verstärken.

Geht man hingegen weiterhin von der Annahme horizontaler Basis- und Kernlinien aus, so läßt sich wiederum ein Projektionsansatz versuchen. Bei einer getrennten y-Projektion von oberen und unteren Wendepunkten, kann auf die so entstandenen (eindimensionalen) Cluster ein modifizierter k-means-Algorithmus angewendet werden (Abb. 2.9).

Für die Initialisierung werden zunächst alle Wendepunkte einer Gruppe (obere oder untere)



initiale obere und untere Wendepunkte (nach Slant-Korrektur)

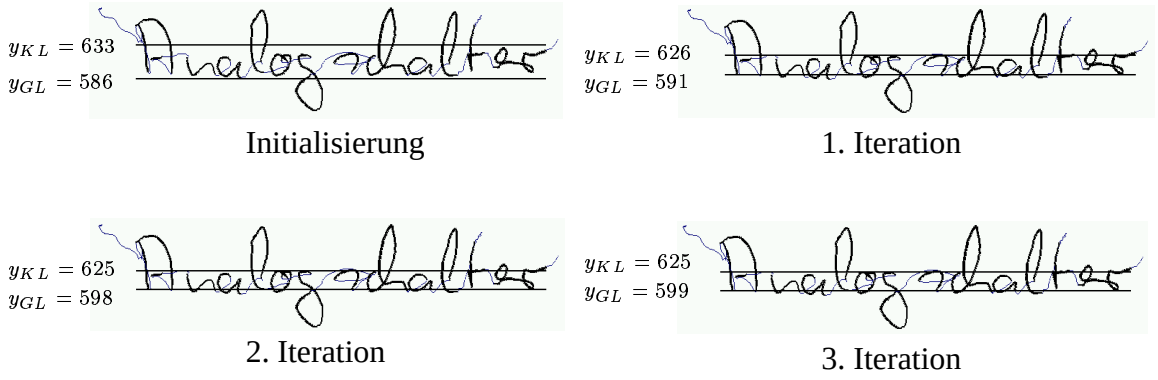


Abbildung 2.9: Iterative Bereichsdetektion

in jeweils einem Cluster zusammengefasst. Damit lassen sich der obere und untere Erwartungswert bilden. Die Positionen dieser Erwartungswerte auf der y-Achse entsprechen somit der geschätzten Kern- bzw. Basislinie. Über ein anzugebendes Abstandsmaß werden aus diesen Clustern alle Wendepunkte entfernt, die dieses Abstandsmaß übersteigen. Mit den reduzierten Clustern werden erneut die Erwartungswerte berechnet und wiederum entfernte Wendepunkte ausgeschlossen. In iterativer Weise werden so die Basis- und Kernlinien schrittweise angenähert. Der Algorithmus wird abgebrochen, wenn sich die Cluster nicht weiter verschieben, d .h. wenn keine weiteren Wendepunkte ausgeschlossen werden. Somit lassen sich kompakte Cluster bilden. Im allgemeinen wird mit diesem Verfahren eine sehr gute Konvergenz erzielt. Mehr als drei Iterationen werden selten benötigt.

2.3 Ergebnisse

Für eine Evaluierung der Verfahren ist es zunächst erforderlich die Testbedingungen zu spezifizieren. Wie bereits erwähnt, kommt der Normalisierung bei schreiberunabhängigen Systemen eine besondere Relevanz zu. Die Testmenge sollte daher Proben mehrerer Schreiber beinhalten. Die Ermittlung der Parameter, wie Neigungswinkel und Kernhöhe ist besonders dann problematisch, wenn eine Probe aus wenig Abtastvektoren besteht. Der Schwerpunkt der Tests liegt somit auf der Normalisierung von Einzelworten. Da diese häufig aus wenigen Buchstaben bestehen und damit eine sinnvolle Normalisierung schwer realisierbar ist, muß darüber hinaus auch ein optimaler Grenzwert für eine Rückweisung gefunden werden.
Ausgehend von diesen Anforderungen, wurde aus dem schreiberunabhängigen Test-Set, bestehend aus insgesamt 4000 Einzelworten per Zufallsauswahl eine Teilmenge von 413 Einzelworten zusammengestellt. Um kostspielige Tests mit dem Gesamtsystem für verschiedene Parameterkombinationen zu vermeiden, wird die Vorverarbeitung zunächst weitgehend entkoppelt betrachtet.
Für die 413 Proben wurden darum die Basis- und die Kernlinien, sowie der Schriftneigungswinkel individuell von Hand ermittelt [Haf99]. Das Set-up für die Vorverarbeitung wird hinsichtlich der mittleren linearen Abweichung zu den manuell bestimmten Referenzwerten optimiert. Im Falle der Schriftneigung wird die Abweichung in Grad angegeben (Tabelle 2.1). Bei der Schätzung der Basis- und Kernlinie erfolgt die Angabe in Pixel bei einer Auflösung von 300 dpi (Tabelle 2.2).
Die z. T. widersprüchliche Forderung nach einer möglichst sicheren Schätzung der Parameter sowie nach einer möglichst umfassenden Normalisierung auch in schwierigen Fällen mit wenigen Abtastpunkten erfordert eine Kompromißlösung. Dementsprechend wird jeweils auch die Anzahl der zurückgewiesenen Wörter angegeben, die zugleich mit dem mittleren Fehler zu optimieren ist.
Wesentlicher Anhaltspunkt für eine Rückweisung ist die Anzahl der verfügbaren Abtastvektoren. Die jeweils akzeptable Mindestanzahl N_θ^- für die Slantkorrektur ist in Spalte 1 der Tabelle 2.1 angegeben. Wird eine Neuabtastung (NA) zwecks Glättung vorab durchgeführt, so beeinflußt die Abtastvektorlänge natürlich die Gesamtanzahl der Vektoren. Die Spalten 2-6 geben die mittleren Fehler $\frac{1}{413}\sum_{i=1}^{413}|\theta_{0,i}-\theta_i|$ der errechneten Winkel θ_i der i-ten Schriftprobe aus den Testdaten zu den manuell gesetzten Referenzwinkeln $\theta_{0,i}$ wieder. In Klammern findet sich die jeweilige Anzahl zurückgewiesener Wörter. Verglichen mit den nicht normalisierten Testdaten, bei denen sich ein mittlerer Fehlerwinkel von $\frac{1}{413}\sum_{i=1}^{413}|\theta_{0,i}|$=16,47 Grad ergibt, bedeutet dies im günstigsten Fall (bei $\hat{l}=1$) eine Slant-Reduktion um 53 % auf 7,72 Grad. Ein positiver Nebeneffekt bei der Abtastung mit kurzen Abtastvektoren ($\hat{l}=1$) ist, dass die Wahl der Rückweisungsschwelle N_θ^- weitgehend

N_θ^-	keine NA	$\hat{l}=1$	$\hat{l}=2$	$\hat{l}=3$	$\hat{l}=5$
40	8.84 (0)	7.78 (0)	8.14 (0)	9.01 (0)	10.05 (2)
60	8.74 (5)	7.78 (0)	8.14 (0)	9.01 (1)	9.78 (22)
80	8.85 (10)	7.78 (0)	8.14 (0)	8.96 (5)	10.30 (52)
100	8.99 (14)	7.78 (0)	8.13 (1)	8.78 (19)	10.76 (97)
120	9.18 (32)	7.78 (0)	8.09 (5)	9.05 (34)	11.03 (152)
140	9.41 (42)	7.78 (0)	8.04 (15)	9.43 (56)	12.12 (199)
160	9.77 (69)	7.78 (0)	8.02 (24)	9.72 (80)	12.87 (242)
180	10.25 (91)	7.78 (0)	8.27 (32)	9.98 (111)	13.69 (283)
200	10.53 (110)	7.78 (1)	8.42 (45)	10.37 (141)	14.69 (319)
220	11.05 (132)	**7.72** (4)	8.78 (59)	10.96 (169)	15.21 (341)

Tabelle 2.1: Ergebnisse der Slantkorrektur

unkritisch ist. Bei einer Glättung des Entropieverlaufes mit einem gleitenden Durchschnitt wird der mittlere Fehler nochmals leicht auf $7,31°$, bzw. um insgesamt 56 % verringert.
Analog zu der Berechnung der mittleren Winkelabweichung kann eine mittlere Abweichung der Basis- und Kernlinien zwischen manuellen Referenzwerten und geschätzten Werten bestimmt werden. Mit der Referenz-Basislinie $y_{BL,0,i}$ der i-ten Probe und den geschätzten Basislinien $y_{BL,i}$, ergibt sich die mittlere Abweichung in Pixel zu $\frac{1}{413}\sum_{i=1}^{413}|y_{BL,0,i}-y_{BL,i}|$. Die mittleren Abweichungen sind wiederum in Tab. 2.2 für die Abtastvektorlänge $\hat{l}=1,2,3$ gegeben. Die Parameter 10, 20, 30 geben in dieser Tabelle die vorzugebende maximale Clustergröße in Pixel nach oben (spaltenweise) bzw. nach unten (zeilenweise) an. Bei moderater Wahl der Rückweisungsschwelle $N^- \leq 40$ sind bei dieser Wahl der Abtastvektorlängen keine Rückweisungen zu erwarten. Die Anzahl der zurückgewiesenen Wörter bleibt daher außer Betracht. Die in Tab. 2.2 fett markierten Werte stellen für die Basis- bzw. Kernlinie den optimalen Wert für die jeweilige Abtastvektorlänge dar. Anschaulicher werden die Größenverhältnisse, wenn die angegebenen Abweichungen nicht in Pixel, sondern metrisch angegeben werden. Die 1,75 Pixel Abweichung bei der Basislinie entsprechen 0,1483 mm, die 1,79 Pixel gleichen 0,15mm. Die Abweichungen bei der Kernlinie liegen leicht höher. Die jeweiligen Optimalwerte schwanken zwischen 2,98 Pixel = 0,252 mm und 3,22Pixel = 0,27 mm. Diese Differenzen stehen einer durchschnittlichen Kernhöhe von 26.94 Pixel (ca. 25 mm) gegenüber. Die geringfügig größere Abweichung bei der Kernhöhenschätzung liegt in der deutlich größeren Häufigkeit von Oberlängen begründet, die einen stärkeren Störeinfluß ausüben, als die weniger häufig auftretenden Unterlängen bei den Basislinien.

		Basislinie			Kernlinie		
		10	20	30	10	20	30
$\hat{l}=1$	10	2.35	**1.79**	2.17	4.27	**3.22**	3.61
	20	4.61	3.0	3.07	10.82	4.74	4.15
	30	6.28	3.78	3.7	18.01	7.39	6.01
$\hat{l}=2$	10	2.25	**1.76**	2.15	4.35	**3.05**	3.58
	20	4.3	2.9	2.97	10.82	4.64	4.04
	30	6.03	3.66	3.58	17.88	7.46	5.93
$\hat{l}=3$	10	2.22	**1.75**	2.24	4.45	**2.98**	3.49
	20	4.02	2.76	2.84	10.98	4.62	4.04
	30	5.68	3.49	3.44	18.12	7.41	6.00

Tabelle 2.2: Ergebnisse der Größenskalierung

2.4 Kapitelzusammenfassung

Mit den hier vorgestellten Vorverarbeitungsverfahren soll die Varianz, mit der die Handschrift gewöhnlich erzeugt und erfasst wird, noch vor der eigentlichen Modellierung weitgehend eingeschränkt werden.
Im wesentlichen konnten drei Störgrößen identifiziert werden, die im Rahmen der Vorverarbeitung reduziert werden konnten. Dies sind die variable Schreibgeschwindigkeit und die meist schreiberabhängige Schriftneigung und Schriftgröße. Das erste Problem wurde in dem hier entwickelten Gesamtsystem durch eine (formtreue) Neuabtastung gelöst. Für die Minimierung der Schriftneigung und der Zeilenneigung konnte ein effektiver Ansatz über die Entropieminimierung der Integralprojektionen präsentiert werden. Die Schriftgrößennormierung wurde durch ein iteratives Verfahren zur Grund- und Basislinienapproximation realisiert. Aus den durchgeführten Untersuchungen läßt sich ein allgemeines Ablaufschema für die Vorverarbeitung skizzieren:

1. räumlich äquidistante Abtastung mit $\hat{l}=1$
2. Normalisierung
   (a) Korrektur der Zeilenneigung
   (b) Slantkorrektur
   (c) Größennormalisierung
3. formtreue Neuabtastung mit $\hat{l}=5$

Diese Vorverarbeitungsschritte werden im folgenden für sämtliche schreiberunabhängigen Verfahren durchgeführt. Da gewisse Eigenschaften bei schreiberabhängigen Systemen mittrainiert werden können, wird dafür die Vorverarbeitung auf den 3. Schritt reduziert. Eine Sonderanwendung stellt die Korrektur der Zeilenneigung dar, da im allgemeinen Schreiblinien vorgegeben werden. Diese Korrektur kann im Bedarfsfall speziell für den schreiberabhängigen, wie auch für den schreiberunabhängigen Modus aktiviert werden.
Wenngleich das hier gewählte Testszenario mit der Verarbeitung von Einzelworten bereits den ungünstigsten Fall darstellt, konnten bereits erstaunliche Genauigkeiten - insbesondere bei der Grundlinienschätzung erzielt werden. Weiterhin ist zu unterstreichen, dass selbst die manuell bestimmten Referenzwerte für den Neigungswinkel und die Grund- und Basislinien nicht unstrittig sind. Häufig können für diese Variablen selbst durch einen menschlichen Betrachter keine eindeutigen Werte angegeben werden (siehe unteres Beispiel in Abb. 2.7). Eine absolute Genauigkeit kann also bei dieser Problemstellung nicht erwartet werden.

Kapitel 3

Merkmalsextraktion

Unabhängig von der spezifischen Aufgabe eines Mustererkennungssystems ist es das Ziel mit Hilfe der Merkmalsextraktion die für eine Erkennung wesentliche Information aus den zur Verfügung stehenden Rohdaten zu gewinnen. Charakteristische Merkmale, welche sich für eine Klassenunterscheidung eignen, sollen hervorgehoben und dem eigentlichen Erkenner zugeführt werden. Zugleich sollten möglichst irrelevante Daten außer Betracht bleiben.

Diese allgemeine und zunächst einfach klingende Spezifikation der Merkmalsextraktion erweist sich bei näherer Betrachtung als eine Aufgabenstellung, deren Lösung bei nahezu jedem neuen Mustererkennungsproblem eine erneute Herausforderung darstellt. Hinsichtlich der speziellen Problemstellungen existieren in nur einigen Fällen optimierte Merkmalsextraktions-Verfahren, die sich als eine Art Standard-Lösung etabliert haben.

Das Fehlen von solchen Standard-Lösungen trifft ebenso für die Online-Handschrifterkennung zu. Das Finden von geeigneten Verfahren erschwert sich auch dadurch, dass einheitliche, öffentlich zugängliche Datenbasen und darauf durchgeführte Evaluierungen anhand derer verschiedene Merkmalsextraktions-Verfahren objektiv verglichen werden können, kaum existieren. Dennoch lassen sich in der Literatur Vorschläge für Merkmale zur Online-Handschrifterkennung finden, die grob in die zwei Gruppen ’Trajektorien-’ [Yan95, Sch95] und ’Bitmap-’ Merkmale [Man94] eingeteilt werden können.

Trajektorien-Merkmale lassen sich direkt aus einem zeitlich begrenzten Segment der Stifttrajektorie extrahieren. Dieser Merkmalstyp basiert damit auf einer zeitlich lokalen Betrachtungsweise. Bitmap-Merkmale hingegen werten das komplette Schriftbild in globaler Form aus, und werden erst extrahiert, nachdem die zu bearbeitende Schriftprobe vervollständigt wurde. Damit eröffnet sich die Möglichkeit Eigenschaften zu erfassen, die zwar in geometrischer Nachbarschaftsbeziehung zueinander stehen, aber zu völlig unterschiedlichen Zeitpunkten eingegeben wurden (z. B. zeitlich verzögert gesetzte i-Punkte

oder t-Striche).
Die gewählte Struktur dieses Kapitels spiegelt diese Grobunterteilung der Verfahren wieder. Abschnitt 3.1 beschreibt einige vielversprechende Extraktionsverfahren für Trajektorien-Merkmale, während Abschnitt 3.2 eine Übersicht über die untersuchten Bitmap-Merkmale liefert. Im Abschnitt 3.3 werden die behandelten Verfahren einzeln, sowie in möglichen Kombinationen evaluiert. Bei den hier dargestellten Verfahren, handelt es sich lediglich um eine Auswahl der erfolgreichsten Ansätze. Untersuchungen zu weiteren in Betracht gezogenen Verfahren können [Hü99] entnommen werden.

3.1 Trajektorienmerkmale

Ausgangspunkt für die Extraktion von Trajektorien-Merkmalen sind die neuabgetasteten Vektorsequenzen der Rohdaten (Glg. (2.1)). Zu jedem Abtastzeitpunkt k wird durch eine Merkmalsextraktion (ME) aus den vorverarbeiteten Kartesischen Daten, ggf. unter Berücksichtigung des Stiftdrucks, ein Merkmalsvektor $\vec{m}$ generiert:

$$(x_{Anf}, y_{Anf}, x_{End}, y_{End}, \tilde{p})(k) \xrightarrow{ME} \vec{m}(k). \tag{3.1}$$

Überwiegend wird bei der Berechnung eines Merkmalsvektors zum Abtastzeitpunkt k zudem ein zeitlicher Kontext $\{k-k', \cdots, k, \cdots, k+k'\}$ berücksichtigt

$$\left(\begin{array}{c} (x_{Anf}, y_{Anf}, x_{End}, y_{End}, \tilde{p})(k-k') \\ \cdots \\ (x_{Anf}, y_{Anf}, x_{End}, y_{End}, \tilde{p})(k) \\ \cdots \\ (x_{Anf}, y_{Anf}, x_{End}, y_{End}, \tilde{p})(k+k') \end{array}\right) \xrightarrow{ME} \vec{m}(k), \tag{3.2}$$

der das Trajektorienstück zwischen den Abtastpunkten $k-k'$ und $k+k'$ einbezieht. Das Trajektorienstück wird als 'Fenster' $\mathcal{F}$ bezeichnet, wobei die Fenstergröße F mit $2k'+1$ angegeben werden kann.
Eine neuabgetastete Schriftprobe, bestehend aus K Abtastvektoren, führt somit zu genau K aufeinanderfolgenden Merkmalsvektoren, deren Dimensionalität von der gewählten Extraktionsmethode selbst abhängt. Die aus einer Schriftprobe generierte Folge von Merkmalsvektoren wird im folgenden auch als Merkmalsstrom bezeichnet.

3.1.1 Kettenkodierung

Bei der Online-Handschrifterkennung ist eine komplette Schriftprobe durch die zeitliche Funktion der Stiftpositionen in Kartesischen- oder Zylinder-Koordinaten definiert. Nahe-

liegend erscheint es daher, die (planare) Trajektorie in parametrischer Form zu beschreiben [Yan95]. Da lediglich eine Unterscheidung zwischen aufgesetztem oder abgehobenem Stift zu erfolgen hat, gibt der ebenfalls abgetastete Stiftdruck $\tilde{p}$ hinreichenden Aufschluß über die Stiftposition in der dritten Dimension. $\tilde{p} > 0$ bedeutet, dass die Stiftspitze das Digitalisiertablett berührt, während $\tilde{p} \leq 0$ anzeigt, dass der Stift abgehoben ist.
Ist eine planare Kurve mit $x(t)$ und $y(t)$ bekannt, stellt die Funktion der Trajektorientangente $\alpha(t)$ mit

$$\alpha(t) = \arctan\left(\frac{\dot{y}(t)}{\dot{x}(t)}\right) \text{ für } \dot{x}(t) \neq 0, \tag{3.3}$$

eine kompakte und bis auf die absolute Position der Schriftprobe vollständige Beschreibung der Trajektorie dar. Durch Abtastung und Neu-Abtastung wird schließlich die Trajektorientangente durch die Trajektoriensekante $\alpha(k)$ zwischen den Anfangs- und Endpunkten $(x_0(k), y_0(k))$ und $(x_1(k), y_1(k))$ des Abtastvektors mit

$$\alpha(k) = \arctan\left(\frac{y_1(k) - y_0(k)}{x_1(k) - x_0(k)}\right) \text{ für } x_1(k) \neq x_0(k), \tag{3.4}$$

approximiert. Da der Abstand zwischen zwei aufeinander folgenden Punkten durch die Neuabtastung quasi konstant ist, läßt sich der neuabgetastete Schriftzug allein durch $\alpha(k)$ rekonstruieren. Würde $\alpha(k)$ nun direkt als Merkmal verwendet, hätte dies den entscheidenden Nachteil, dass sich bei 0 bzw. 2π eine Unstetigkeit ergäbe. Dies würde dazu führen, dass $\alpha(k)$ bei Abtastvektoren in positiver x-Richtung mit horizontaler Ausrichtung bei kleinen Abweichungen zwischen 0 und 2π springen würden. Aus diesem Grund wird die nach Glg. (3.4) ermittelte Sekantensteigung durch Angabe des Sinus- und Cosinuswinkels als Merkmalsvektor verwendet [Rig96a]:

$$\vec{m}_\alpha(k) = \begin{pmatrix} \sin(\alpha(k)) \\ \cos(\alpha(k)) \end{pmatrix}. \tag{3.5}$$

In analoger Weise läßt sich ebenso ein Merkmalsvektor berechnen, um Richtungsänderungen explizit zu erfassen. Mit $\Delta\alpha(k) = \alpha(k+1) - \alpha(k)$ ist dazu lediglich in Glg. (3.5) $\alpha(k)$ durch $\Delta\alpha(k)$ zu ersetzen:

$$\vec{m}_{\Delta\alpha}(k) = \begin{pmatrix} \sin(\Delta\alpha(k)) \\ \cos(\Delta\alpha(k)) \end{pmatrix}. \tag{3.6}$$

Die anschließende Quantisierung der durch Glg. (3.5) gewonnenen Merkmalsvektoren ergibt eine spezielle Form einer Kettenkodierung. Im Gegensatz zu einer Standard-Kettenkodierung [Abm94] wird auf eine statische a priori Einteilung der Richtungsvektoren verzichtet. Vielmehr wird hierbei das Codebuch für die Richtungsquantisierung aus den Trainingsdaten gelernt.
Es ist leicht ersichtlich, dass es sich bei dieser Extraktionsmethode um ein durch Glg. (3.1) typisiertes Verfahren handelt. Ein zeitlicher Kontext läßt sich jedoch

auch hier erfassen. Dies geschieht, indem eine Folge von Merkmalsvektoren $\vec{m}(k-k'),\cdots,\vec{m}(k),\cdots,\vec{m}(k+k') = \vec{m'}(k)$ zu einem *Multi-Frame*-Vektor $\vec{m'}(k)$ zusammengefasst wird, und dieser Multi-Frame-Vektor schließlich dem nachfolgenden Vektorquantisierer präsentiert wird.

3.1.2 Diskrete Cosinus Transformation der Trajektorie

Betrachtet man den zeitlichen Verlauf der abgetasteten Stiftposition in Abb. 3.1, separiert nach x und y, bietet sich auf Grund der zu beobachtenden Schwingungen eine weitere Repräsentation der Stiftbewegungen durch eine Schwingungsanalyse an. Als vorteilhaft in verschiedenen Bereichen der Mustererkennung oder der Bilddatenkompression (JPEG, MPEG) hat sich die Diskrete Cosinus Transformation (DCT) erwiesen. Die DCT soll hiermit auch für die Online-Handschrifterkennung eingehender untersucht werden. Insbesondere ist bei der DCT die Kovarianz-mindernde Eigenschaft der resultierenden DCT-Koeffizienten hervorzuheben [Pra91].

Der in Abb. 3.1 zu sehende, mit k tendenziell steigende Verlauf von x, spiegelt die überwiegend in positive x-Richtung verlaufende Schreibrichtung wieder. Mit der Verwendung von Sub-Wort-Modellen für die spätere Erkennung kann die absolute (x,y)-Position eines Buchstabens innerhalb eines Wortes für die Merkmalsextraktion jedoch nicht relevant sein. Dies gilt darüber hinaus auch für einen entstehenden Offset in y, hervorgerufen

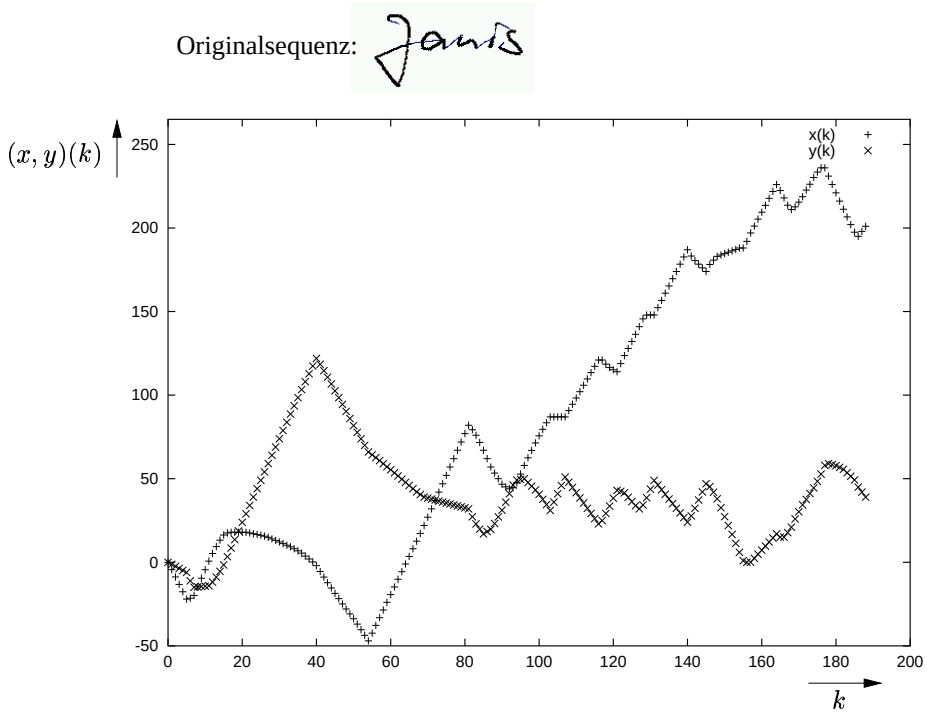


Abbildung 3.1: Verlauf von (x,y)(k) einer Schriftprobe

durch einen Zeilenwechsel. Aus diesem Grund wird statt des gezeigten absoluten Verlaufes von x und y die Analyse innerhalb eines Fensters durchgeführt (entsprechend Glg. (3.2)).

Fensterung

Bei den Fenster-basierten Extraktionsverfahren sind einige Besonderheiten zu beachten, die stellvertretend für weitere Fenster-Verfahren am Beispiel der DCT erklärt werden. Dazu betrachte man einen Ausschnitt bestehend aus F Abtastvektoren einer Abtastsequenz $V = \{\vec{v}(1), \cdots, \vec{v}(k-k'), \cdots, \vec{v}(k), \cdots, \vec{v}(k+k'), \cdots, \vec{v}(K)\}$ der Länge K, mit $K >> F = 2k'+1$. $\vec{v}(k)$ bezeichnet dabei den k-ten Abtastvektor zwischen den Punkten $\mathbf{v}_0(k) = (x_0, y_0)(k)$ und $\mathbf{v}_1(k) = (x_1, y_1)(k)$. Da in einer solchen Sequenz der Endpunkt $\mathbf{v}_1$ des einen Abtastvektors mit dem Anfangspunkt $\mathbf{v}_0$ des nachfolgenden Abtastvektors identisch ist, ist es ausreichend in die Berechnungen lediglich die Anfangspunkte einzubeziehen. Das für die Merkmalsberechnungen nötige Fenster um den k-ten Abtastvektor $\vec{v}(k)$ wird demnach auf eine Punktmenge $\mathcal{F}_k$ reduziert.

Unter besonderer Berücksichtigung von Sequenzanfang ($k = 0, \cdots, k'$) und -Ende ($k = K-k', \cdots, K$) ergibt sich

$$\begin{aligned}
\mathcal{F}_k &= \{\mathbf{v}_0(k-k'), \mathbf{v}_0(k-k'+1), \cdots, \mathbf{v}_0(k+k'-1), \mathbf{v}_0(i+k')\} && \text{für } k \in [k', K-k'] \\
\mathcal{F}_k &= \{\mathbf{v}_0(0), \cdots, \mathbf{v}_0(2k')\} && \text{für } k \in [0, k'] \\
\mathcal{F}_k &= \{\mathbf{v}_0(K-2k'), \cdots, \mathbf{v}_0(K)\} && \text{für } k \in [K-k', K].
\end{aligned} \tag{3.7}$$

Wie bereits oben erwähnt, sollte die Merkmalsextraktion unabhängig von der absoluten Position der Trajektorie erfolgen. Werden die Punkte eines Fensters hingegen auf einen gleitenden Referenzpunkt bezogen, lassen sich zudem Effekte ausblenden, die durch die vorherrschende Schreibrichtung (von links nach rechts) oder einen Zeilenwechsel hervorgerufen werden. Als ein solcher Referenzpunkt bietet sich natürlich der mittlere Punkt $\mathbf{v}_k$ an, zu dem das Fensters $\mathcal{F}_k$ gebildet wurde. Setzt sich, allgemein betrachtet, das Fenster $\mathcal{F}_k$ aus den Punkten $\{\mathbf{f}'_0, \mathbf{f}'_1, \cdots, \mathbf{f}'_{F-1}\}$ zusammen, die entsprechend Glg. (3.7) bestimmt wurden, können diese nun um den Vektor $([0,0]^T - \mathbf{f}'_{k'})$ in Richtung des Koordinatenursprungs verschoben werden, wodurch sich

$$\mathcal{F}_k = \{\mathbf{f}_0, \cdots, \mathbf{f}_{k'}, \cdots, \mathbf{f}_{F-1}\} \tag{3.8}$$

mit $\mathbf{f}_{k'} = [0,0]^T$ ergibt. Bei allen folgenden Verfahren wird ein solches relatives Fenster zur Berechnung der Trajektorienmerkmale benutzt.

Für die eigentliche Merkmalsextraktion lassen sich nun aus einem relativen Fenster die DCT-Koeffizienten berechnen. Mit der Aufteilung der Punkte $\mathbf{f}$ nach x und y läßt sich mit

$$\begin{aligned}
m_{DCT_x}(i) &= \sqrt{\tfrac{2}{F}} \sum_{k=0}^{F-1} f_{_xk} \cos \tfrac{(2k+1)\pi i}{2F}, \\
m_{DCT_y}(i) &= \sqrt{\tfrac{2}{F}} \sum_{k=0}^{F-1} f_{_yk} \cos \tfrac{(2k+1)\pi i}{2F}
\end{aligned} \tag{3.9}$$

für das k-te Fenster ein Merkmalsvektor

$$\vec{m}_{DCT}(k) = [m_{DCT_x}(0), \cdots, m_{DCT_x}(i), m_{DCT_y}(0), \cdots, m_{DCT_y}(i)]^T \tag{3.10}$$

mit $i+1$ Merkmalen pro Richtung definieren.
Der Vorteil der Betrachtung eines relativen Fensters (Verschiebung zum Koordinatenursprung) wird bei der Spektralanalyse noch nicht offensichtlich. Ein ähnlicher Effekt hätte hier sicherlich durch eine Unterdrückung der Tiefpaßanteile der Transformierten erzielt werden können. Bei den nachfolgend beschriebenen Verfahren hingegen ist die relative Betrachtungsweise unerläßlich. Die Berücksichtigung der Vergleichbarkeit verschiedener Verfahren, wie auch die Berücksichtigung von Implementierungsaspekten sprechen schließlich für eine konsistente Fensterung, auch bei der DCT.

3.1.3 Bézierkurve

Kehrt man aus der 'Frequenzwelt' zurück zu einer geometrischen Betrachtung einer Kurve, ergeben sich potenziell weitere Ansatzpunkte, um ein Kurvenstück einerseits möglichst kompakt, andererseits möglichst genau zu beschreiben. Dazu lassen sich beispielsweise Methoden aus dem Computer Aided Geometric Design (CAGD) heranziehen [Far94, Ram97]. Gegenüber anderen Approximationsverfahren haben Bézierkurven und Splines wichtige CAGD-relevante Eigenschaften, sodass mit ihnen einfach Kurven und Flächen zu gestalten sind.
Bézierkurven oder Splines werden anhand von Kontrollpunkten konstruiert. In CAGD-Systemen, wie in konventionellen Zeichenprogrammen, kann der Benutzer durch einfaches Hinzufügen oder Löschen von Kontrollpunkten und insbesondere mit der Veränderung ihrer Lage schnell und präzise Kurven- bzw. Flächenformen konstruieren und modifizieren. Dabei ist es möglich mit wenigen Kontrollpunkten komplexe Formen zu bilden. Wird dieser Gedanke auf die Handschrifterkennung übertragen, sollten diese Kontrollpunkte aussagekräftige Merkmale über die Form der Stifttrajektorie bieten.
Während man bei CAGD-Anwendungen versucht Kurven mit Kontrollpunkten zu konstruieren, präsentiert sich das Problem für die Merkmalsextraktion in genau umgekehrter Weise. Anhand eines gegebenen Kurvenstückes sind die Kontrollpunkte in der Form zu ermitteln, dass die Approximierende das Kurvenstück möglichst exakt nachbildet.
Bézierkurven und Splines zeichnen sich darüber hinaus durch einige Vorteile aus, die speziell für die Online-Handschrifterkennung relevant sind. Insbesondere sind Bézierkurven

- variationsvermindernd, wodurch sich eine insgesamt glättende Wirkung auf die Trajektorie einstellt.
- Bézierkurven und Splines sind relativ einfach und robust zu konstruieren und

- sind invariant unter affinen Abbildungen. D. h., normalisierende Operationen, wie Skalierungen, Scherungen oder Rotationen können prinzipiell auch nach der Merkmalsextraktion auf die Merkmale angewendet werden.

Bevor die Approximation einer Trajektorie durch eine Bézierkurve beschrieben werden kann, ist es erforderlich deren Konstruktion in Grundzügen zu verstehen. Eine Bézierkurve J-ten Grades wird allgemein mittels folgender Summe definiert:

$$\vec{B}_J(t) = \sum_{j=0}^{J} r_{Jj}(t)\vec{a}_j. \tag{3.11}$$

Bei einer Bézierkurve handelt es sich folglich um eine gewichtete Summe der Kontrollpunk-

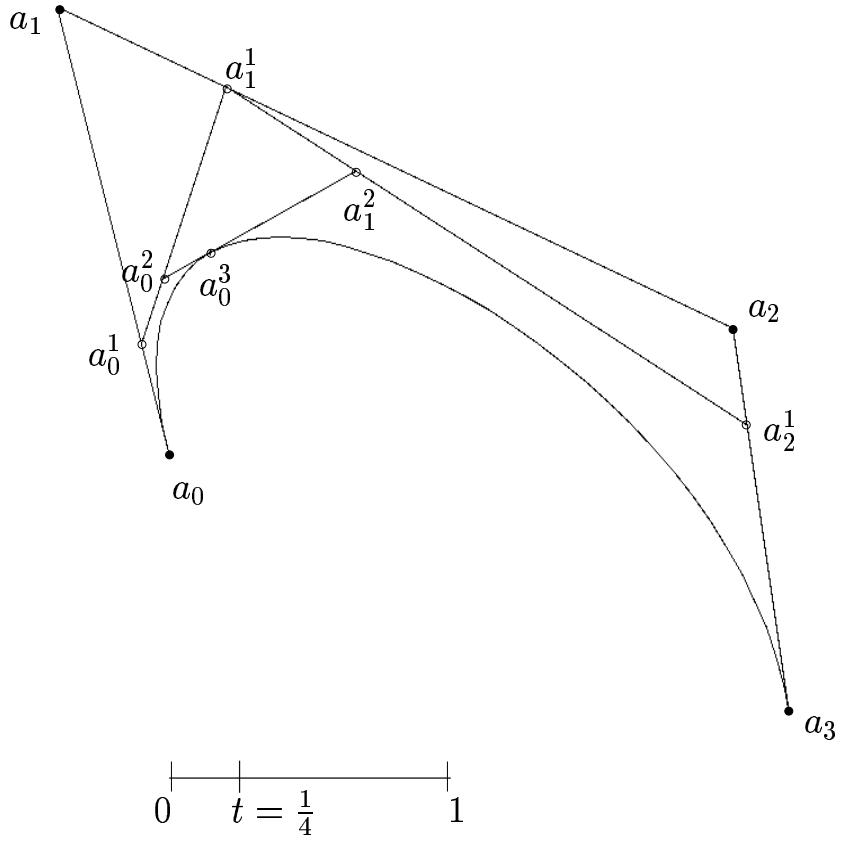


Abbildung 3.2: Konstruktion einer Bézierkurve

te $\vec{a}_j$. Bei einem Grad $J = 1$ ergibt sich die lineare Interpolation zwischen den Kontrollpunkten $\vec{a}_0$ und $\vec{a}_1$. Als Gewichtungsfaktoren werden die Bernstein-Operatoren

$$r_{Jj}(t) = \binom{J}{j} t^j (1-t)^{J-j} \tag{3.12}$$

verwendet. Man beachte, dass die Bernstein-Operatoren ihrerseits wiederum von dem Argument t abhängen. Für das Argument $t \in \Re$ gilt hier $0 \leq t \leq 1$. Die Skalierung erfolgt über die Position der Kontrollpunkte. Des weiteren genügen die Bernstein-Operatoren der

allgemeinen Normierungsbedingung

$$\sum_{j=0}^{J} r_{Jj}(t) = 1, \tag{3.13}$$

wodurch die Bézierkurve auch als baryzentrische Kombination gilt. Die konstruierte Kurve liegt bei nicht negativen Gewichtungen $r_{Jj}(t)$ innerhalb der durch die Kontrollpunkte aufgespannten konvexen Hülle (Abb. 3.2).

Mit dem de-Casteljau-Algorithmus [Far94] lassen sich in rekursiver Form Zwischenpunkte $\vec{a}_j^i(t)$ berechnen, mit

$$a_j^l(t) = (1-t)a_j^{l-1}(t) + a_{j+1}^{l-1}(t), \tag{3.14}$$

von denen dann $\vec{a}_0^J(t)$ genau auf der Bézierkurve $\vec{B}_J$ liegt, sodass $\vec{a}_0^J(t) = \vec{B}_J(t)$ gilt.

Vorausgesetzt, die Kontrollpunkte $\vec{a}_0, \vec{a}_1, \cdots, \vec{a}_J$ sind gegeben, können die Zwischenpunkte wiederum durch die Bernstein-Operatoren vom Grad l ausgedrückt werden:

$$\vec{a}_j^l(t) = \sum_{i=0}^{l} r_{Ji}(t)\vec{a}_{i+j} \qquad \text{mit} \begin{cases} l \in \{0, \cdots, J\} \\ j \in \{0, \cdots, J-l\} \end{cases} \tag{3.15}$$

Abb. 3.2 zeigt eine Bézierkurve vom Grad 3 mit den gegebenen Kontrollpunkten $\vec{a}_0, \vec{a}_1, \vec{a}_2, \vec{a}_3$ und den konstruierten Zwischenpunkten. Mit der Konvention $\vec{a}_j^0(t) = \vec{a}_j$ läßt sich aus Abb. 3.2 auch ablesen, dass der Zwischenpunkt $\vec{a}_j^l$ in dem selben Verhältnis zu $\vec{a}_j^{l-1}$ und $\vec{a}_{j+1}^{l-1}$ steht, wie der Kurvenparameter t zu 0 und 1.

Sehr anschaulich wird die Konstruktion der Bézierkurve in Abb. 3.3, wenn man sich die Konstruktionslinien zwischen Kontroll- und Zwischenpunkten als Stabkonstruktion vorstellt. Die Kontrollpunkte stehen fest, während die Schnittpunkte der Verbindungslinien beidseitig gleitend gelagert sind. Verschiebt man das dehnbare, bewegliche innere Stabwerk unter Berücksichtigung gleicher Schnittverhältnisse und läßt man den Punkt $\vec{a}_0^3$ im gleichen Schnittverhältnis zu den übrigen Verbindungslinien auf seiner Verbindungslinie mitgleiten, zeichnet der Punkt $\vec{a}_0^3$ die Bézierkurve.

Die entscheidende Frage, wie nun aus einem abgetasteten Kurvenstück die Kontrollpunkte zur Nutzung als Merkmale bestimmt werden, ist allein mit den bisherigen Erläuterungen noch nicht zu beantworten.

Um diese wesentliche Frage schließlich zu klären, wird wieder von einem relativen Fenster gemäß Glg. (3.8) ausgegangen, welches die Abtastpunkte $\mathbf{f}_k$ des Trajektorienstückes, verschoben zum Koordinatenursprung enthält. Soll nun dieses Trajektorienstück beispielsweise durch vier Kontrollpunkte beschrieben werden, ist eine Approximation durch eine Bézierkurve vom Grade 3 zu wählen. Aufgetrennt nach x und y ergäbe dies einen Merkmalsvektor, bestehend aus 8 Komponenten.

Für die Bézierkurve dritten Grades ist folglich

$$\vec{B}_3(t_k) = \begin{bmatrix} x(t_k) \\ y(t_k) \end{bmatrix} \stackrel{!}{=} \begin{bmatrix} f_{xk} \\ f_{yk} \end{bmatrix} = \mathbf{f}_k \;\text{ mit } t_k = \frac{k}{F-1} \;\text{ und } k = 0, \cdots, F-1 \tag{3.16}$$

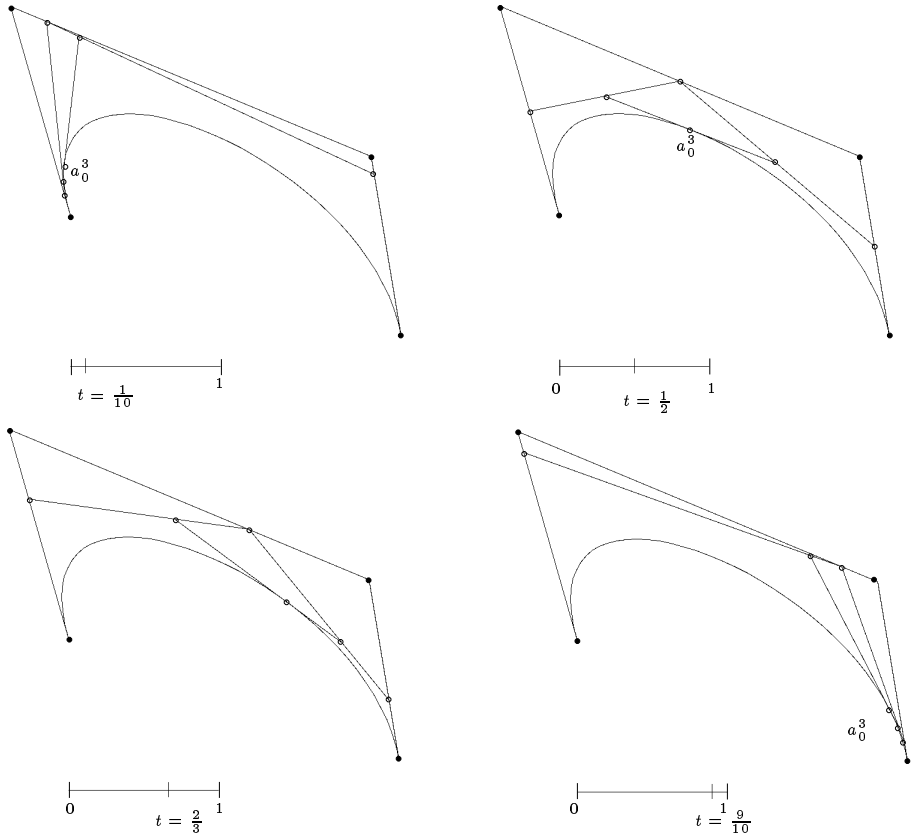


Abbildung 3.3: Bézierkurve - $\vec{B}_3(t)$

zu fordern. Für jeden diskreten Zeitpunkt t_k ist weiterhin nach Glg. (3.11)

$$\vec{B}_3(t_k) = \sum_{j=0}^{3} r_{3,j}(t_k)\vec{a}_j, \tag{3.17}$$

bzw. unter Einbeziehung der Approximationsforderung (Glg. (3.16))

$$\begin{bmatrix} x(t_k) \\ y(t_k) \end{bmatrix} = \sum_{j=0}^{3} r_{3,j}(t_k) \begin{bmatrix} a_{xj} \\ a_{yj} \end{bmatrix} \tag{3.18}$$

zu lösen. Eine detailiertere Darstellung dieses Zusammenhangs, separiert nach x- und y-Richtung führt zu einem linearen Gleichungssystem:

$$\begin{bmatrix} x(t_0) \\ \vdots \\ x(t_{F-1}) \end{bmatrix} = \begin{bmatrix} r_{3,0}(t_o) & r_{3,1}(t_o) & r_{3,2}(t_o) & r_{3,3}(t_o) \\ \vdots & \vdots & \vdots & \vdots \\ r_{3,0}(t_{F-1}) & r_{3,1}(t_{F-1}) & r_{3,2}(t_{F-1}) & r_{3,3}(t_{F-1}) \end{bmatrix} \begin{bmatrix} a_{x_0} \\ a_{x_1} \\ a_{x_2} \\ a_{x_3} \end{bmatrix} \tag{3.19}$$

$$\vec{x} \quad = \quad R \quad \vec{a}_x,$$

bzw. in analoger Weise zu

$$\vec{y} = R\vec{a}_y \tag{3.20}$$

für die y-Richtung. In der Regel stehen den 4 Kontrollpunkten mehr als 4 Abtastpunkte zur Verfügung. Das Gleichungssystem ist damit überbestimmt. Häufig handelt es sich darüber hinaus bei derartigen Gleichungssystemen um schlecht konditionierte Probleme [Häm89]. D. h. geringe Änderungen in den zugrunde liegenden Daten führen zu erheblichen Änderungen des Ergebnisses. Bei der Wahl des Lösungsverfahrens sollte dies daher entsprechend berücksichtigt werden.
Ein Verfahren, welches bei dieser Art von Problemen gute Ergebnisse liefert, ist die Singulärwertzerlegung [Häm89]. Dabei wird die Matrix R in der Form

$$R = U\Sigma V^T \tag{3.21}$$

in die orthogonalen Matrizen $U \in \Re^{(F,F)}$ und $V \in \Re^{(4,4)}$ zerlegt, sowie in die $(F \times 4)$-Diagonalmatrix Σ.
Zur Lösung des Gleichungssytems wird zunächst ein äquivalentes Minimierungsproblem

$$\|R\vec{a}_x^* - \vec{x}\|_2^2 = \min_{\vec{a}_x \in R^4} \|R\vec{a}_x - \vec{x}\|_2^2 \tag{3.22}$$

aufgestellt, mit der Euklid'schen Norm $\|\cdot\|_2$. Es läßt sich weiterhin zeigen, dass ein solches äquivalentes Minimierungsproblem stets lösbar ist [Häm89]. Unter Verwendung der Matrixzerlegung lassen sich mit $d_i = +\sqrt{\lambda_i}$, also der Wurzel der Eigenwerte von $R^T R$, alle Lösungen durch

$$\vec{a}_x^* = \sum_{i=1}^{4} \frac{1}{d_i} \left[U^T \vec{x}\right]_i \vec{v}^i + \sum_{i=5}^{n} z_i \vec{v}^i \tag{3.23}$$

angeben. Eine ausgezeichnete Lösung des äquivalenten Minimierungsproblems ist die Pseudonormallösung

$$\vec{a}_x^* = \sum_{i=1}^{4} \frac{1}{d_i} \left[U^T \vec{x}\right]_i \vec{v}^i, \tag{3.24}$$

da sie die Lösung mit der minimalen Euklid'schen Norm darstellt. Existenz und Eindeutigkeit der Pseudonormallösung sind ebenfalls garantiert.
Bei konstanter Fenstergröße und Merkmalsanzahl ist die Matrix R unabhängig von den zu verarbeitenden Abtastpunkten. Für die Merkmalsextraktion ist dies von großem Vorteil, da für jede Kombination Fenstergröße/Merkmalsanzahl die Singulärwertzerlegung vorab durchgeführt werden kann. Die eigentliche Merkmalsextraktion ist dann die Berechnung der Pseudonormallösung nach Glg. (3.24).
Mit Glg. (3.24) wird schließlich die Hälfte des resultierenden Merkmalsvektors, durch die x-Koordinaten der Kontrollpunkte besetzt. Analog dazu wird die zweite Hälfte durch die y-Koordinaten der Kontrollpunkte $\vec{a}_y^*$ besetzt:

$$\vec{m}_{Bezier} = \left[a_{x,0}^*; \cdots a_{x,3}^*;\ a_{y,0}^*; \cdots a_{y,3}^*\right]^T \tag{3.25}$$

Zur visuellen Kontrolle der Merkmalsextraktion können nun die so erzeugten Merkmale wiederum als Kontrollpunkte verwendet werden, aus denen sich eine Bézierkurve rekonstruieren läßt. Somit bietet sich ein Vergleich der originalen Abtastsequenz und der rekonstruierten

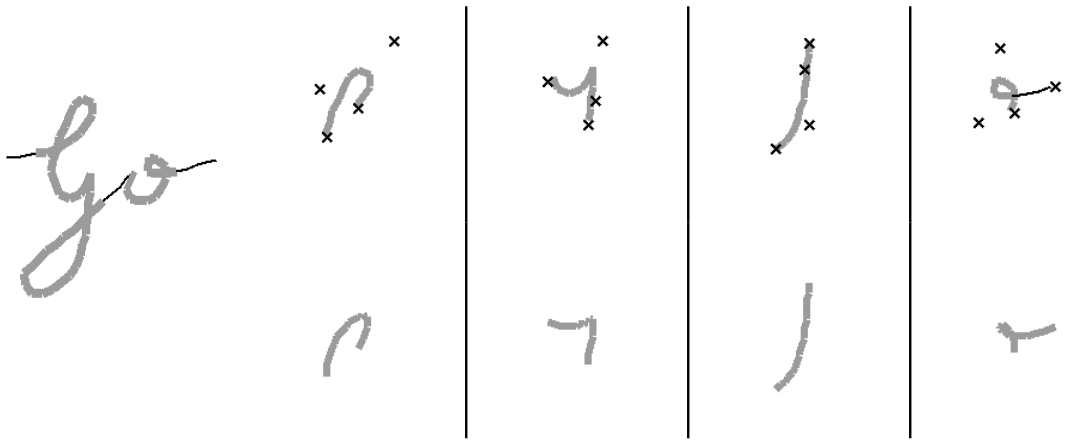


Abbildung 3.4: Abgetastete Trajektorienstücke des Wortes 'Go' mit approximierten Kontrollpunkten und rekonstruierten Bézier-Kurven

Trajektorie an. Dies ist in Abb. 3.4 dargestellt. Die obere Zeile in Abb. 3.4 zeigt die Abtastsequenz mit den daraus extrahierten Kontrollpunkten. Die untere Zeile gibt das dazugehörige Rekonstruktionsergebnis wieder. Erstaunlich ist hierbei, dass es mit einer relativ kleinen Anzahl von Kontrollpunkten möglich ist, komplexe Kurvenverläufe weitgehend fehlerfrei zu modellieren. Ob, und in wie weit sich dies auch auf die Erkennungsgenauigkeit auswirkt, werden später die Ergebnisse zeigen.

Mit der Verwendung von Bézierkurven eröffnet sich eine weitere interessante Option. Bei den abgetasteten Trajektorien handelt es sich um stückweise lineare Funktionen, die im allg. nicht stetig differenzierbar sind. Die Bildung von partiellen Ableitungen, über mehrere Abtastvektoren hinweg, ist damit nicht möglich. Für Bézierkurven, und damit auch für approximierte Trajektorien, existiert hingegen die Möglichkeit Ableitungen zu berechnen. Ausgehend von den berechneten Kontrollpunkten können diese in die Ableitungsvorschriften für Bézierkurven eingesetzt werden:

$$\begin{aligned}
(\vec{B}_{x3})'(t) &= 3\sum_{j=0}^{2}(\mathbf{a}_{x(j+1)} - \mathbf{a}_{xj})t^j(1-t)^{2-j} \\
(\vec{B}_{y3})'(t) &= 3\sum_{j=0}^{2}(_{y(j+1)} - \mathbf{a}_{yj})t^j(1-t)^{2-j} \\
(\vec{B}_{x3})''(t) &= 6\left[(\mathbf{a}_{x2} - 2\mathbf{a}_{x1} + \mathbf{a}_{x0})(1-t) + (\mathbf{a}_{x3} - 2\mathbf{a}_{x2} + \mathbf{a}_{x1})t\right] \\
(\vec{B}_{y3})''(t) &= 6\left[(\mathbf{a}_{y2} - 2\mathbf{a}_{y1} + \mathbf{a}_{y0})(1-t) + (\vec{b}_{y3} - 2\mathbf{a}_{y2} + \mathbf{a}_{y1})t\right].
\end{aligned} \tag{3.26}$$

Mit $t = \frac{1}{2}$ können so die Ableitungen zu dem mittigen Abtastpunkt $\mathbf{f}_{k'}$ des betreffenden Fensters gebildet werden.

3.1.4 B-Splines

B-Splines werden - zumindest in CAGD-Anwendungen - gegenüber Bézierkurven bevorzugt. Besonders werden für diese Anwendungen die einfachen Konstruktions- und Modifi-

kationsmöglichkeiten von B-Splines geschätzt. Die Gestalt von B-Splines wird, wie die von Bézierkurven, mit Hilfe von Kontrollpunkten gesteuert. Potenziell können sich B-Splines damit auch für die Modellierung von Trajektorienstücken eignen, wobei sich wiederum die Kontrollpunkte als Merkmale verwenden lassen.

Die Konstruktion eines kubischen B-Splines $\vec{S}_3$ verläuft anscheinend ähnlich zu der Bildung von Bézierkurven. Dazu werden die Kontrollpunkte $\mathbf{a}_j$, gewichtet durch einen Träger, aufsummiert:

$$\vec{S}_3 = \sum_{j=0}^{3} \eta_{j,3}(t_k)\mathbf{a}_j \tag{3.27}$$

Bei diesem Träger handelt es sich um normalisierte, kubische B-Splines, die wie folgt berechnet werden:

$$\begin{aligned} \eta_{0,3} &= \tfrac{1}{6}(1-t)^3, & \eta_{1,3} &= \tfrac{1}{2}t^3 - t^2 + \tfrac{3}{2}, \\ \eta_{2,3} &= \tfrac{1}{2}t^3 - \tfrac{1}{2}t^2 + \tfrac{1}{2}t + \tfrac{1}{6}, & \eta_{3,3} &= \tfrac{1}{6}t^3, \end{aligned} \tag{3.28}$$

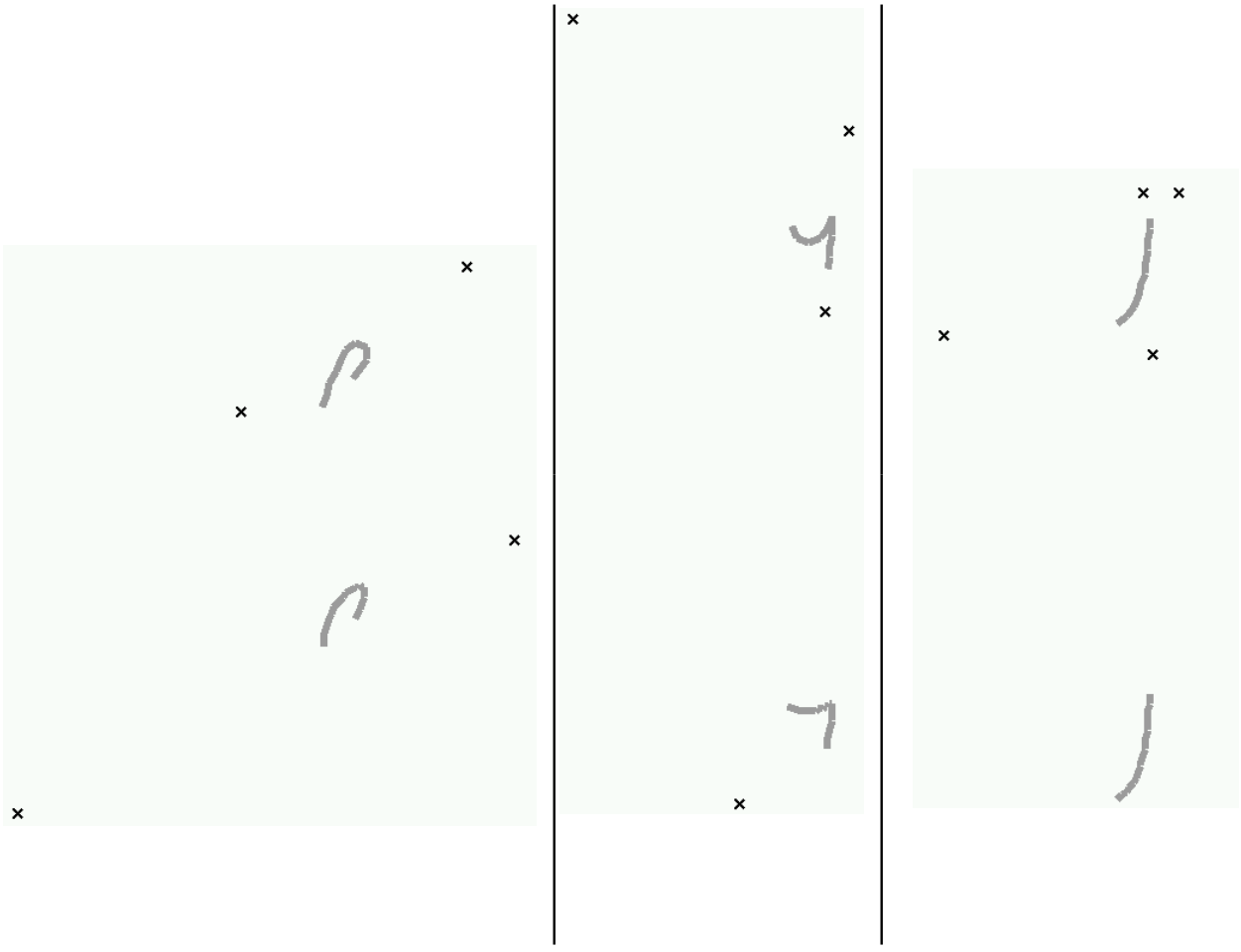


Abbildung 3.5: Abgetastete Trajektorienstücke des Wortes ’Go’ aus Abb. 3.4 mit approximierten Kontrollpunkten und rekonstruierten B-Splines

Nach dem gleichen Prinzip wie bei den Bézierkurven kann daraus ein Gleichungssystem aufgestellt werden. Wiederum durch eine Singulärwertzerlegung ergeben sich die Kontrollpunkte aus der Pseudonormallösung.
Neben den gezeigten Verfahren bietet sich das Prinzip der Singulärwertzerlegung darüber hinaus für diverse andere Approximationsansätze, wie z. B. die Approximation durch Polynome an [Hü99].
Abb. 3.5 erlaubt einen ersten Vergleich zwischen Original mit approximierten Kontrollpunkten und rekonstruierter Trajektorie. Bei den gezeigten Trajektorienstücken handelt es sich um die ersten drei Beispiele aus Abb. 3.4 mit identischen Größenverhältnissen. Auch mit der Verwendung von B-Splines scheint die vorgegebene Kurve gut rekonstruierbar zu sein. Verglichen mit den Bézierkurven fällt allerdings eine insgesamt größere Streuung der Kontrollpunkte auf. Zudem ist eine weitere Eigenschaft von B-Splines erkennbar: die Kontrollpunkte liegen nicht zwangsweise (wie bei Bézierkurven) auf den Anfangs- bzw. Endpunkten der Trajektorie.

3.1.5 Hauptachsentransformation

In den vorangegangenen Abschnitten wurden die entwickelten Lösungsansätze zur Extraktion von Trajektorienmerkmalen aus verschiedenen Blickwinkeln erarbeitet. Neben parametrischen Beschreibungsformen wurden spektrale (DCT), sowie geometrische Eigenschaften (Bézierkurven und B-Splines) einer Trajektorie einbezogen.
Abb. 3.6 zeigt die Superposition aller relativen Fenster aus einem schreiberabhängigen Trainings-Set. Der Grad der Schwärzung ist proportional zu der beobachteten Häufigkeit der Abtastpunkte. Aus Abb. 3.6 wird unmittelbar ersichtlich, dass bestimmte Vorzugsrichtungen existieren. Neben den parametrischen, spektralen und geometrischen Verfahren ist es daher naheliegend, gewisse statistische Eigenschaften der Trajektorien auszunutzen, um zu einer kompakten Repräsentation einer Trajektorie zu gelangen.
Dazu könnte die Hauptachsentransformation dienlich sein. Für diese Transformation wird zunächst aus den K relativen Fenstern $\mathcal{F}_k$ ein Mittelwertvektor berechnet:

$$\vec{\bar{f}} = \frac{1}{K} \sum_{k=1}^{K} \vec{f}(k) \tag{3.29}$$

Der Mustervektor $\vec{f}(k)$ wird mit den x- und y-Komponenten der Abtastpunkte aus dem Fenster $\mathcal{F}_k$ besetzt. Dies geschieht unter Ausschluß des Abtastpunktes $\mathbf{f}_k$, da dieser ohnehin mit dem Koordinatenursprung identisch ist. Es werden also $2(F-1)$-dimensionale Mustervektoren gebildet. Aus den Mustervektoren $\vec{f}(k)$ und dem Mittelwertvektor $\vec{\bar{f}}$ wird eine $M \times M$ Kovarianzmatrix C berechnet, mit $M = 2(F-1)$. Eine Verschiebung der Mustervektoren um $\vec{\bar{f}}$ und eine anschließende Multiplikation mit einer dimesionsreduzierten, orthogonalen

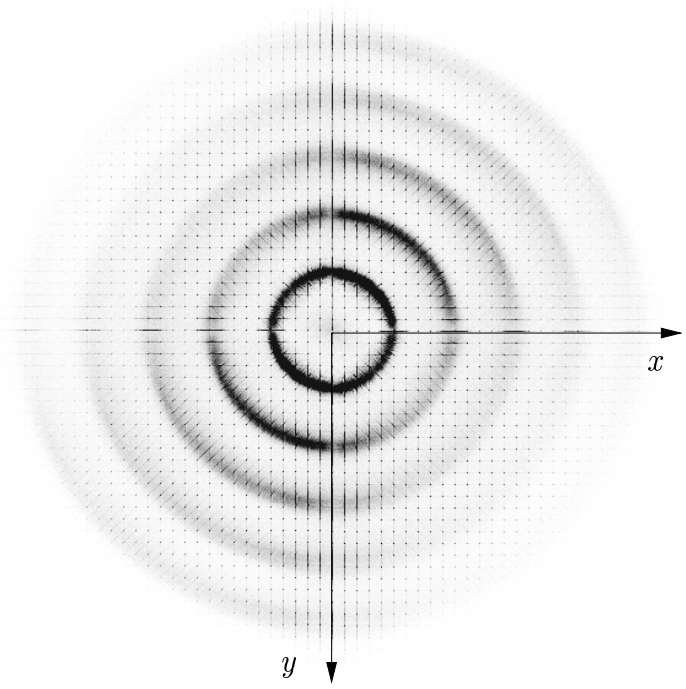


Abbildung 3.6: Superposition relativer Fenster (Trainings-Set)

$(M \times N_m)$-Matrix U, mit $N_m < M$, bestehend aus den Eigenvektoren von C liefert schließlich einen Merkmalsvektor der Dimension N_m.

$$\vec{m}(k) = U(\vec{f}(k) - \vec{\bar{f}}) \tag{3.30}$$

3.2 Bitmap-Merkmale

Die bisher präsentierten Verfahren waren konzeptionell darauf ausgerichtet eine Trajektorie lokal, oder aber innerhalb eines gewissen zeitlichen Abschnittes zu beschreiben. Um geometrische Abhängigkeiten zu erfassen, die nicht unbedingt in einem zeitlichen Zusammenhang stehen, ist es angebracht die zeitliche Folge $V = \{(\vec{v}, \tilde{p})_0, (\vec{v}, \tilde{p})_1, \cdots, (\vec{v}, \tilde{p})_K\}$ einer online abgetasteten Schriftprobe, bestehend aus K Abtastvektoren zunächst in eine statische Bitmap $g(x, y) \in [0, 1]$ zu überführen [Man94].

$$\{(\vec{v}, \tilde{p})_0, (\vec{v}, \tilde{p})_1, \cdots, (\vec{v}, \tilde{p})_K\} \xrightarrow{BM} g(x, y). \tag{3.31}$$

In dieser binären Bitmap werden die Anfangs- und Endpunkte der Abtastvektoren, abhängig vom Stiftdruck $\tilde{p}$, in räumlich diskretisierter Form gesetzt. Durch eine diskretisierte Interpolation werden zudem die Pixel gesetzt, die von den Abtastvektoren mit positivem Stiftdruck überdeckt werden.
Aus der so erzeugten statischen Bitmap wird zur Erzeugung einer Merkmalssequenz die

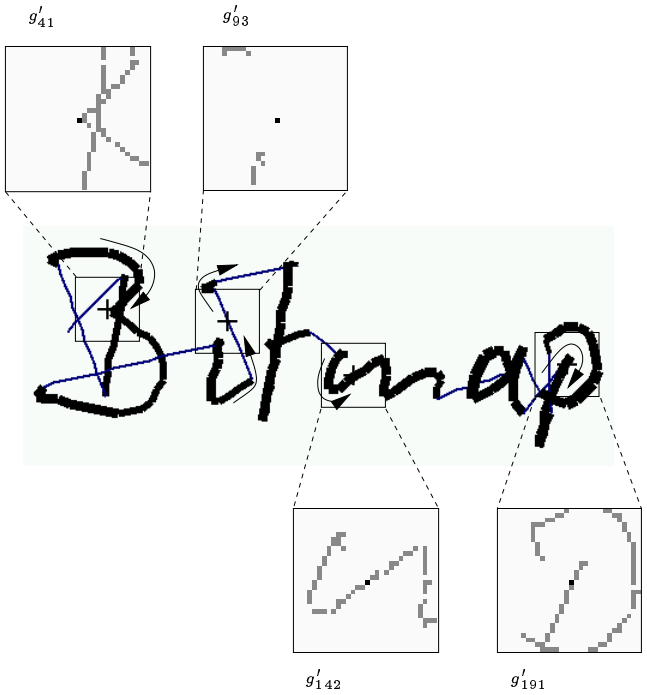


Abbildung 3.7: Gleitende Bitmap

Folge von Abtastvektoren V durchschritten. Wie in Abb. 3.7 gezeigt, wird auch die Spur der Segmente mit negativem Stiftdruck verfolgt. Zu jedem diskretisierten Punktepaar $(\bar{v}_{0,x}(k), \bar{v}_{0,y}(k))$ wird ein um $(\bar{v}_{0,x}(k), \bar{v}_{0,y}(k))$ symmetrischer, quadratischer Ausschnitt $F_{BM,k}$ gebildet. $g'_k(x, y)$ mit $x, y \in F_{BM,k}$ stellt die binären Pixelwerte des betrachteten Fensters dar. Das schwarze Pixel in den gezeigten Ausschnitten dient lediglich dazu den Mittelpunkt $(\bar{v}_{0,x}(k), \bar{v}_{0,y}(k))$ zu markieren. Die grauen Pixel entsprechen den gesetzten Bits. Als sinnvolle Abmessung für $F_{BM,k}$ hat sich ein Wert von 30×30 Pixel herausgestellt. Abb. 3.7 illustriert diese Vorgehensweise und gibt eine Vorstellung über die Größenverhältnisse der ausgeschnittenen Fenster $F_{BM,k}$. Die mit der Abtastfolge $V = \{(\vec{v}, \tilde{p})_0, (\vec{v}, \tilde{p})_1, \cdots, (\vec{v}, \tilde{p})_K\}$ assoziierte Bitmap-Folge $\{g'_0, g'_1, \cdots, g'_K\}$ bildet die Grundlage für die im Folgenden vorgestellten Verfahren zur Extraktion von Bitmap-Merkmalen.

3.2.1 Räumliche Unterabtastung der Bitmap

Die direkte Verwendung einer Bitmap g' als Merkmalsvektor stellt offensichtlich keine vernünftige Lösung dar. Zum einen verhindert dies die Dimension eines solchen Vektors (30×30). Zum anderen würden geringe Verschiebungen einer durch ein Fenster verlaufenden Trajektorie um nur ein Pixel zu einem völlig unterschiedlichen Merkmalsvektor führen. Dem kann jedoch damit begegnet werden, indem der Ausschnitt aus 30×30 Pixel einer Unterabtastung unterzogen wird. Dazu wird ein Ausschnitt $F_{BM,k}$ zunächst in 3×3 gleich große Blöcke $\{F_{BM,k,1}, F_{BM,k,2}, \cdots, F_{BM,k,9}\}$, jeweils bestehend aus 10×10 Pixel, unter-

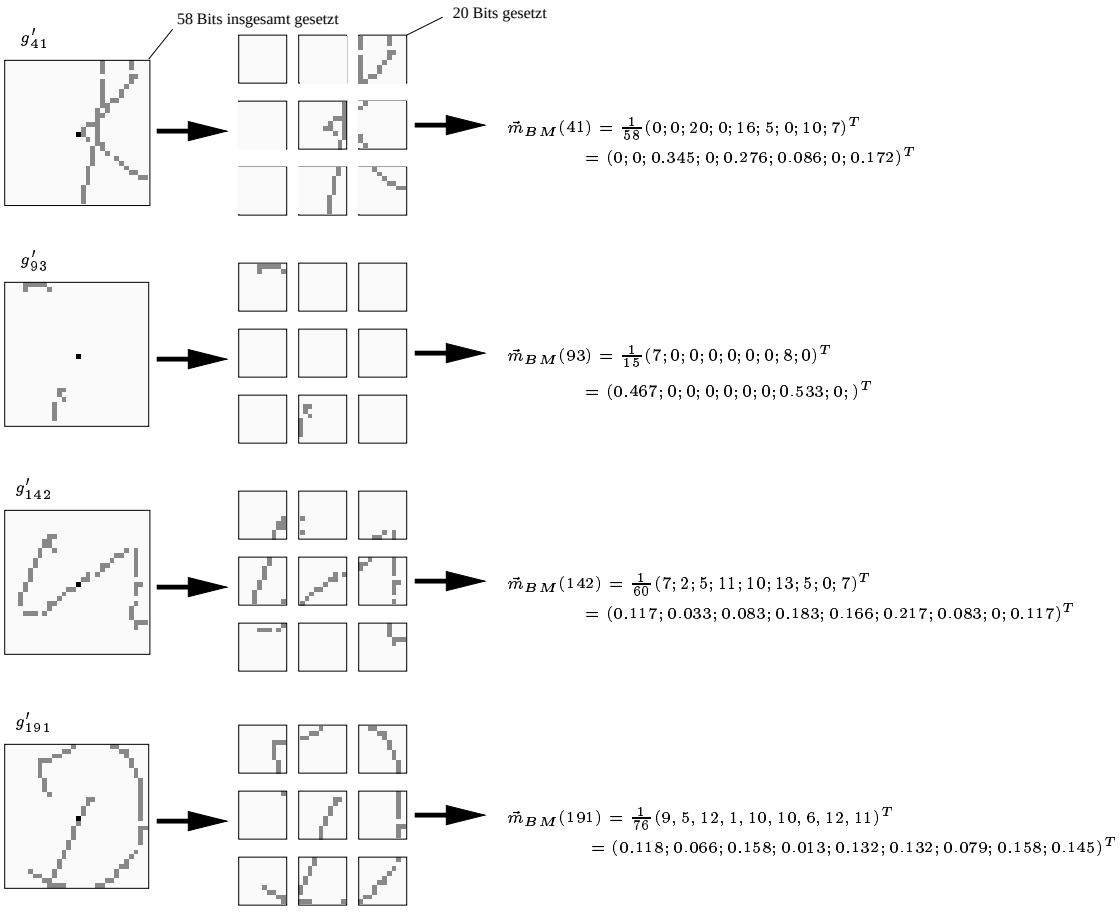


Abbildung 3.8: Unterabtastung der Bitmap

teilt. Die angegebenen Werte sind wiederum aus entsprechenden Experimenten als optimal identifiziert worden. Für jede Bitmap $g'_{k,i}$ des Blocks $F_{BM,k,i}$ wird des weiteren ein mittlerer Grauwert $m_{BM,i}$ bestimmt. Dazu wird die Anzahl der in $g'_{k,i}$ gesetzten Pixel auf die Gesamtanzahl der in g'_k enthaltenen und gesetzten Pixel normiert:

$$m_{BM,i}(k) = \frac{\sum\limits_{x \in F_{BM,k,i}} \sum\limits_{y \in F_{BM,k,i}} g'_{k,i}(x,y)}{\sum\limits_{x \in F_{BM,k}} \sum\limits_{y \in F_{BM,k}} g'_k(x,y)} \tag{3.32}$$

$m_{BM,i}(k)$ stellt sodann die i-te von 9 Komponenten des Merkmalsvektors $\vec{m}_{BM}(k)$ dar.

3.2.2 Diskrete Cosinus Transformation (DCT) der Bitmap

Wie bereits erwähnt, weist die DCT einige positive Eigenschaften auf, die bereits zu erfolgreichen Einsätzen dieser Transformation in verschiedenen Bereichen der Bildverarbeitung führten. Neben der Bildkodierung wurde die Eignung der DCT bereits für verschiedene Problemstellungen im Bereich der Mustererkennung bewiesen. Beispielsweise kann ein auf DCT-Koeffizienten basiertes Erkennungssystem für Bildobjekte direkt auf komprimierten JPEG- oder MPEG-Daten arbeiten [Eic00].

Ausgehend von der Bitmap $g_k(l,n)$ eines 30×30 Pixel großen Ausschnitts $F_{BM,k}$, ergeben sich die DCT-Koeffizienten

$$m_{DCT,k}(i,j) = \frac{1}{30^2} \sum_{l=0}^{29} \sum_{n=0}^{29} g_k(l,n) \cos \frac{(2l+1)\pi i}{2(30+1)} \cos \frac{2(n+1)\pi j}{2(30+1)}. \tag{3.33}$$

Der Merkmalsvektor $\vec{m}_{DCT}(k)$ wird mit den so berechneten DCT-Koeffizienten $m_{DCT,k}(i,j)$ besetzt. Mit $i = 0,1,2$ und $j = 0,1,2$ ergibt sich folglich ein neundimensionaler Merkmalsvektor $\vec{m}_{DCT}(k)$.

3.2.3 Walsh Transformation der Bitmap

Neben der DFT und der DCT wird in der Bildverabeitung häufig die Walsh-Transformation eingesetzt. Die Tranformation in den Bildbereich geschieht, ähnlich wie bei der DCT durch eine mit einem Funktionensystem gewichtete Summe der Originalfunktion. Bei dem Funktionensystem handelt es sich um ein binäres (-1 und 1), vollständiges orthogonales Funktionensystem [Gon92]. Anhand der ersten neun Walsh-Funktionen wird wiederum ein neundimensionaler Merkmalsvektor für jeden Ausschnitt $F_{BM,k}$ berechnet.

3.3 Ergebnisse

Um möglichst repräsentative Ergebnisse für eine größere Anzahl von Schreibern zu erzielen, fällt die Wahl bei der Evaluierung der Merkmale wiederum auf das schreiberunabhängige System.
Um die Vergleichbarkeit verschiedener Extraktionsverfahren zu wahren, wird die verwendete Parameteranzahl nach Möglichkeit konstant gehalten. Die Codebuchgröße des Vektorquantisierers von 100 wurde durchgängig für alle Extraktionsverfahren beibehalten.
Der gewälte Ansatz für das Erkennungssystem wird in den nachfolgenden Kapiteln beschrieben. Vorwegnehmend sei hier bereits erwänt, dass die Erkennungsergebnisse mit diskreten HMM-Systemen ermittelt wurden. Die Erkennungsraten für den Einzelworttest werden anhand der Levenshtein-Distanz

$$Korrektheit = \frac{korrekt\ erkannte\ Worte - gelöschte\ Worte}{Gesamtzahl\ der\ Worte} \tag{3.34}$$

berechnet, mit einer Gesamtanzahl von 4134 Wörtern. Die Ergebnisse wurden mit einem Lexikon bestehend aus 2000 Wörtern erzeugt.

3.3.1 Trajektorienmerkmale

Aus vorab durchgeführten Tests kann - weitgehend unabhängig von dem Extraktionsverfahren - als günstige Fenstergröße $F = 11$ angegeben werden. Die Ergebnisse der

einzelnen Verfahren werden im folgenden in der Reihenfolge dargestellt, wie sie in den vorangegangenen Abschnitten behandelt wurden. Zur übersichtlichen Darstellung der Ergebnisse wird auf die Darstellung aller 20 Einzelergebnisse zu jedem Testschreiber verzichtet. Die Ergebnistabellen enthalten neben der Gesamterkennungsrate (letzte Zeile) die Erkennungsraten der Top-3-Schreiber (1. Zeile), der Flop-3-Schreiber (3. Zeile), sowie der drei Schreiber, deren Ergebnisse am nächsten an der Gesamterkennungsrate liegen (2. Zeile).

Der Blick auf die erzielten Gesamterkennungsraten zeigt zunächst bei den verschiedenen

ank:	86.0 %	bar:	92.0 %	uts:	87.4 %
abr:	70.1 %	dpo:	81.7 %	vdm:	72.7 %
all:	64.1 %	frs:	54.6 %	jmr:	57.6 %
$\emptyset = 76.7\ \%$					

Tabelle 3.1: Erkennungsergebnis - *Sinus- und Cosinuswinkel* mit $F = 11$ und $N_m = 22$

ank:	88.7 %	bar:	93.5 %	uts:	89.8 %
abr:	74.1 %	dpo:	80.7 %	vdm:	76.5 %
all:	61.3 %	frs:	60.8 %	jmr:	61.4 %
$\emptyset = 77.3\ \%$					

Tabelle 3.2: Erkennungsergebnis - *DCT* mit $F = 11$ und $N_m = 6$

ank:	89.3 %	bar:	91.0 %	uts:	88.8 %
abr:	77.0 %	dpo:	79.7 %	vdm:	75.4 %
all:	59.0 %	frs:	62.5 %	jmr:	63.6 %
$\emptyset = 77.4\ \%$					

Tabelle 3.3: Erkennungsergebnis - *Bézierkurve* mit $F = 11$ und $N_m = 8$

Extraktionsverfahren eine relativ geringe Streuung zwischen 77,4 % bei den Bézierkurven (Tab. 3.3) und 74,7 % bei der Hauptachsentransformation (Tab. 3.5). Dies mag zunächst den Schluß nahe legen, dass die Methode mit der die Merkmale aus einer Trajektorie extrahiert werden, nicht sehr ausschlaggebend auf das Ergebnis wirkt. Die hier vorgestellten Extraktionsverfahren stellen allerdings bereits eine Vorauswahl dar. Nicht weiter berücksichtigt wurden hier z. B. die diskrete Fourier Transformation, die mit einer Erkennungsrate von 50,1 % ein eher ernüchterndes Ergebnis lieferte, ähnlich wie die Trajektorienapproximation durch Polynome, mit der maximal 65,6 % Korrektheit erreicht wurden.

Auffällig ist hingegen, dass zum einen die Bézier-Kurven unter den Trajektorienmerkmalen das beste Ergebnis liefert, während die ähnlich konzipierten B-Splines das zweitschlechteste

ank:	84.4 %	bar:	89.1 %	uts:	85.9 %
abr:	74.1 %	dpo:	72.8 %	vdm:	74.9 %
all:	63.1 %	frs:	57.4 %	jmr:	58.2 %
$\emptyset = 75.5\ \%$					

Tabelle 3.4: Erkennungsergebnis - *B-Spline* mit $F = 11$ und $N_m = 8$

ank:	84.4 %	bar:	90.1 %	uts:	89.3 %
abr:	69.5 %	dpo:	73.3 %	vdm:	74.3 %
all:	57.4 %	frs:	61.4 %	jmr:	59.8 %
$\emptyset = 74.7\ \%$					

Tabelle 3.5: Erkennungsergebnis - *Hauptachsentransformation* mit $F = 16$ und $N_m = 8$

Ergebnis zeigen. Zu begründen ist diese Abweichung mit der größeren Varianz, die bei den B-Splines zu beobachten ist. Während optisch ähnliche Vektorsequenzen bei den Bézierkurven zu ähnlich gelagerten Kontrollpunkten führen, bedeuten geringe Abweichungen in den Vektordaten bereits größere Lageveränderungen für die Kontrollpunkte der B-Splines.

3.3.2 Bitmap-Merkmale

Bei den Bitmap-Merkmalen stellt sich im Vergleich zu den Trajektorien-Merkmalen insgesamt eine etwas höhere Streuung unter den Ergebnissen ein. Während mit der einfachen Unterabtastung der gleitenden Bitmap eine maximale Erkennungsrate von 79,7 % erzielt wird, dicht gefolgt von der Walsh-Transformation der Bitmap mit 79,3 %, zeigt sich das Ergebnis der zweidimensionalen DCT der Bitmap deutlich abgeschlagen mit 75,6 %. Dieses Ergebnis überrascht, da mit der DCT und der Walsh-Transformation vollständig orthogonale Funktionensysteme für die Transformation verwendet werden. Dies wiederum sollte zu einer geringen Korrelation der einzelnen Merkmale untereinander führen, und damit zu einer verbesserten Klassifizierbarkeit.

ank:	89.8 %	bar:	94.5 %	uts:	85.9 %
abr:	75.9 %	dpo:	90.6 %	vdm:	81.3 %
all:	72.8 %	frs:	59.1 %	jmr:	56.5 %
$\emptyset = 79.7\ \%$					

Tabelle 3.6: Erkennungsergebnis - *spatial unterabgetastete Bitmap* mit $F_{BM} = 30 \times 30$ und $N_m = 9$

ank:	86.6 %	bar:	90.1 %	uts:	83.0 %
abr:	73.6 %	dpo:	86.6 %	vdm:	75.4 %
all:	63.6 %	frs:	52.3 %	jmr:	54.4 %
$\emptyset = 75.6\,\%$ [N=4134]					

Tabelle 3.7: Erkennungsergebnis - *DCT der Bitmap* mit $F_{BM} = 30$ und $N_m = 9$

ank:	88.7 %	bar:	92.5 %	uts:	89.3 %
abr:	82.2 %	dpo:	85.6 %	vdm:	80.8 %
all:	74.9 %	frs:	61.4 %	jmr:	59.2 %
$\emptyset = 79.3\,\%$ [N=4134]					

Tabelle 3.8: Erkennungsergebnis - *Walsh-Transformation* mit $F_{BM} = 30$ und $N_m = 9$

3.3.3 Merkmalskombination

Nach der separaten Untersuchung der verschiedenen Merkmale scheint eine Merkmals-Kombination zur Steigerung der Erkennungsgenauigkeit opportun.
Da die Merkmalsextraktion eine der unteren Stufen in der gesamten Verarbeitungskette darstellt, ist es sinnvoll einen Frühindikator zur Bewertung verschiedener Verfahren heranzuziehen. Mit solchen Frühindikatoren ließe sich für bestimmte Merkmale eine Vorauswahl treffen. Von der Simulation, einschließlich Modellierung und Tests, könnten so weniger geeignete Merkmale bereits im Vorfeld ausscheiden.
Der Vorteil eines solchen Vorgehens wird insbesondere bei der Frage nach der günstigsten Merkmalskombination ersichtlich. Während die hier betrachteten acht Einzelmerkmale noch konsequent getestet werden konnten, scheint dies für $8 \cdot (8-1)/2$ Merkmalskombinationen nicht mehr sinnvoll. Allein ein Testdurchlauf für ein Merkmal kann bei vorhandenen Modellen bereits einige Tage an CPU-Zeit in Anspruch nehmen. Unter diesen Randbedingungen scheidet die vollständige Kombination verschiedener Merkmalstypen allein schon aus Rechenzeitgründen aus.
In der Verarbeitungskette Vorverarbeitung, Merkmalsextraktion, Merkmalsquantisierung, Modellierung und Test sind die exakten Erkennungsergebnisse natürlich erst nach dem Test verfügbar. Nach der Modellierung ist bereits die mittlere Auftrittswahrscheinlichkeit der Trainingsdaten unter der Voraussetzung gegebener Modelle bekannt. Nach der Merkmalsquantisierung wiederum, kann die Transinformation der quantisierten Trainingsdaten bestimmt werden. Vereinfacht ausgedrückt gibt die Transinformation das Maß an Information wieder, welches sich effektiv über die Verarbeitungskette an den Erkenner übermitteln läßt. Wie dem nachfolgenden Kapitel weiterhin entnommen werden kann, ist die Berechnung der Transinformation auch für die Multi-Codebuch-Technik möglich. Dabei wird neben der Einzeltransinformation der jeweiligen Merkmalsströme auch deren Korrelation, bzw. die

Trajektorienmerkmale	$I(S,Y)$	$\log P(Y\|S)$
Sinus- und Cosinuswinkel	0.5264 bit	-2.5677
DCT	1.0142 bit	-2.663
Bézierkurve	1.0851 bit	-2.571
B-Spline-Kurve	0.9364 bit	-2.761
Hauptachsentransformation($F = 16$)	0.9225 bit	-2.702
DFT	0.7240 bit	-3.276
Polynom dritten Grades	0.5527 bit	-3.264
Bitmap-Merkmale		
Unterabtastung der Bitmap	0.7391 bit	-3.241
DCT der Bitmap	0.6901 bit	-3.255
Walsh-Transformation	0.6959 bit	-3.251

Tabelle 3.9: MMI und log-Likelihood von Einzelmerkmalen

Redundanz der Merkmale einbezogen. Die Verwendung der gemeinsamen Transinformation für multiple Merkmale stellt eine frühestmögliche Abschätzung für die Eignung der Merkmalskombinationen dar, da dieses Bewertungsmaß unmittelbar nach der Vektorquantisierung bestimmt werden kann.

Tab. 3.9 zeigt die Transinformation $I(S,Y)$ und die logarithmierte Auftrittswahrscheinlichkeit der Trainingsdaten. Überwiegend lassen sich innerhalb einer Merkmalsgruppe gute Übereinstimmungen zwischen Transinformation, log-Likelihood und letztendlicher Er-

Verfahren mit $F = 11$	*DCT auf Bitmap* $I(S,Y)$=0.6959	*Walsh-Transformation* $I(S,Y)$=0.6901	*Grauwertbestimmung von Bitmapteilen* $I(S,Y)$=0.7391
Sinus- und Cosinuswinkel $I(S,Y)$=1.0871	1.3747	1.3698	1.3668
DCT $I(S,Y)$=1.0408	1.3216	1.3135	1.3219
Bézierkurve $I(S,Y)$=1.0851	1.3640	1.3605	1.3549
B-Spline-Kurve $I(S,Y)$=0.9364	1.2690	1.2752	1.1.2727
Hauptachsentransformation ($F = 16$), $I(S,Y)$=0.9225	1.2348	1.2267	1.2290

Tabelle 3.10: Gesamt-Transinformation $I(S,Y^1,Y^2)$ in bit für Kombinationen aus Bitmap- und Trajektorienmerkmalen

	DCT der Bitmap $I(S,Y) = 0.6959$	*unterabgetastete BM* $I(S,Y) = 0.7391$
DCT der BM $I(S,Y) = 0.6959$	-	0.8085
Walsh-Transf. $I(S,Y) = 0.7054$	0.7094	0.7783

Tabelle 3.11: Gesamt-Transinformation $I(S,Y^1,Y^2)$ in bit für zwei Bitmap-Merkmale

	Bézierkurve $I(S,Y) = 1.0851$	*DCT* $I(S,Y) = 1.0408$
Bézierkurve $I(S,Y) = 1.0851$	-	0.9705
KLT $I(S,Y) = 0.9225$	0.9776 bit	0.9024

Tabelle 3.12: Gesamt-Transinformation $I(S,Y^1,Y^2)$ in bit für zwei Trajektorienmerkmale (F=11)

kennungsrate feststellen. Zum Vergleich sind in Tab. 3.9 weiterhin die Werte für die DFT und die Polynomapproximation aufgeführt. Aufgrund der grundsätzlich unterschiedlichen Extraktionsprinzipien der Trajektorien- und Bitmap-Merkmale ist eine maximale Komplementärwirkung bei der Kombination dieser beiden Merkmalstypen zu erwarten. Tab. 3.10 zeigt dazu die gemeinsame Transinformation der Zweier-Kombinationen von Trajektorien- und Bitmap-Merkmalen.
Zum Vergleich ist in Tab. 3.11 die Kombination verschiedener Bitmap-Merkmale zusammengefasst. Der Gewinn liegt hier, wie auch bei der Kombination von Trajektorienmerkmalen in Tab. 3.12 deutlich niedriger. Am Beispiel der Kombination von DCT-Trajektorienmerkmalen ist regelmäßig sogar ein nicht unerheblicher Transinformationsverlust festzustellen.

Die Kombination von Bitmap- und Trajektorienmerkmalen ist demnach die Wahl der Mittel. Es zeigt sich weiterhin, dass eine Dreier-Kombination wegen der offensichtlichen Red-

	DCT der Bitmap	*Walsh-Transformation*	*Grauwertbestimmung von Bitmapteilen (X)*
Sinus- und Cosinuswinkel	85.6 %	83.5 %	85.5 %
DCT	85.1 %	85.6 %	85.0 %
Bézierkurve	**86.2 %**	86.0 %	85.8 %

Tabelle 3.13: Erkennungsraten kombinierter Merkmale

undanz innerhalb einer Merkmalsgruppe keine weiteren Voreile bringt. Speziell läßt sich aus Tab. 3.10 ableiten, dass sich die Kettenkodierung (Sinus- und Cosinuswinkel), sowie die DCT der Trajektorie und die Bézierkurve gut mit den Bitmap-Merkmalen kombinieren läßt. Die Ergebnisse dieser Kombination sind abschließend in Tab. 3.13 dargestellt.
Hier zeigt sich, dass die Kombination aus DCT der Bitmap mit den Bézier-Merkmalen klar das beste Ergebnis liefert, was wiederum die Korrelation zur gemeinsamen Transinformation (Tab. 3.10) bestätigt.

3.4 Kapitelzusammenfassung

In diesem Kapitel wurden verschiedene Verfahren zur Merkmalsextraktion in Online-Handschrifterkennungssystemen präsentiert. Die Verfahren stützen sich auf zwei unterschiedliche Prinzipien der Merkmalsentnahme. Die erste Gruppe von Verfahren basiert auf der Merkmalsextraktion aus einem Fenster der Stifttrajektorie. Die zweite Gruppe nutzt die Information aus einer gleitenden Bitmap, wodurch sich zusätzlich die Möglichkeit eröffnet, die geometrische Umgebung zu erfassen. Die Erfassung der Umgebungsinformation ist dabei nicht auf ein zeitlich begrenztes Fenster beschränkt.
Speziell die Extraktion der Trajektorienmerkmale wurde unter verschiedenen Blickwinkeln betrieben. Neben einer parametrischen Kurvenbeschreibung wurde die Nutzung spektraler, geometrischer und statistischer Eigenschaften für eine kompakte Beschreibungsform untersucht.
Zur Auswahl und Kombination der Merkmale wurden neben der exakten Erkennungsrate weitere Bewertungsmaße - insbesondere die Transinformation - herangezogen, mit der eine sinnvolle Vorauswahl der Merkmale getroffen werden konnte.
Schließlich zeigte sich eine Kombination aus diskreter Cosinus Transformation der Bitmap mit den Bézierkurven als ideale Kombination mit maximaler Erkennungsrate, wobei die Kontrollpunkte der approximierenden Bézierkurve als Merkmale Verwendung fanden. Diese Merkmalskombination sollte demnach fortan eingesetzt werden.
Da im Rahmen dieser Arbeit einige Entwicklungen parallel vorangetrieben wurden, gilt insbesondere für die nachfolgend beschriebenen höheren Systemebenen, dass die Kettenkodierung zusammen mit der unterabgetasteten Bitmap weiterhin als Referenzverfahren eingesetzt wird. Dies stellt die Vergleichbarkeit zu früheren Ergebnissen sicher.

Kapitel 4

Modellierung

Bei der Frage nach einem geeigneten Paradigma zur Modellierung der Handschrift ließen sich theoretisch alle bisher genutzten Prinzipien der Mustererkennung untersuchen. Dies würde einerseits sowohl regelbasierte/heuristische Methoden berühren, wie auch Verfahren, mit denen sich bestimmte Eigenschaften aus Beispielen erlernen lassen, wie z. B. neuronale Netze oder Hidden Markov Modelle.
Abgesehen von eher akademischen Problemstellungen der Mustererkennung scheint sich zunehmend die Erkenntnis durchzusetzen, dass sich die sog. lernenden Methoden - insbesondere bei komplexen Mustererkennungsproblemen - im Vergleich zu regelbasierten Ansätzen durch eine deutlich höhere Robustheit und Generalisierungsfähigkeit auszeichnen.
Als ’komplex’ ist im Zusammenhang mit einer automatischen Verarbeitung durchaus die menschliche Sprache, sowohl in gesprochener als auch in geschriebener Form einzustufen. Diese speziellen Mustererkennungsprobleme weisen darüber hinaus noch eine besondere Eigenschaft auf: Es handelt sich dabei um dynamische Muster, bei denen zunächst unbekannt ist, wie groß die Gesamtlänge des beobachteten Musters ausfällt, oder wann genau welche Merkmale auftreten und wie stark diese dann ausgeprägt sind. Verschleifungseffekte mit einer einhergehenden zeitlichen Unschärfe der einzelnen Klassengrenzen erschweren die Situation zudem.
Hidden Markov Modelle scheinen genau für solche Problemstellungen ein probates Paradigma darzustellen, da sie über wichtige Eigenschaften verfügen, wie:

- automatische Lernfähigkeit anhand vorgegebener Trainingsbeispiele,
- hervorragende Eigenschaften zur Modellierung dynamischer Muster und
- Möglichkeiten zur integrierten Erkennung und Segmentierung.

Diese Vorteile von Hidden Markov Modellen ermöglichen über die oben genannten Kerneinsatzgebiete weitere interessante Anwendungsmöglichkeiten, wie z. B. im Bereich der

Dokumentenverarbeitung [Bra00, Bra99b, Bra99a], der Unterschriftenverifikation [Rig98b], der automatischen Indexierung von Videosequenzen [Eic97a], der Videosequenz-Erkennung [Rig98a, Eic98, Eic97b] oder aber der Erkennung handskizzierter Piktogramme [Mü98, Mü99], die für eine inhaltsorientierte Bilddatenbankabfrage verwendet werden können [1].
Als genereller Ansatz für die Online-Handschrifterkennung bieten sich im besonderen HMM auf Grund der genannten Vorteile und Vielseitigkeiten an. Innerhalb dieses Rahmens eröffnen sich jedoch wiederum zahlreiche Realisierungsmöglichkeiten. Die nachfolgenden Abschnitte sollen zunächst eine Antwort darauf liefern, welche grundlegende Modellierungsform im Bereich der Online-Handschrifterkennung vorteilhaft ist.

4.1 Vergleich zwischen diskreten und kontinuierlichen Modellen

Neben anderen Faktoren hat die Wahl der Modellierungsform einen entscheidenden Einfluß auf die Erkennungsgenauigkeit des Systems. In der Spracherkennung scheinen sich nach einigen z. T. widersprüchlichen Ergebnissen verstärkt kontinuierliche Modelle durchzusetzen. So wurde in einigen Arbeiten dokumentiert [Bah81, Bro87], dass diskrete Spracherkennungssysteme im Vergleich zu kontinuierlichen Systemen höhere Erkennungsraten zeigen, während in [Rab85, VNG87] gegensätzliche Ergebnisse belegt werden. Die teilweise stark abweichenden Ergebnisse in den referenzierten Veröffentlichungen zeigen, dass eine Untersuchung der Frage nach der geeigneten Modellierungsform individuell erfolgen sollte. Abb. 4.1 verdeutlicht die wesentlichen Unterschiede zwischen diskreter und kontinuierlicher Modellierung. Bei kontinuierlichen HMM werden die Ausgabeverteilungen in parametrischer Form ausgeführt. Unter der Annahme einer Gauß-förmigen Verteilung der Merkmale kommen i. d. R. auch Gauß'sche Mischverteilungen zum Einsatz.

Bei diskreten Systemen wird der (im Beispiel zweidimensionale) Merkmalsraum zunächst mittels eines Vektorquantisierers (VQ) in Partitionen eingeteilt. Die Merkmalsvektoren $\vec{x}$ werden dazu anhand eines Abstandsmaßes dem nächstliegenden Codebuchvektor des Vektorquantisierers zugeordnet. Ein Codebuch der Größe N_C führt folglich zu N_C verschiedenen Partitionen (Voronoi-Zellen) des Merkmalsraumes, die bei der Verarbeitung i-dimensionaler Merkmalsvektoren i-dimensionale konvexe Polytope bilden. Die diskrete Ausgabeverteilung $b_s(n)$ des Zustandes s modelliert damit die Auftrittswahrscheinlichkeit der in der n-ten VQ-Partition zusammengefassten Merkmalsvektoren. Damit sind diskrete Modelle unabhängig von Annahmen bezüglich der Merkmalsverteilung. Als Standard-Lösung werden in diskreten Systemen zur Partitionierung des Merkmalsraumes unüberwachte, selbstorganisierende,

[1] Siehe dazu auch http://www.fb9-ti.uni-duisburg.de/rotdemo.html

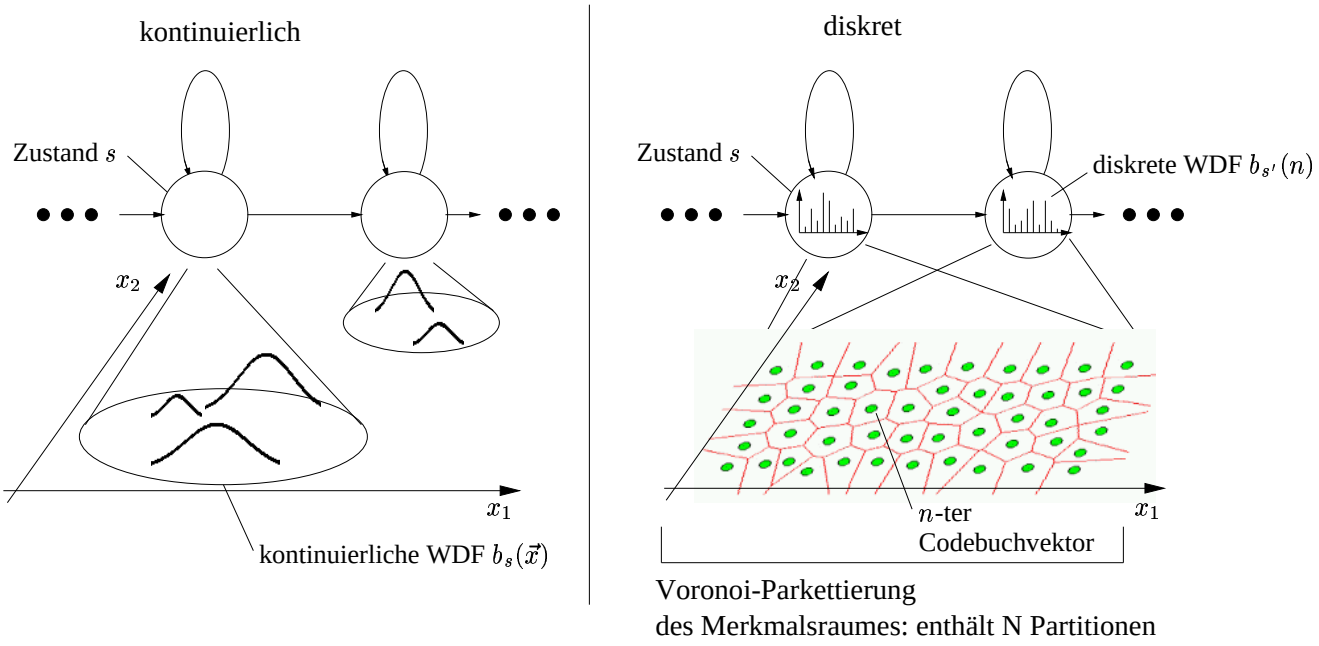


Abbildung 4.1: Prinzipielle Darstellung kontinuierlicher und diskreter HMM

ggf. hierarchische Vektorquantisierer [Lee89] eingesetzt. Als weiterer Vorteil diskreter Modelle gilt die effiziente Berechnung der Ausgabewahrscheinlichkeiten, was insbesondere bei der Erkennung zu deutlichen Geschwindigkeitsvorteilen führen kann. Bei gegebenem VQ-Index n läßt sich die Berechnung von $p(n|s)$ auf einen einfachen Speicherzugriff auf die als Vektor abgelegten Wahrscheinlichkeitsdichtefunktionen realisieren.
Beiden Modellierungsformen ist allerdings die generelle Vorgehensweise bei der Modellbildung gemein. Ausgehend von allgemeinen Modellprototypen, mit denen die Modellstruktur definiert wird, werden aus einigen vorsegmentierten Zeichenbeispielen initiale Verteilungen und Übergangswahrscheinlichkeiten ermittelt. Experimentell konnte als optimale Struktur das Links-Rechts-Modell mit 12 Zuständen pro Graphem (bzw. Zeichen) bestimmt werden. Ein Links-Rechts-Modell ist mit Selbsttransitionen, sowie mit Übergängen zum Nachfolgezustand ausgestattet.
Zur Berechnung initialer Verteilungen werden in einer ersten Iteration die vorsegmentierten Beobachtungssequenzen eines Graphems in 12 gleich lange Abschnitte unterteilt. Diese 12 gleich langen aufeinander folgenden Abschnitte werden den 12 in einem Modell enthaltenen aufeinander folgenden Zuständen zugeordnet. Für diese Initialisierungsstufe werden die exemplarisch im Anhang B, in Abb. B.1 gezeigten Trainingsbeispiele verwendet. Anhand dieser zunächst willkürlichen Zuordnung lassen sich erste Verteilungsparameter berechnen. Diese Verteilungsparameter dienen in einer nächsten Iteration dazu, mit einem sog. Viterbi-Alignment die vorhandenen Trainingsdaten den Zuständen zuzuordnen, woraus sich die Parameter mit jeder Iteration verfeinert schätzen lassen. Abb. 4.2 zeigt die Zustandssegmentie-

rung einer Trainingssequenz in der 1. Iteration der Initialisierung und nach erfolgtem Training. Es zeigt sich dabei, dass sich trotz willkürlicher Zuordnung gleich langer Sequenzen zwecks Initialisierung, das Alignment im Laufe des Trainings teilweise deutlich verschieben kann. Es bilden sich Zustände für verschiedene, unterschiedlich lange Merkmalssequenzen heraus.

Im Anschluß an das Viterbi-Training werden die so initialisierten Modelle mit dem Baum-

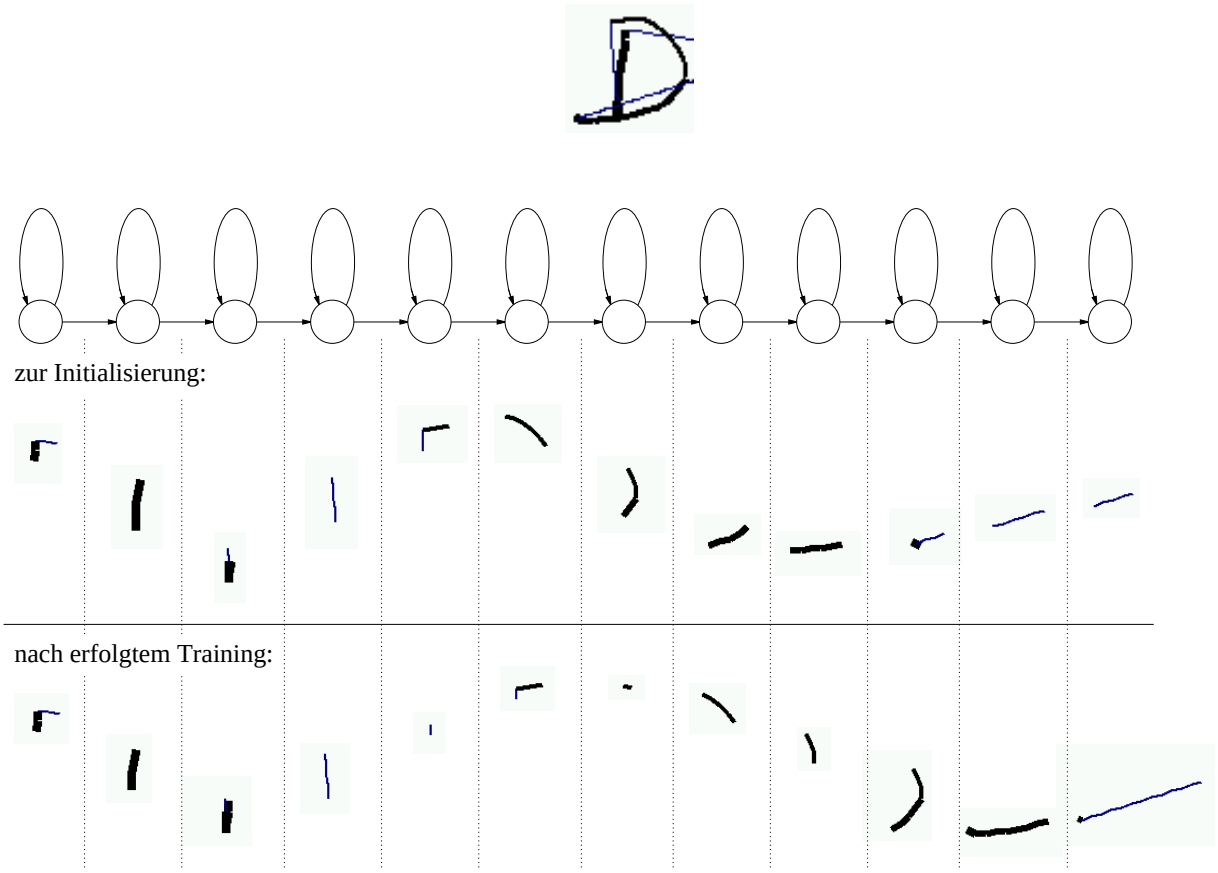


Abbildung 4.2: Alignment nach Initialisierung und nach vollständigem Training

Welch-Algorithmus weiter optimiert. Diese Optimierung läßt sich in zwei Stufen einteilen. Die erste Stufe basiert wiederum auf den schon für das Viterbi-Training vorsegmentierten Daten. In der zweiten Stufe werden dann ganze Wörter oder Sätze für das Training herangezogen. Beispiele dazu sind in Abb. B.2 gezeigt. Im Gegensatz zu dem Viterbi-Training werden die Trainingsmuster den Zuständen nicht mehr in deterministischer Weise zugeordnet, sondern mit gewissen Wahrscheinlichkeitswerten.

Grundsätzlich steckt hinter dem Gedanken der Parameter-Optimierung die Idee, die Wahrscheinlichkeit zu maximieren, dass die gegebene Trainingsmenge X von den vorhandenen Modellen λ generiert wurde. Dieses - auch als Likelihood bekannte - Gütemaß $p(X|\lambda)$ läßt sich bedauerlicherweise weder analytisch, noch direkt maximieren. Statt dessen wurde in

[Bau69] vorgeschlagen, die Kullback-Leibler Distanz

$$Q(\lambda, \lambda^*) = \sum_S p(S|X, \lambda) \log p(X, S|\lambda^*) \tag{4.1}$$

zwischen einem Ausgangsmodell λ und einem optimierten Modell λ^* zu maximieren. Die Optimierungsgleichungen der Modellparameter (sowohl der Ausgabeverteilungen wie auch der Übergangswahrscheinlichkeiten) lassen sich direkt aus Glg. 4.1 ableiten. Wird ein Parametersatz λ^* berechnet, der Glg. 4.1 maximiert, läßt sich weiterhin zeigen, dass

$$P(X|\lambda^*) \geq P(X|\lambda) \tag{4.2}$$

gilt womit sich in iterativer Weise die HMM hinsichtlich deren Likelihood optimieren lassen. Weitergehende Grundlagen zur Theorie der Hidden Markov Modelle können [ST95] entnommen werden. Eine sehr gute und verständliche Zusammenfassung ist in [Rab89] zu finden. Im weiteren sollen grundlegende Aspekte zu diesem Thema lediglich im Bedarfsfall betrachtet werden.

4.2 Verwendung neuronaler Netze zur Merkmalsquantisierung

Der Vergleich der Eigenschaften kontinuierlicher und diskreter HMM zeigt einerseits, dass es vorteilhaft ist, auf bestimmte Verteilungsannahmen der Merkmale zu verzichten. Andererseits stellt sich durch die Vektorquantisierung immer auch ein Quantisierungsfehler ein. Ein weiterer Schwachpunkt der bekannten diskreten Standardverfahren ist, dass der selbstorganisierende Vektorquantisierer ohne Klasseninformation des Merkmalsvektors trainiert wird. Damit stellt der Vektorquantisierer eine weitere unabhängige, verlustbehaftete Verarbeitungsstufe im Gesamtsystem dar. Die wesentlichen Nachteile diskreter Standard-Systeme sind in den folgenden Punkten zusammenzufassen:

- Die Merkmalskompression durch vorangehende Vektorquantisierung ist stets verlustbehaftet.
- Die Berechnung des Codebuches, welches später den Merkmalsraum partitioniert, wird bei den gängigen Standardverfahren (wie z. B. k-means) in selbstorganisierender Weise durchgeführt. D. h. es wird keine Klasseninformationen ausgenutzt.

Um die Schwachpunkte selbstorganisierender Vektorquantisierer zumindest teilweise zu umgehen, bieten sich neuronale Netze an. Eine umfassende Beschreibung zu gängigen

neuronalen Netzen und den zugrunde liegenden Paradigmen ist in [Zel94] gegeben. Die speziell für die Handschrifterkennung adaptierten Neuronalen Netztypen erlauben es, das Wissen über die Klassenzugehörigkeit von Merkmalsvektoren zu integrieren. Eine derartige Verknüpfung neuronaler Netze (NN) mit Hidden Markov Modellen (HMM) führt zu einer speziellen Variante hybrider NN/HMM-Systeme [Rig94]. Verschiedene neuronale Paradigmen wurden bereits zur Verwendung in hybriden Spracherkennungssystemen untersucht [Neu99, Rot00]. Vergleichbare Untersuchungen zur Online-Handschrifterkennung existierten bisher nicht [Rig96a, Rig98c]. Während in [Neu99] die synchrone Optimierung mehrerer Codebücher für Perzeptron-Strukturen dokumentiert ist, sollen die nachfolgenden Abschnitte insbesondere das sequentielle wie auch das multiple Training von nN- (nächster-Nachbar-) Vektorquantisierern und die damit erzielten Ergebnisse beschreiben [Kos98a, Rig99, Rig98d] .

4.2.1 Die Transinformation als Gütemaß für das Training neuronaler Netze

Bei eingehender Betrachtung läßt sich bei Verwendung eines selbstorganisierenden Vektorquantisierers die Berechnung der Wahrscheinlichkeit für einen Merkmalsvektor $\vec{x}$ in einem HMM-Zustand s eines diskreten Systems aus der Wahrscheinlichkeit der VQ-Partition y_n im Zustand s ableiten:

$$p(\vec{x}|s) = p(y_n|s) \tag{4.3}$$

Das Codebuch des diskreten Systems besteht dabei aus N_C Einträgen y_n mit $n = 1, \cdots, N_C$. Der Vektorquantisierer ordnet einem Merkmalsvektor $\vec{x}$ denjenigen Index y_n des VQ-Prototypen zu, dessen Abstand zu dem Merkmalsvektor am geringsten ist (s. Abb. 4.1)
Im diskreten, wie im hybriden System wird durch den Vektorquantisierer bei Präsentation des Merkmalsvektors $\vec{x}$ ein VQ-Prototypindex y_n erzeugt: $\vec{x} \xrightarrow{VQ} y_n$. Der statistische Zusammenhang zwischen dem Merkmalsvektor $\vec{x}$ und dem VQ-Index y_n kann durch die bedingte Wahrscheinlichkeit $p(y_n|\vec{x})$ beschrieben werden. Mit Hilfe des Satzes von Bayes

$$p(\vec{x}, y_n) = p(\vec{x}) \cdot p(y_n|\vec{x}) = p(y_n) \cdot p(\vec{x}|y_n) \tag{4.4}$$

kann die Wahrscheinlichkeit $p(\vec{x})$ berechnet werden:

$$p(\vec{x}) = \frac{p(\vec{x}|y_n)}{p(y_n|\vec{x})} \cdot p(y_n). \tag{4.5}$$

Glg. (4.5) beschreibt somit den VQ-Prozeß, der unabhängig vom HMM-Zustand s ist. Fügt man nun die Bedingung eines bestimmten Zustandes s ein, ergibt sich für die bedingte Auftrittswahrscheinlichkeit:

$$p(\vec{x}|s) = \frac{p(\vec{x}|y_n)}{p(y_n|\vec{x})} \cdot p(y_n|s). \tag{4.6}$$

Mit Hilfe des nach $p(\vec{x}|y_n)$ umgestellten Satzes von Bayes (Glg. (4.4)) kann Glg. (4.6) schließlich umgeformt werden zu

$$p(\vec{x}|s) = \frac{p(\vec{x})}{p(y_n)} \cdot p(y_n|s). \tag{4.7}$$

Unter der Annahme, dass der Quotient $p(\vec{x})/p(y_n)$ nun bekannt ist läßt sich die Wahrscheinlichkeit des Merkmalsvektors $p(\vec{x})$ aus der Wahrscheinlichkeit $p(y_n)$ des VQ-Index y_n berechnen. Zur Verdeutlichung sei an dieser Stelle angemerkt, dass nach Glg. (4.3) in einem diskreten System die Identität der Wahrscheinlichkeiten $p(\vec{x})$ und $p(y_n)$ in Glg. (4.7) angenommen wird. Dies wiederum entspricht der - zweifellos idealisierten - Annahme einer verlustlosen Vektorquantisierung.

Ausgehend von Gleichung (4.7) kann nun ein Algorithmus abgeleitet werden, der zu einem optimierten Vektorquantisierer führt, wobei das Gesamtsystem (VQ *und* HMMs) die Vorteile diskreter und kontinuierlicher Modelle vereinen. Analog zur ansonsten üblichen Minimierung eines Fehlermaßes basiert das Training dieser Vektorquantisierer auf der Maximierung eines informationstheoretischen Gütemaßes. In der Gesamtheit ergibt sich somit ein hybrides HMM/NN-System.

Die Parameterschätzung von HMM-Systemen basiert zumeist auf dem sog. *Maximum Likelihood* (ML) Kriterium. Die Likelihood $P(X|S)$ eines Systems wird anhand seiner aktuellen Parameter λ (Ausgabeverteilungen in den HMM-Zuständen) über alle K Abtastwerte der Trainingsmenge aufsummiert:

$$p(X|S) = \sum_{k=1}^{K} \log p(\vec{x}(k)|s(k)). \tag{4.8}$$

Zur Berechnung bedient man sich aus numerischen Erwägungen i. d. R. der logarithmierten Form. Die erforderliche Zuordnung der einzelnen Merkmalsvektoren $\vec{x}(k)$ zu den entsprechenden HMM-Zuständen $s(k)$ kann z. B. automatisch mittels des *Viterbi-Alignments* bestimmt werden. Aus einem Parametersatz λ kann nun durch Anwendung von Gradientenverfahren in iterativer Weise ein optimierter Parametersatz $\hat{\lambda}$ berechnet werden:

$$\hat{\lambda} = \arg\max_{\lambda} \left\{ \frac{1}{K} \sum_{k=1}^{K} \log p(\vec{x}(k)|s(k)) \right\}. \tag{4.9}$$

Verwendet man weiterhin die statistischen Zusammenhänge des Vektorquantisierers aus Glg. (4.7) in Glg. (4.9), so können selbst die Parameter des Vektorquantisierers (Codebuchvektoren) λ_{VQ} in Übereinstimmung mit dem ML-Kriterium optimiert werden. Üblicherweise werden die Komponenten der Codebuchvektoren als *Gewichte* bezeichnet. Wird Glg. (4.7) in Glg. (4.9) eingesetzt, ergibt sich damit folgender Ausdruck für die Optimierung der Codebuchvektoren:

$$\hat{\lambda}_{VQ} = \arg\max_{\lambda_{VQ}} \frac{1}{K} \left\{ \sum_{k=1}^{K} \log p(\vec{x}(k)) - \sum_{k=1}^{K} \log p(y_n(k)) + \sum_{k=1}^{K} \log p(y_n(k)|s(k)) \right\}. \tag{4.10}$$

Als *Informationsgehalt* eines Zeichens bezeichnet man den negativen Logarithmus der Auftrittswahrscheinlichkeit dieses Zeichens. Die einzelnen Terme aus Glg. (4.10) sind die Erwartungswerte der entsprechenden Informationsgehalte und sind demnach auch bekannt unter dem Begriff der *Entropie* H. Damit läßt sich obige Gleichung umschreiben in

$$\hat{\lambda}_{VQ} = \arg\max_{\lambda_{VQ}}\{-H(X) + H(Y) - H(Y|S)\}. \tag{4.11}$$

Da die Merkmalsvektoren X auf der Eingangsseite des neuronalen Netzes anliegen, ist die Entropie $H(X)$ unabhängig von dem Codebuch des Vektorquantisierers, sodass folglich die Differenz $H(Y) - H(Y|S)$ zu maximieren ist. Das zu maximierende Gütemaß des neuronalen Vektorquantisierers ist damit die Transinformation I und lautet

$$I(Y,S) = H(Y) - H(Y|S) = H(S) - H(S|Y). \tag{4.12}$$

Aus Glg. (4.12) und anhand des informationstheoretischen Kanalmodells (Abb. 4.3) wird deutlich, dass die Maximierung von $I(S,Y)$ ebenso einer Maximierung von $H(S) - H(S|Y)$ entspricht. Da der Wert für $H(S)$ durch das Viterbi-Alignment der Trainingsdaten vorgegeben ist, reduziert sich die Maximierung der Transinformation $I(S,Y)$ deshalb auf die Minimierung der *Äquivokation* $H(S|Y)$.

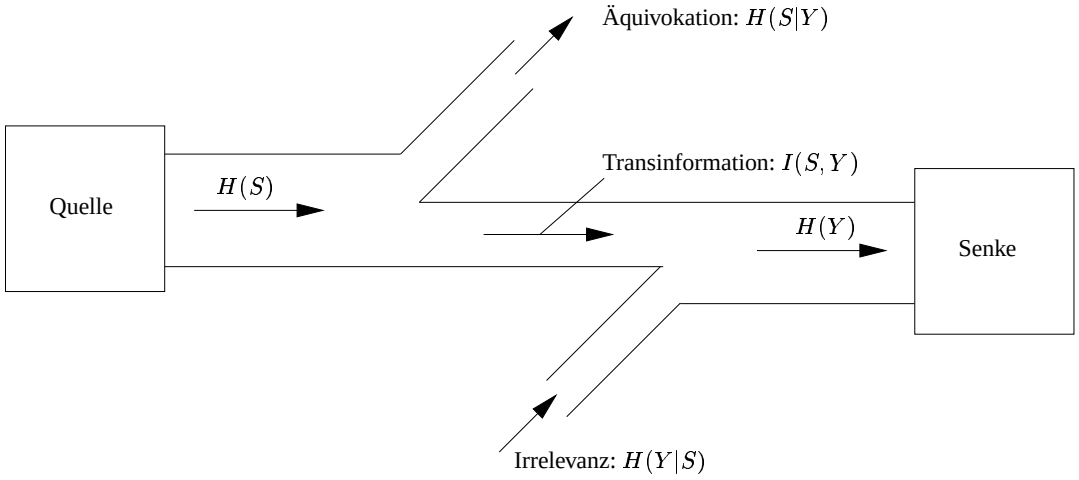


Abbildung 4.3: Informationstheoretisches Kanalmodell (Berger'sches Diagramm)

4.2.2 Automatische Handschrifterkennung - ein Nachrichtenübertragungssystem

Das Kanalmodell erlaubt darüber hinaus eine interessante systemorientierte Betrachtung der zuvor beschriebenen Theorie, die am Beispiel der Handschrifterkennung nachfolgend erläutert wird. Die wesentlichen Komponenten des Kanalmodells sind die Nachrichtenquelle, die Nachrichtensenke (Empfänger), und der Übertragungskanal, welcher Quelle und

Senke miteinander verbindet. Im Falle der Schrifterkennung repräsentiert die Quelle eine bestimmte Zeichen- bzw. Zustandsfolge S (basierend auf dem zu schreibenden Text), die von einem Schreiber generiert wird. Im Falle der Online-Handschrifterkennung werden diese geometrischen Signale dann durch eine Reihe von Transformationen in die VQ-Folge Y umgeformt. Im wesentlichen handelt es sich hierbei um die Vorverarbeitungsschritte, die Merkmalsextraktion und die Vektorquantisierung. Die Funktionen der Schrifterzeugung und der anschließenden Transformationen lassen sich entsprechend des Kanalmodells in Abb. 4.3 dem Übertragungskanal selbst zuordnen. Die Senke umfasst schließlich die Funktion des Erkenners, der aufgrund der empfangenen vektorquantisierten Daten mit Hilfe der Hidden Markov Modelle die Erkennung durchführt und die VQ-Daten wieder in eine Textform transformiert.

Ein Beobachter am Kanaleingang sieht demnach eine Nachrichtenquelle mit einem mittleren Informationsgehalt $H(S)$ (vergl. Abb. 4.3). Davon geht - bedingt durch verschiedene Verarbeitungsstufen - der Anteil $H(S|Y)$ verloren. Die bedingte Entropie $H(S|Y)$ ist der mittlere Informationsgehalt je Zeichen, den das Zeichen, bzw. der Zustand s noch zusätzlich liefern würde, nachdem y empfangen wurde. Da der Empfänger jedoch nur Zugriff auf die Zeichen Y hat, ist - wie bereits oben erwähnt - $H(S|Y)$ der Verlustanteil, der in erster Linie durch den Quantisierungsfehler entsteht. Da die übrigen Vorverarbeitungsstufen ohnehin fest vorgegeben sind, reduziert sich die Optimierung auf die Minimierung dieses Informationsverlustes. Die Transinformation $I(S,Y)$ ist der verbleibende Teil der Quelleninformation, der über den Kanal übertragen wird und den Empfänger erreicht. Ggf. wird der Transinformation bei der Übertragung noch eine gewisse Fehlinformation $H(Y|S)$ hinzugefügt (z. B. durch Kanalstörungen, Rauschen), die in dem mittleren Informationsgehalt $H(Y)$ resultiert. Der Fehlinformationsanteil $H(Y|S)$ stammt also nicht aus der Nachrichtenquelle, sondern ist der mittlere Informationsanteil, den ein Zeichen y noch liefern würde, nachdem ein gesendetes Zeichen s bekannt ist. Dem Beobachter am Kanalausgang erscheint schließlich eine Nachrichtenquelle mit der Entropie $H(Y)$. Die Aufgabe des Erkenners ist es nun, aufgrund der empfangenen Daten Y einen (möglichst) fehlerfreien Rückschluß auf die gesendeten Daten (Zustands- bzw. Graphemfolge S) zu ziehen.

Realisierung verteilter Architekturen

Das hier vorgestellte Verfahren ermöglicht die Realisierung komplexer Mensch-Maschine-Schnittstellen (Handschrift- oder auch Spracherkennung) auf Plattformen mit geringer Rechenleistung und relativ schmalbandiger Netz-Anbindung (Mobiltelefone, PDA) [Rig00]. Dies läßt sich durch eine Übertragung der relevanten Merkmale in hoch-komprimierter Form und einer Erkennung dieser komprimierten Daten auf CPU-Servern realisieren, die z. B. ein Mobilfunkanbieter bereithält. Das Erkennungsergebnis wird anschließend wieder zum Endgerät zurückgesendet und kann dort je nach gewünschter Anwendung als Befehl

interpretiert oder als Text-Passage weiter verarbeitet werden.
Wenn die rechenaufwendige Erkennung auf dem Endgerät nicht möglich ist, bietet sich die Auslagerung des eigentlichen Erkennungsprozesses an. Dies ließe sich über die Komprimierung der Merkmalsvektoren auf dem Endgerät und die Übertragung dieser Merkmale an einen leistungsfähigen Rechner ermöglichen, auf dem der eigentliche Erkennungsprozeß stattfindet. Zum Zweck der Merkmalskompression können Vektorquantisierer eingesetzt werden, die einen vieldimensionalen kontinuierlichen Merkmalsvektor auf eindimensionale diskrete Werte abbilden und so eine sehr hohe Kompressionsrate erzielen. Dies wiederum ermöglicht selbst die Nutzung schmalbandigster Übertragungskanäle. Abb. 4.4 gibt einen

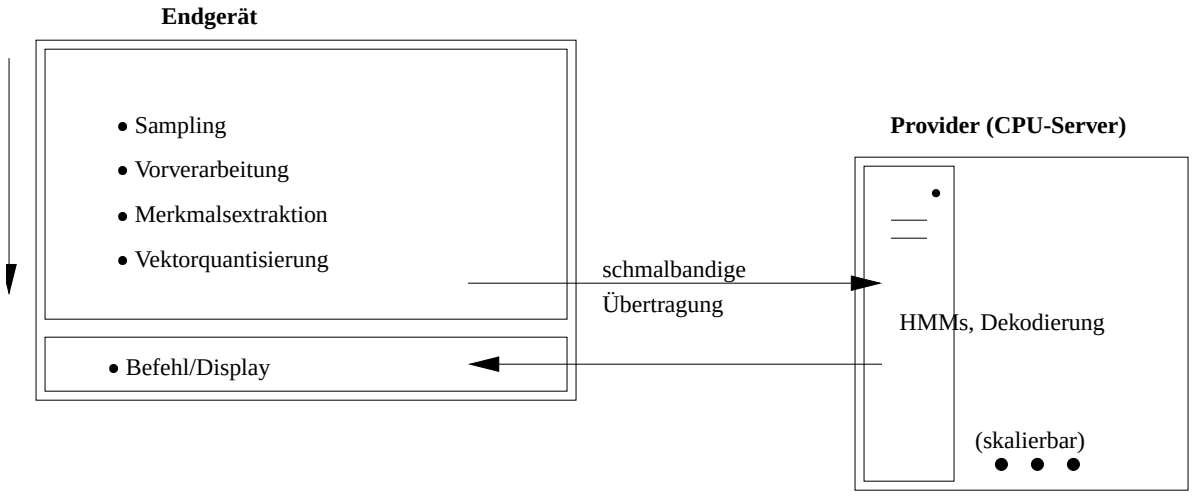


Abbildung 4.4: Übersicht eines verteilten Erkennungssystems

Überblick über eine mögliche verteilte Systemarchitektur. Auf der Benutzerseite werden dabei sämtliche vorbereitende Operationen durchgeführt, wie die Signalaufnahme, Vorverarbeitung, Merkmalsextraktion und Vektorquantisierung. Die so gewonnenen komprimierten Merkmale werden anschließend mit geringer Bandbreite an ein skalierbares System von CPU-Servern gesendet, auf denen der eigentliche, rechenaufwendige Erkennungsprozeß stattfindet. Das Erkennungsergebnis kann anschließend zum Benutzer übertragen werden, auf dessen Endgerät das Ergebnis dann als Textpassage oder als Befehl verwendet werden kann.
Ein wesentlicher Aspekt des hier beschriebenen hybriden Ansatzes ist der Zusammenhang zwischen der Vektorquantisierung und der Erkennung mittels Hidden Markov Modellen, da dies genau die Schnittstelle zwischen Endgerät und Erkenner in Abb. 4.4 (bzw. zwischen Sender und Empfänger in Abb. 4.3) darstellt. Dieser Zusammenhang soll im nachfolgenden Abschnitt beschrieben werden.

4.3 Hybrides NN/HMM Handschrifterkennungssystem

Im Detail läßt der hier vorgestellte theoretische Rahmen zahlreiche Realisierungsformen zu. So ist es durch entsprechende Modifikationen möglich, einen MMI-Vektorquantisierer mit diversen neuronalen Paradigmen zu realisieren [Neu99]. Hierbei kommen neben dem *nächster-Nachbar*-VQ (nN-VQ) auch Strukturen wie das MLP (*Multi-Layer-Perzeptron*), RBF- (Radiale Basisfunktionen-) Netze oder rekurrente Netze in Frage. Gemeinsam ist den verschiedenen Paradigmen eine generelle Arbeitsweise. Es wird ein Merkmalsvektor $\vec{x}(k)$ - ggf. zusammen mit einem gewissen zeitlichen Kontext $\vec{x}(k-k'),\cdots,\vec{x}(k),\cdots\vec{x}(k+k')$ an den Eingang des Netzes angelegt. Ausgangsseitig erzeugt das Netz mittels einer *winner-takes-all*-Schicht (WTA) einen diskreten VQ-Label y_n mit

$$y_n = y(\vec{x}) \tag{4.13}$$

und

$$n = \arg\max_i f_i(\vec{x}). \tag{4.14}$$

$f_i(\vec{x})$ ist dabei die interne Aktivierung des Ausgangsknotens i. Des weiteren ist es möglich einen Merkmalsvektor $\vec{x}$ in Z verschiedene Untervektoren $\vec{x}^{(z)}$ zu unterteilen. Bei dieser sog. Multi-Codebuchtechnik, wie sie in Abschnitt 4.3.2 beschrieben wird, können dann die einzelnen Codebücher simultan nach dem MMI-Prinzip trainiert werden, sodass das Training in einer maximierten Verbund-Transinformation $I(Y^{(1)},\cdots,Y^{(z)},S)$ resultiert.
Ähnlich viele Freiheitsgrade hinsichtlich des System-Designs ergeben sich bei der Auswahl der Klassen. Während bisher davon ausgegangen wurde, dass das Viterbi-Alignment die Trainingsdaten entsprechend den HMM-Zuständen s segmentiert, besteht ebenso die Möglichkeit die Klassenzugehörigkeiten entsprechend den Graphemgrenzen (Mono-, Bi- oder Trigrapheme) oder aber auch entsprechend der Wortgrenzen festzulegen. In der Praxis könnten so z. B. wortbasierte HMMs eingesetzt werden, die speziell bei Erkennungssystemen mit kleinen Wortschätzen Vorteile bieten. Außerdem ist hier zu unterstreichen, dass die Codebuchgröße unabhängig von der Anzahl der Klassen (Zustände, Grapheme, etc.) gewählt werden kann.
Ein allgemeines Problem bei Gradientenabstiegsverfahren stellen mögliche lokale Nebenminima, sowie flache Plateaus im Funktionsverlauf dar, die bei der Parameteroptimierung zu suboptimalen Lösungen führen können. Als Lösung bieten sich dazu aus der Neuroinformatik bekannte Verfahren an, wie z. B. *Momentum*, oder *RProp* (resilient propagation) [Zel94]. Bei den durchgeführten Experimenten zeigten speziell die RProp-Verfahren eine schnelle und gute Konvergenz.

4.3.1 MMI-Training einzelner Codebücher

Die grundlegende Idee des Trainings des neuronalen Netzes, bzw. die Optimierung der Gewichtungsvektoren, ist die Minimierung einer Bewertungsfunktion (Äquivokation $H(S|Y)$) mittels Gradientenabstieg [Neu99]. Die bedingte Entropie $H(S|Y)$ ist definiert mit

$$H(S|Y) = -\sum_{l=1}^{L}\sum_{n=1}^{N} p(s_l, y_n) \cdot \log p(s_l|y_n) \tag{4.15}$$

und läßt sich als Funktion von $p(s, y)$ umschreiben:

$$H(S|Y) = -\sum_{l=1}^{L}\sum_{n=1}^{N} p(s_l, y_n) \cdot \log \frac{p(s_l, y_n)}{\sum_{m=1}^{L} p(s_m, y_n)}. \tag{4.16}$$

Aus den Zustands-weise segmentierten Trainingsdaten läßt sich wiederum die Verbundwahrscheinlichkeit $p(s_l, y_n)$ abschätzen

$$p(s_l, y_n) \approx \frac{1}{K} \cdot \sum_{k=1}^{K} \delta_{s(k),s_l} \cdot o_n(k), \tag{4.17}$$

mit dem Kronecker-Delta δ. Da die WTA-Funktion aus Glg. (4.14), angewandt auf die Aktivierung der Ausgangsknoten nicht stetig differenzierbar ist, wird eine Soft-Max Funktion o eingeführt, welche die Maximumsuche unter den Ausgangsaktivierungen $f_n(\vec{x})$ aus Glg. (4.14) annähert durch

$$o_n(\vec{x}) = \frac{e^{C \cdot f_n(\vec{x})}}{\sum_{i=1}^{N} e^{C \cdot f_i(\vec{x})}}. \tag{4.18}$$

Bei hinreichend großer Wahl der Konstante C, kann nun die WTA-Funktion durch die Soft-Max-Funktion approximiert werden:

$$\lim_{C\to\infty} o_n(\vec{x}) = \begin{cases} 1 & \text{wenn } f_n(\vec{x}) > f_i(\vec{x}) \quad \forall \ 1 \le i \le N \ , \ n \ne j \\ 0 & \text{sonst} \end{cases} \tag{4.19}$$

Für die Einstellung eines bestimmten VQ-Parameters λ_{VQ} wird nachfolgend die partielle Ableitung $\partial H(S|Y)/\partial\lambda_{VQ}$ betrachtet. Unter Anwendung der verallgemeinerten Kettenregel ergibt sich damit

$$\frac{\partial H(S|Y)}{\partial \lambda_{VQ}} = \sum_{l=1}^{L}\sum_{n=1}^{N} \frac{\partial H(S|Y)}{\partial p(s_l, y_n)} \cdot \sum_{k=1}^{K} \frac{\partial p(s_l, y_n)}{\partial o_n(\vec{x}(k))} \cdot \sum_{i=1}^{N} \frac{\partial o_n(\vec{x}(k))}{\partial f_i(\vec{x}(k))} \cdot \frac{\partial f_i(\vec{x}(k))}{\partial \lambda_{VQ}}. \tag{4.20}$$

Nach der Bildung der partiellen Differentiale und einigen Umformungen ergibt sich mit der Einführung einer Hilfsfunktion h_i mit

$$h_i(\vec{x}(k)) = -\frac{C}{K} \cdot o_n(\vec{x}(k)) \cdot \left(\log p(s(k)|y_n) - \sum_{n=1}^{N} \log p(s(k)|y_n) \cdot o_n(\vec{x}(k)) \right), \tag{4.21}$$

der folgende Ausdruck für die Ableitung der Äquivokation:

$$\frac{\partial H(S|Y)}{\partial \lambda_{VQ}} = \sum_{k=1}^{K}\sum_{i=1}^{N} \frac{\partial f_i(\vec{x}(k))}{\partial \lambda_{VQ}} \cdot h_i(\vec{x}(k)). \tag{4.22}$$

Glg. (4.22) stellt nun eine allgemeingültige Berechnungsvorschrift für die Adjustierung des VQ-Parameters λ_{VQ} dar, die unabhängig von der Struktur des neuronalen Netzes verwendet werden kann. Unter Einbeziehung des Lernparameters β, mit welchem sich die Stabilität und die Konvergenz des Trainings kontrollieren lässt, kann mit Hilfe von Glg. 4.22 das Gewicht λ_{VQ} um $\Delta\lambda_{VQ}$ korrigiert werden:

$$\Delta\lambda_{VQ} = -\beta\frac{\partial H(S|Y)}{\partial\lambda_{VQ}} \tag{4.23}$$

Die bedingten Wahrscheinlichkeiten $p(s(k)|y_n)$ sind wiederum aus der gegebenen Trainingsdatenbasis zu schätzen.
Unter Verwendung eines nN-Vektorquantisierers ist für die Aktivierung f_i der Euklid'sche Abstand des J-dimensionalen Merkmalsvektors $\vec{x}$ zum nächsten VQ-Prototypenvektor $\vec{\lambda}_i$ einzusetzen:

$$f_i(\vec{x}) = \|\vec{x} - \vec{\lambda}_i\|_2^2 = \sum_{j=1}^{J}(x_j - \lambda_{VQ_{i,j}})^2. \tag{4.24}$$

4.3.2 Simultanes MMI-Training multipler Codebücher

Die Quantisierung eines reellwertigen, vieldimensionalen Vektors, auf einen skalaren, ganzzahligen Wert ist in der Regel ein mehr oder minder stark verlustbehafteter Prozeß. Diese Aussage gilt natürlich ebenso für MMI-trainierte VQ. Auch hier ist der Informationsverlust nicht vollständig aufzufangen. Neben dem MMI-Training besteht eine weitere einfache Möglichkeit den quantisierungsbedingten Verlust zu minimieren, indem das Codebuch vergrößert wird. Diese Möglichkeit der Verlustminimierung ist allerdings nur bis zu einem gewissen Grade möglich, da ab einer bestimmten Codebuchgröße die Parameter nicht mehr zuverlässig geschätzt werden können. Dieses *sparse-data*-Problem betrifft zum einen die VQ-Parameter, insbesondere aber die diskreten Wahrscheinlichkeitsdichtefunktionen (WDF) der HMM-Zustände, da diese WDF bei einer Codebuchgröße von J allgemein auch J zu schätzende Parameter umfassen. Bei einer vorgegebenen finiten Trainingsmenge muß also stets die Trainierbarkeit der Paramater gewährleistet bleiben. Ein günstiger Kompromiß zwischen Auflösungsvermögen des VQ und Trainierbarkeit sämtlicher Parameter kann nur auf experimentelle Weise gefunden werden.
Wird die Tatsache berücksichtigt, dass verschiedene Teile eines Merkmalsvektors in vielen Fällen aus verschiedenen Merkmalsextraktionsverfahren gewonnen werden, ergibt sich eine weitere Option zur Verbesserung der Auflösung der Vektorquatisierung. Diese wird durch eine Aufteilung des Merkmalsvektors $\vec{x}$ in Z verschiedene Untervektoren $\vec{x}^{(z)}$ erreicht, wobei jeder Untervektor $\vec{x}^{(z)}$ von einem bestimmten Vektorquantisierer $VQ^{(z)}$ mit moderater Codebuchgröße $J^{(z)}$ komprimiert wird. Damit läßt sich zum einen die Parameteranzahl der einzelnen WDF auf ein sinnvolles Maß begrenzen ($\sum_{z=1}^{Z} J^{(z)}$), zum anderen aber noch immer

eine hinreichend gute Auflösung mit theoretisch bis zu $\prod_{z=1}^{Z} J^{(z)}$ verschiedenen Partitionskombinationen erzielen. Mit den aus Kapitel 3 gewonnenen Erkenntnissen bietet sich eine Aufteilung in Trajektorien- und Bitmap-Merkmalen an. Eine Hinzunahme weiterer Merkmalsströme scheint aus derzeitiger Sicht wenig gewinnbringend, womit die folgenden Betrachtungen für $Z = 2$ gelten. Eine Verallgemeinerung auf weitere Merkmalsströme ist in analoger Weise möglich.
Idealerweise sollte die Multi-Stream-Technik so angewendet werden, dass die Aufteilung zu einer statistischen Unabhängigkeit der Teilvektoren $\vec{x}^{(z)}$ untereinander führt. Eine solche Annahme jedoch global zu treffen ist sicherlich nicht sehr realistisch. Bei einer sinnvollen Aufteilung in Untervektoren kann hingegen als Näherung die Annahme der lokalen statistischen Unabhängigkeit getroffen werden, wie sie auch bei der HMM-Parameterschätzung für Multi-Stream-Modelle genutzt wird. D. h., dass zumindest die Ausschnitte eines Streams als statistisch unabhängig angesehen werden, die einem bestimmten Zustand s zugeordnet werden. Die bedingte Auftrittswahrscheinlichkeit wird dann als Produkt der bedingten Einzelwahrscheinlichkeiten berechnet:

$$p(y(\vec{x})|s) = p(y^{(1)}(\vec{x}^{(1)}), \cdots, y^{(Z)}(\vec{x}^{(Z)})|s) = p(y^{(1)}(\vec{x}^{(1)})|s) \cdot p(y^{(2)}(\vec{x}^{(2)})|s). \tag{4.25}$$

Im Idealfall - bei globaler statistischer Unabhängigkeit - können somit die beiden VQ einzeln, d. h. unabhängig voneinander trainiert werden. Die Gesamt-Transinformation I_{ges} ist dann die Summe der Transinformationswerte der Einzel-VQ:

$$I_{ges} = I(Y^{(1)}, S) + I(Y^{(2)}, S). \tag{4.26}$$

Diese globale statistische Unabhängigkeit ist jedoch in starkem Maße idealisiert: in der Praxis sind stets Korrelationen zwischen verschiedenen Merkmalen festzustellen. Dies ist selbst bei Trajektorien- und Bitmap-Merkmalen der Fall. Um optimale Ergebnisse zu erzielen, sollte die Optimierung des einen VQ unter Berücksichtigung der übrigen VQ erfolgen. Dazu kann zunächst ausgehend vom Kanalmodell und Glg. (4.12), übertragen auf die Multi-Stream-Technik, die Gesamt-Transinformation I_{ges} als Gütemaß wie folgt formuliert werden:

$$I_{ges} = I(Y^{(1)}, Y^{(2)}, S) = H(Y^{(1)}, Y^{(2)}) - H(Y^{(1)}, Y^{(2)}|S). \tag{4.27}$$

Basierend auf der Annahme einer lokalen statistischen Unabhängigkeit (s. Glg. (4.25)), kann die Berechnung des zweiten Terms aus Glg. (4.27) auf die Summation der bedingten Einzelentropien zurückgeführt werden:

$$I_{ges} = H(Y^{(1)}, Y^{(2)}) - (H(Y^{(1)}|S) + H(Y^{(2)}|S)). \tag{4.28}$$

Erweitert man den Ausdruck um $\mp \sum_{z=1}^{2} H(Y^{(z)})$, ergibt sich

$$\begin{aligned} I_{ges} = \quad & -\left(\underbrace{\left\{ (H(Y^{(1)}) + H(Y^{(2)}) \right\} - H(Y^{(1)}, Y^{(2)})}_{I(Y^{(1)}, Y^{(2)})} \right) \\ & + \underbrace{H(Y^{(1)}) - H(Y^{(1)}|S)}_{I(Y^{(1)}, S)} + \underbrace{H(Y^{(2)}) - H(Y^{(2)}|S)}_{I(Y^{(2)}, S)}, \end{aligned} \tag{4.29}$$

was schließlich zu der verwendeten Bewertungsfunktion I_{ges} führt:

$$I_{ges} = -I(Y^{(1)}, Y^{(2)}) + I(Y^{(1)}, S) + I(Y^{(2)}, S) \tag{4.30}$$

Verglichen mit der idealisierten Funktion für I_{ges} aus Glg. (4.26), wird die Transinformationssumme der einzelnen Ströme um den Summanden $-I(Y^{(1)}, Y^{(2)})$ korrigiert. Dieser Korrekturterm beschreibt nun genau die Korrelation der einzelnen diskreten Merkmalsströme untereinander und kann daher auch als Maß für die Redundanz der Merkmalströme interpretiert werden. Für die Praxis bedeutet dies, dass die Aufteilung der Untervektoren und deren statistische Unabhängigkeit nicht mehr als kritisch für die spätere Erkennungsleistung gelten muß. Dies wird dadurch erreicht, dass neben der Maximierung der Transinformation einzelner Merkmalsströme die Dekorrelation der Merkmale als zusätzliches Bewertungskriterium in den Trainingsprozeß einfließt.

Die Maximierung von $I(Y^{(z)}, S)$ der einzelnen VQ erfolgt bei simultanem Training analog zu dem separaten MMI-Training aus Glg. (4.21) und Glg. (4.22). Entsprechend Glg. (4.30) ist zur Maximierung des Gesamtausdrucks noch die Minimierung der statistischen VQ-Abhängigkeiten nötig. Mit folgender Gleichung kann die Transinformation der statistischen VQ-Abhängigkeiten ausgedrückt werden:

$$I(Y^{(1)}, Y^{(2)}) = \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} p(y_{j1}, y_{j2}) \cdot \log \frac{\bar{p}(y_{j1}, y_{j2})}{p(y_{j1}) \cdot p(y_{j2})} \tag{4.31}$$

Entsprechend der Jensen-Ungleichung [ST95] läßt sich die Transinformation der VQ-Abhängigkeiten $I(Y^{(1)}, Y^{(2)})$ aus den WDF der Modelle $\bar{p}$ und den wahren WDF p berechnen, welche approximativ aus den Trainingsdaten ermittelt werden. Ähnlich wie im vorangegangenen Abschnitt kann nun unter Anwendung der Kettenregel die partielle Ableitung von $I(Y^{(1)}, Y^{(2)})$ nach einem bestimmten VQ-Gewicht λ_{VQ} des z-ten VQ bestimmt werden:

$$\frac{\partial I(Y^{(1)}, Y^{(2)})}{\partial \lambda_{VQ}} = \sum_{k=1}^{K} \sum_{i=1}^{J_z} \frac{\partial f_i(\vec{x}(k))}{\partial \lambda_{VQ}} \cdot h_i(\vec{x}(k)). \tag{4.32}$$

Mit der Aktivierungsfunktion f_i entsprechend Glg. (4.24) und der Hilfsfunktion h_i, mit

$$\begin{aligned} h_i^{(z)}(\vec{x}(k)) = \quad & \tfrac{C}{T} \cdot o_n^{(z)}(\vec{x}(k)) \cdot \\ & \cdot \sum_{j=1}^{J_z} \left(\tfrac{\partial I(Y^{(1)}, Y^{(2)})}{\partial p(y_1(k), \cdots, y_j^{(z)}, \cdots, y^{(Z)}(k))} + \tfrac{\partial I(Y^{(1)}, Y^{(2)})}{\partial p(s(k), y_j^{(z)})} \right) \cdot (\delta_{i,j} - o_j^{(z)}(\vec{x}(k))) \end{aligned} \tag{4.33}$$

läßt sich die Berechnungsvorschrift entsprechend Glg. (4.32) für die Einstellung der Gewichte angeben.

4.4 Ergebnisse

Die Ergebnisse des Trainings eines Multi-Stream-Systems mit zwei Merkmalsströmen $Y^{(1)}$ und $Y^{(2)}$ sind mit dem entsprechenden Transinformationsverlauf in Abb. 4.5 dargestellt. In der Regel wird ein solches Training in drei Stufen durchgeführt. Die erste Stufe dient der Initialisierung des Netzes. Diese Initialisierung wird hier bei Verwendung Euklid'scher Abstandsmaße als Aktivierungsfunktionen mit einem k-means Algorithmus durchgeführt.
In der zweiten Stufe setzt dann auf diesem k-means-VQ das MMI-Training für einzelne Codebücher auf, wie es in Abschnitt 4.3.1 beschrieben wurde. In Abb. 4.5 entspricht dies den Iterationen 1 bis 10. In dieser zweiten Trainingsstufe (separates Training) ist der Anstieg der Transinformation deutlich für jedes der eingesetzten Codebücher zu sehen.
Anschließend wird in der dritten Stufe ein simultanes Training der beiden Codebücher durchgeführt (Iterationen 11-20). In dieser Stufe schließlich, wird als Bewertungsfunktion die Gesamt-Transinformation $I_{ges} = I(Y^{(1)}, Y^{(2)}, S)$ verwendet. Bei diesem simultanen Training wird zwecks Maximierung von I_{ges} eine leichte Steigerung von $I(Y^{(1)}, S)$ erreicht. Die Transinformation des zweiten Codebuches $I(Y^{(2)}, S)$ wird hingegen sogar verringert. Dies geschieht zu Gunsten der Dekorrelation der einzelnen Merkmale. Die Korrelation ist abzulesen an dem Verlauf der Transinformation der VQ-Abhängigkeiten $I(Y^{(1)}, Y^{(2)})$. Insgesamt ist jedoch ein deutlicher Anstieg der Gesamt-Transinformation I_{ges} zu erkennen.

Prinzipiell könnte die zweite Stufe dieses Verfahrens (separates Training) durch eine verlängerte dritte Trainingsstufe (simultanes Training) ersetzt werden, was durchaus zu ähnlichen Ergebnissen führt. Da die dritte Stufe jedoch relativ rechenzeitaufwendig ist stellt sich die geschilderte Variante mit k-means, separatem und simultanem Training als sehr praktikabel dar.

Tab. 4.1 zeigt zunächst die Ergebnisse eines schreiberabhängigen Systems mit einem Vokabular bestehend aus 30000 Wörtern. Die hier gezeigten Ergebnisse wurden mit einem diskreten System erzielt. Die einzelnen Zeilen zeigen die Ergebnisse jedes Schreibers ('ank', 'jmr' und 'vdm') mit den entsprechenden Durchschnittsergebnissen für verschiedene Merkmals-

	ank	jmr	vdm	ø
BM:	95.2 %	84.5 %	95.2 %	91.6 %
$[\alpha, \tilde{p}]$:	90.3 %	85.5 %	70.7 %	82.2 %
alle:	93.0 %	77.9 %	83.4 %	84.8 %

Tabelle 4.1: Erkennungsergebnis - diskretes System, schreiberabhängig, 30K Lexikon

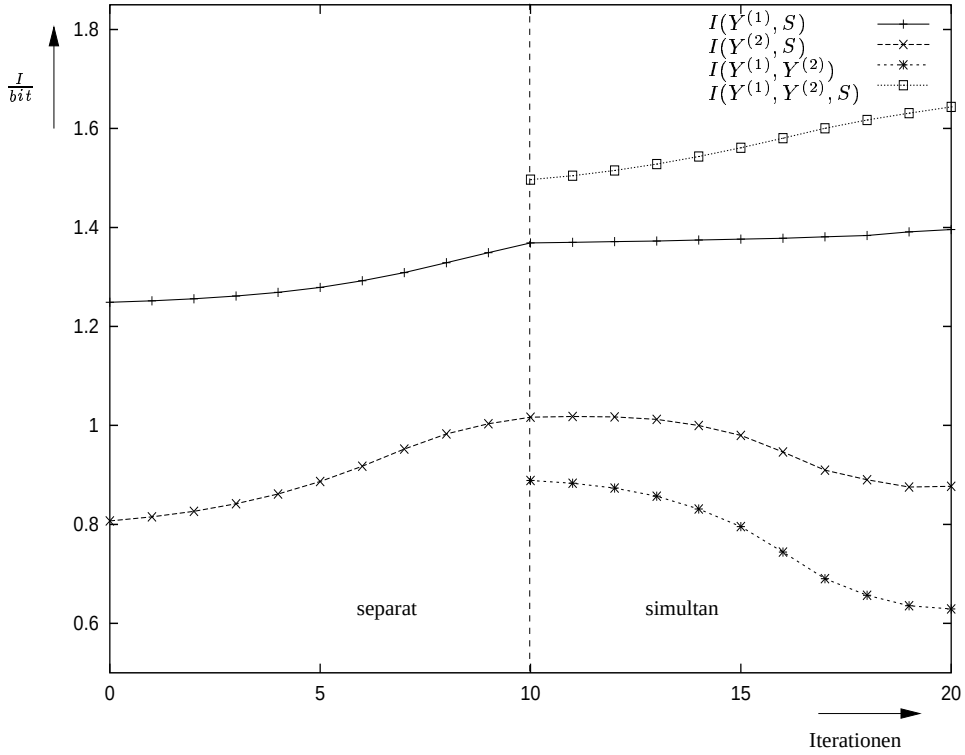


Abbildung 4.5: Verlauf der Transinformation für separates und simultanes MMI-Training

kombinationen. Die Kettencodierung (Zeile 2) weist hierbei deutlich schlechtere Resultate auf, als die gleitende Bitmap (Zeile 1). Erstaunlich in dem schreiberabhängigen Fall ist, dass die Merkmalskombination auch eher kontraproduktiv wirkt. Bei den folgenden Diskussionen soll daher der Fokus auf der Bitmap und auf den kombinierten Merkmalen liegen.

Tab. 4.2 zeigt die Ergebnisse der untersuchten Paradigmen. Zeile 1 und 3 zeigen die Ergebnisse für die Bitmap als Einzelmerkmal. Die übrigen Zeilen stellen die Ergebnisse kombinierter Merkmale dar. Während mit dem diskreten System unter Verwendung der Bitmap als Einzelmerkmal noch 91,6 % Korrektheit erzielt wurden (Tab. 4.1), konnten mit kontinuierlichen Modellen bereits 92,0 % erzielt werden. Das hybride System hingegen lieferte bei gleichen Bedingungen durchschnittlich 95,2 %. Die Merkmalskombination wirkt sich bei kontinuierlichen Systemen mit 94,6 %, wie hybriden Systemen mit 94,3 % wiederum uneinheitlich aus. Bei dem kontinuierlichen System liegt der Vorteil darin begründet, dass sämtliche Merkmale in einem Merkmalsstrom zusammengefasst werden, und so statistische Unabhängigkeitsannahmen weniger kritisch sind. Dieser Nachteil kann schließlich mit dem simultanen MMI Training von Bitmap und α-Merkmal aufgeholt werden. Es ergibt sich schließlich eine Erkennungsrate von 96,8 %, was einer relativen Fehlerreduktion verglichen mit dem diskreten Basissystem (Tab. 4.1, Zeile 3) um 79 % entspricht. Diese Experimente werden anschließend unter Verwendung eines Lexikons mit 200000 Wörtern wiederholt (s. Tab. 4.3). Die Ergebnisse des kontinuierlichen Systems und des hybriden Systems liegen zunächst wieder relativ dicht beieinander. Es zeigt sich jedoch auch hier eine Überlegenheit

des simultanen MMI-Trainings mit 91,3 % Korrektheit gegenüber der kontinuierlichen Modellierung oder dem separaten MMI-Training. Im Vergleich zu dem kontinuierlichen System mit 87,7 % bedeutet dies eine relative Fehlerreduktion um immerhin 29 %. Die Ergebnisse des schreiberabhängigen Systems sind bereits optimiert bezüglich Codebuchgröße bzw. Anzahl der Mischverteilungen und bezüglich der Kontextgröße.
Diese Erfahrungswerte werden nun wiederum eingesetzt um das schreiberunabhängige System zu evaluieren. Mit dieser näherungsweise optimalen Parameteranzahl wird für das diskrete WI-System bei Verwendung eines 2000 Wörter umfassenden Lexikon unter Nutzung kombinierter Merkmale eine Erkennungsrate von 86,9 % erreicht. Wird das α-Codebuch konstant gehalten und nur das Bitmap-Codebuch MMI-trainiert, verbessert sich das Ergebnis auf 88,9 %. Bei einem separaten MMI-Training beider Codebücher läßt sich dieser Wert auf 89,7 % steigern. Eine letztmalige Steigerung ist mit dem simultanen MMI-Training beider Codebücher (dritte Stufe) möglich. Hierbei wird schließlich eine Erkennungsrate von 90,6 % möglich.

4.5 Kapitelzusammenfassung

Wie in den vorangegangenen Abschnitten gezeigt wurde, minimiert ein MMI- (*maximum mutual information*-) Training des Vektorquantisierers die Äquivokation, die ihrerseits im wesentlichen von den Informationsverlusten der Vektorquantisierung herrührt. Gleichzeitig wird das Gesamtsystem (VQ und HMMs) aber geschlossen nach dem ML-Verfahren trainiert, wobei in das Training des Vektorquantisierers zusätzlich die Klasseninformation der einzelnen Trainingsvektoren einfließt. Das MMI-Training ist sowohl für verschiedene neuronale Paradigmen mit verschiedenen Aktivierungsfunktionen anwendbar, wie auch für das simultane Training mehrerer VQ in einem Multi-Stream-System, bei dem verschiedene Merkmalstypen über verschiedene Codebücher in unabhängigen Merkmalströmen modelliert werden.
Abschließend läßt sich zusammenfassen, dass mit dem hier beschriebenen Verfahren die entscheidenden Nachteile eines diskreten Systems verglichen mit einem kontinuierlichen Sys-

	ank	jmr	vdm	ø	
BM:	95.7 %	88.2 %	92.2 %	92.0 %	kontinuierlich
alle:	96.8 %	91.4 %	95.7 %	94.6 %	
BM:	96.8 %	87.7 %	95.7 %	95.2 %	MMI (separat)
alle:	96.2 %	89.8 %	96.8 %	94.3 %	
alle:	97.3 %	94.1 %	98.9 %	96.8 %	MMI (simultan)

Tabelle 4.2: Erkennungsergebnis - schreiberabhängig, 30K Lexikon

	ank	jmr	vdm	ø	
alle:	94.1 %	77.5 %	91.4 %	87.7 %	kontin.
BM:	91.4 %	79.3 %	94.2 %	88.3 %	MMI (separat)
alle:	97.3 %	83.4 %	93.1 %	91.3 %	MMI (simultan)

Tabelle 4.3: Erkennungsergebnis - schreiberabhängig, 200K Lexikon

tem (s. Abschnitt 4.1) eliminiert werden konnten. Neben den sehr guten Erkennungsraten ergeben sich mit der Verwendung hybrider Modelle einige weitere wesentliche Vorteile:

- Es bietet sich mit dem hybriden Ansatz die Möglichkeit zur Realisierung verteilter Systeme. Bei geringer Bitrate können die quantisierten Merkmale ohne Verluste der Erkennungsgenauigkeit zum Erkenner übertragen werden.
- Die für das MMI-Training zu definierenden Klassen können Zustände, Grapheme (Bi- und Trigrapheme) oder - im Fall von Erkennungssystemen mit kleinem Vokabular - Worte sein.
- Die Unabhängigkeit von der Klassendefinition führt dazu, dass die Anzahl der Ausgangsknoten des VQ unabhängig von der Klassenanzahl (Anzahl der Zustände, Grapheme oder Worte) ist. Die Parameteranzahl der Codebücher und HMM-Verteilungen läßt sich also unabhängig von den gewählten Klassen optimieren.
- Es bedarf keiner Annahme über die Art der Verteilung der verwendeten Merkmale. Die Merkmale können so auch nicht Gauß-verteilt oder sogar diskreter Natur sein.
- Die Berechnung der Wahrscheinlichkeiten aus den diskreten HMM-Verteilungen läßt sich durch *table look-up* effizient gestalten.

Die überlegene Erkennungsleistung mit den weiteren genannten Vorteilen legt daher die Anwendung hybrider Modelle, d. h. die Kombination MMI-trainierter neuronaler Netze für die Vektorquantisierung mit HMM nahe.

Kapitel 5

Strukturoptimierung bei Verwendung kontextabhängiger Modelle

Bezüglich der Modellstruktur sind die verschiedensten Ansätze zur Systemoptimierung denkbar. Dies kann sowohl die Variation der Anzahl der Zustände betreffen, wie auch die Modelltopologie als solche, die durch die Zustandsübergänge vorgegeben wird oder anhand von Trainingsbeispielen gelernt werden kann. Der Schwerpunkt der Betrachtungen liegt in diesem Kapitel jedoch auf der Einführung multipler Modelle. Wie später noch zu verdeutlichen ist, wird mit der Verwendung multipler Modelle eine Strukturoptimierung in der Form erforderlich, dass gewisse Parameter verschiedener Modelle optimal miteinander verknüpft werden müssen.

Bisher wurde stets davon ausgegangen, dass für jedes lexikalische Element *ein* Modell zur Verfügung steht. Als Alternative ist es jedoch auch denkbar, für jedes zu erkennende Zeichen mehrere Modelle einzuführen [Kos99d, Kos97c]. Damit stellt sich zunächst die grundsätzliche Frage, nach welchem Kriterium dies zu erfolgen hat. Betrachtet man dazu das in Abb. 5.1 gezeigte Beispiel und dort insbesondere die markierten mehrfach auftretenden Buchstaben, so wird deutlich, dass selbst bei einem festen Schreiber Realisierungen des selben Buchstabens doch deutlich voneinander abweichen können. Offensichtlich handelt es sich hier um den Einfluß der benachbarten Zeichen. Dieser Effekt kann dadurch erklärt werden, dass der Schreiber versucht, während des Schreibprozesses eine möglichst effiziente Stiftführung zu erzielen und dabei gewisse Inkonsistenzen innerhalb der selben Zeichenklasse auftreten. Das erste auftretende ’e’ im Kontext von ’t’ und ’x’ z. B. ist im Vergleich mit dem zweiten ’e’ (im Kontext von ’k’ und ’i’) deutlich ausgeprägter. Auch liegt der Anfangspunkt des ersten ’e’ höher als bei der zweiten Realisierung. Insgesamt läßt sich bei der ersten Variante eine deutlichere Separation von den benachbarten Zeichen feststellen. In gleicher Form ließen sich offensichtliche Differenzen bei den übrigen mehrfach vorkommenden Zeichen aufzählen.

Dem könnte entgegenhalten werden, dass möglicherweise mehr Gemeinsamkeiten inner-

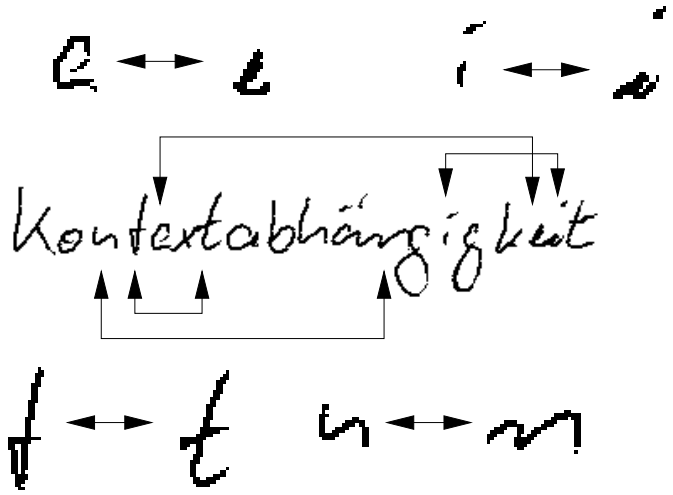


Abbildung 5.1: Kontextbedingte Graphemvariationen

halb einer Zeichenklasse existieren als Unterschiede. Ein Beispiel dazu wären sicherlich die in Abb. 5.1 gezeigten 'g', sowie das zweite und dritte 't', welche jeweils über sehr ähnliche Charakteristika verfügen. Dabei muß man sich allerdings verdeutlichen, in welcher Form die Schrift dem Erkennungssystem präsentiert wird. Die extrahierten Merkmale - insbesondere die gleitende Bitmap - erfassen neben dem lokalen Ausschnitt der Trajektorie auch einen bestimmten Umgebungsbereich (vergl. Abb. 3.7). Zeitlich betrachtet geschieht dies zudem global. Am Beispiel der gezeigten 'g' bedeutet dies, dass selbst bei scheinbar großer Ähnlichkeit erhebliche Unterschiede in der Merkmalssequenz vorliegen. Bei der ersten Realisierung wird durch die gleitende Bitmap ein Teil des vorherigen 'n' erfasst und als Merkmal mitgeführt. Sowohl die kontextbedingten Einflüsse auf die Form der Trajektorie wie auch die kontexterfassende Mermalssequenz der Bitmap sollten sich natürlich um so deutlicher auswirken, je stärker die Buchstaben untereinander verbunden sind (kursive Schreibweise).
Aufgrund dieser Betrachtungen scheint es also naheliegend zusätzliche Modelle für ein Zeichen - in Abhängigkeit des jeweiligen Kontextes - einzuführen. In Anlehnung an die in der Spracherkennung übliche Terminologie für kontextabhängige Modelle wird diese Form von Modellen nachfolgend als Allographeme, bzw. als Bi- und Trigrapheme bezeichnet.

5.1 Bi- und Trigrapheme

Zentrales Problem bei der Einführung von Allographemen ist die große Parameteranzahl der Menge aller Modelle. Bei der Verwendung von N-Graphemen wird ein Kontext von N-1 benachbarten Zeichen berücksichtigt. Dies bedeutet, dass bei einem Grundzeichensatz bestehend aus 80 Modellen letztendlich 80^N Zeichenkombinationen vorhanden sind. Weiterhin muß davon ausgegangen werden, dass die Trainingsdatenbasis und damit die Anzahl

der Trainingsbeispiele nicht beliebig erhöht werden kann oder gar fest vorgegeben ist. Diese beiden Randbedingungen (stark steigende Parameteranzahl und feste Trainingsdatenbasis) erfordern demnach einen sinnvollen Kompromiß zwischen einer möglichst hohen Genauigkeit der Modelle und deren hinreichender Trainierbarkeit [Lee90]. Die Trainierbarkeit ist gegeben, wenn jeder Modellparameter anhand ausreichenden Trainingsmaterials geschätzt werden kann.

Der Kompromiß zwischen Trainierbarkeit und Genauigkeit läßt sich zuerst durch einige naheliegende Maßnahmen anstreben. Bei einer betrachteten Kontexttiefe von N sollte man sich zunächst vor Augen führen, dass viele der 80^N theoretisch möglichen Zeichenkombinationen äußerst ungebräuchlich sind. Darum ist es sinnvoll die Expansion von Monographemen zu Allographemen auf der Basis eines festen Lexikons, bestehend aus gültigen und gebräuchlichen Wörtern, vorzunehmen. Bei der Verwendung sehr großer Lexika in Kombination mit einer hohen berücksichtigten Kontexttiefe kann es dennoch im Verhältnis zur Trainingsdatenbasis zu einer unangemessenen Überhöhung der Modell- und Parameteranzahl kommen. Da die Modellanzahl also bei wachsender Kontexttiefe überproportional ansteigt, gleichzeitig aber der Einfluß weiter entfernter Zeichen abnimmt, ergibt sich als guter Kompromiß mit handhabbarer Modellanzahl eine maximale Kontexttiefe von $N = 3$.

Tabelle 5.1 zeigt neben der Graphemfolge die Entwicklung der Modellanzahl bei variierender Kontexttiefe von $N = 1, 2, 3$. Die Notation bei der Graphemisierung der Lexikoneinträge gibt Auskunft über das zugrundeliegende Modell und den berücksichtigten Kontext.

		Monographeme	Bigrapheme (li.)
Graphemisierung		/K/ /o/ /n/ **/t/** /e/ /x/ **/t/**	/K/ K/o/ o/n/ **n/t/** t/e/ e/x/ **x/t/**
Anzahl Modelle	1k-Lexikon	80	800
	30k-Lexikon	80	1300
	200k-Lexikon	80	2800
	kein Lexikon	80	6400
		Bigrapheme (re.)	Trigrapheme
Graphemisierung		/K/o /o/n /n/t **/t/e** /e/x /x/t **/t/**	/K/o K/o/n o/n/t **n/t/e** t/e/x e/x/t **x/t/**
Anzahl Modelle	1k-Lexikon	800	3000
	30k-Lexikon	1300	8900
	200k-Lexikon	2800	25000
	kein Lexikon	6400	512000

Tabelle 5.1: Modellexpansion bei Verwendung von Allographemen am Beispiel des Wortes 'Kontext'

’/t/’ steht in diesem Zusammenhang für das kontextunabhängige HMM (Monographem) des Buchstabens ’t’. Für den Fall der links-kontextabhängigen Bigrapheme würden für den Buchstaben ’t’ in dem Wort ’Kontext’ zwei HMM eingeführt werden: n/t/ mit dem linken Kontextbuchstaben ’n’ und x/t/ mit dem linken Nachbarn ’x’. Entsprechend würden bei der Verwendung rechts-kontextabhängiger Modelle die HMM /t/e und /t/ gebildet werden. Für Trigrapheme schließlich ergäben sich in analoger Weise die Modelle n/t/e und x/t/ (jeweils in Fettdruck markiert).

5.2 Parameterreduktion

Auch unter der Voraussetzung, dass bei der Erkennung ein festes Lexikon eingesetzt wird, und damit die Kombinationsvielfalt kontextueller Modelle eingeschränkt wird, zeigen die vorangehenden Betrachtungen, dass die hinreichende Trainierbarkeit kontextabhängiger Modelle damit allein nicht unbedingt gegeben sein muß. Im Falle der Trigrapheme werden auf der Grundlage eines 200000 Wörter umfassenden Lexikons statt der ursprünglichen 80 HMM nun 25000(!) HMM generiert.
Jedes dieser HMM besteht weiterhin aus 12 Zuständen. Pro Zustand wiederum setzen sich die diskreten Verteilungen aus insgesamt 142 Parametern zusammen. Insgesamt besteht ein Trigraphem-System also aus $12 \cdot 142 = 1704$ Verteilungsparametern zuzüglich der 24 (pro Zustand ein Selbstübergang und ein Übergang zum Nachfolgezustand) von Null verschiedenen Transitionswahrscheinlichkeiten. Dies bedeutet in der Summe, dass für den Aufbau eines Trigraphem-Systems 42600000 Parameter zu schätzen sind. Dem stehen in der WD-Datenbasis (Abschnitt B.1) ca. 500000 Merkmalsvektoren zur Verfügung. Diese deutliche Diskrepanz zeigt klar die Notwendigkeit die Anzahl der Modellparameter weiter zu reduzieren.
Dazu sind im Prinzip zwei verschiedene Ansätze denkbar, die schließlich auf der Basis von Trigraphemen verglichen werden. Neben dem selektiven Ansatz [Kos97b, Kos97a], der in Abschnitt 5.2.1 beschrieben wird, sind darüber hinaus noch Verfahren zur Parameterverknüpfung (Tying) denkbar [Lee90]. Das Grundprinzip der Parameterverknüpfung ist in Abschnitt 5.2.2 erklärt. Spezielle Clustering-Algorithmen dazu werden in den Abschnitten 5.3 und 5.4 beschrieben. Da das *sparse-data* Problem insbesondere bei Trigraphemen auftritt, diese aber den Vorteil haben, beidseitig den Kontext zu berücksichtigen, werden die verschiedenen Verfahren schließlich anhand von Trigraphemen evaluiert.

5.2.1 Selektives Verfahren

Um die höhere Konsistenz von Trigraphemen auszunutzen, muß also die Parameteranzahl zum Teil deutlich reduziert werden. Dazu ist es zunächst sinnvoll die Häufigkeiten der Trigrapheme zu betrachten. Abb. 5.2 zeigt die Verteilung der Trigraphem-Häufigkeiten in der Trainingsdatenbasis auf der Grundlage des 1K-Lexikons. N bezeichnet dabei die Anzahl der Trainingsbeispiele für ein HMM. $M(N)$ ist die Anzahl der Trigrapheme, welche durch genau N Trainingsbeispiele in der Trainingsdatenbasis repräsentiert sind. Erstaunlich ist hier, dass - selbst bei einem mittelgroßen Vokabular von 1000 Wörtern - die meisten erzeugten Trigrapheme absolut unterrepräsentiert sind. So existieren 310 Trigrapheme, die lediglich anhand von zwei Beispielen ($N = 2$) in der Trainingsmenge trainiert werden könnten. Für $N = 1$ würden sogar 860 HMM existieren und für $N = 0$ nochmals 300 Modelle. Diese 300 'ungesehenen' Trigrapheme resultieren daher, dass das Lexikon über Worteinträge verfügt, die ausschließlich für den Test benötigt werden, wofür es aber keine Trainingsbeispiele gibt. Zusammen würden also bereits rund die Hälfte (1470 von insgesamt 3000) aller erzeugten Trigrapheme mit keinem, einem oder nur anhand von zwei Beispielen trainiert werden.
Anhand dieser Verteilungen läßt sich nun eine minimal erforderliche Häufigkeit definieren, anhand derer die Generierung von Trigraphemen gesteuert werden kann. Dazu wird zunächst eine Expansion des Monographem-Lexikons auf Trigrapheme durchgeführt mit einer anschließenden Berechnung der Häufigkeitsverteilung. Die Kontextkombinationen deren Häufigkeit der Trainingsbeispiele N einen Schwellwert N_{min} überschreitet werden schließlich als Trigrapheme eingeführt.
Mit einer hinreichend großen Wahl von N_{min} kann somit die Trainierbarkeit jedes einzelnen Modells garantiert werden. Ein Nachteil dieses Verfahrens ist, dass die erzeugten Trigraph-

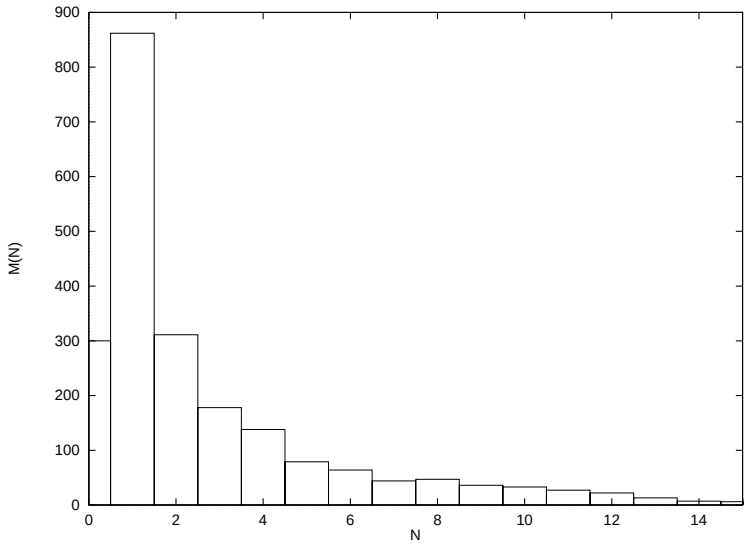


Abbildung 5.2: Trigraphem-Häufigkeiten der Trainingsdatenbasis (1K-Lexikon)

eme keine Generalisierungsfähigkeiten aufweisen, da sie für eine ganz bestimmte Kontextkombination vorgesehen sind. Es kann also der Fall eintreten, dass die Erzeugung zweier Trigrapheme, z. B. n/a/c und m/a/c, aufgrund unzureichenden Trainingsmaterials zurückgewiesen wird, obwohl deren Kontextkombination zu sehr ähnlichen Realisierungen führt. Liegt eine gewisse Ähnlichkeit vor, würde dies wiederum bedeuten, dass beide Trigrapheme mit Realisierungen aus beiden Kontextkombinationen trainiert werden könnten. Die Parameter vieler gering okkupierter, jedoch ähnlicher Grapheme könnten gemeinsam und damit robuster geschätzt werden, als dies bei separaten Trainingsbeispielen der Fall wäre.

5.2.2 Parameter-Tying

Das Training verschiedener HMM anhand identischer Trainingsbeispiele führt natürlich zu absolut identischen Modellparametern. Als Konsequenz dazu können diese Trigrapheme ebenso zu einem einzigen generalisierten Trigraphem vereinigt werden. Dies führt bei dem genannten Beispiel zu einem Trigraphem $\{$m,n$\}$/a/c, bei dem der rechte Kontext des Buchstabens 'a' ein 'c' ist, und der linke Kontext entweder aus einem 'm' oder einem 'n' bestehen kann. Führt man diese Idee weiter, gelangt man zu Strukturen, wie sie in Abb. 5.3 gezeigt sind. Die gestrichelt dargestellten Zustände bedeuten logische Zustände, die im Prinzip lediglich aus einem Zeiger auf physikalische Zustände bestehen. Solche Überkreuz-Verknüpfungen ermöglichen die gemeinsame Parameternutzung über verschiedene Trigrapheme des gleichen Basisgraphems, wobei die Anfangszustände des Modells d/e/s mit einem völlig anderen Modell verknüpft werden, als dessen Endzustände.
Verschiedene Allographeme des gleichen Stammgraphems benutzen damit einen Pool von Parametern (insbesondere Zustände) gleichzeitig und bilden so eine Form verallgemeinerter Allographeme. Diese verallgemeinerten Allographeme sollten dann so strukturiert werden, dass sie ggf. bei vorhandenem Trainingsmaterial spezielle Graphemcharakteristika abbilden können oder aber die zuverlässig schätzbaren Parameter ähnlicher Allographeme nutzen können. Konkret bedeutet dies, dass Allographeme mit gleichem Stammgraphem - im Beispiel in Abb. 5.3 das /e/ - über die selbe Transitionsmatrix verfügen. Diese Transitionsmatrix wird dem entsprechend auch anhand aller Trainingsbeispiele $*/e/* = /e/$ geschätzt.
Das Tying kann sowohl auf die Transitionsmatrizen angewendet werden, wie auch auf einzelne merkmalsspezifische Verteilungen verschiedener Zustände oder gar auf ganze HMM-Zustände. Geht man davon aus, dass insbesondere innerhalb einer Graphemklasse gewisse Korrelationen in den Merkmalen auftreten und diese sich wiederum in den verschiedenen Verteilungen widerspiegeln, ist es angemessen das Tying allgemein auf ganze HMM-Zustände zu beziehen. Vor allem im Zusammenhang mit großen und sehr großen Lexika und der damit einhergehenden großen Anzahl von Parametern erfordert das Tying selbstorganisierende Clustering-Verfahren, wie das datengetriebene State-Clustering oder das Entschei-

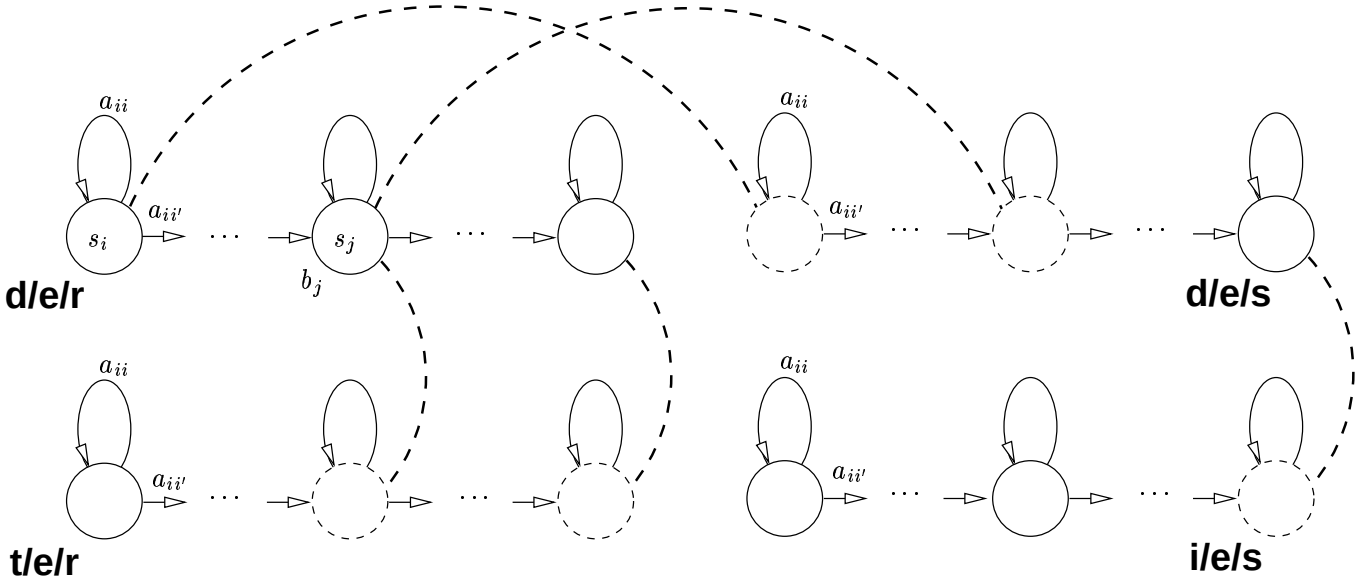


Abbildung 5.3: Parameterverknüpfung bei Trigraphemen

dungsbaum basierte Clustering.

5.3 Datengetriebenes State-Clustering

Die Grundidee des datengetriebenen Clusterings ist es, Zustände aufgrund eines gewissen Abstandsmaßes zusammenzufassen [You92, You93]. Da es sich hier um diskrete Wahrscheinlichkeitsdichtefunktionen handelt, kann die Ausgabeverteilung als Vektor der Dimension $N_C^{(z)}$ betrachtet werden, wenn das Codebuch des z-ten Merkmalsstromes die Größe N_C hat. Die einzelnen Vektorelemente entsprechen demnach den zustandsbedingten Wahrscheinlichkeiten einzelner VQ-Partitionen. Ein Vergleich verschiedener Verteilungen b_i und b_j des selben Merkmalsstromes z in den Zuständen i und j kann dann anhand des Euklid'schen Abstandes zwischen den Ausgabeverteilungen (bzw. Vektoren) b_i und b_j erfolgen:

$$d(i,j) = \sum_{n=1}^{N_C} (b_i(n) - b_j(n))^2. \tag{5.1}$$

Das Abstandsmaß der Ausgabeverteilungen kann dabei die direkte Auftrittswahrscheinlichkeit $P(n)$ der Partition n eines Vektorquantisierers berücksichtigen, wie auch deren logarithmierte Form $\log P(n)$. Für den gebräuchlicheren Fall der logarithmierten Wahrscheinlichkeiten ergibt sich somit

$$d(i,j) = \sum_{n=1}^{N_C} (\log\{P_i(n)\} - \log\{P_j(n)\})^2. \tag{5.2}$$

Weiterhin ist zu berücksichtigen, dass verschiedene Merkmalsströme unterschiedliche Codebuchgrößen aufweisen. Soll nun das Tying anstatt auf einzelne Verteilungen auf ganze

HMM-Zustände angewendet werden, ist die Codebuchgröße $N_C^{(z)}$ des Vektorquantisierers z zu berücksichtigen, sodass sich der Gesamtabstand zweier Zustände i und j mit Z unabhängigen und ggf. unterschiedlich dimensionierten Codebüchern wie folgt formulieren läßt:

$$d(i,j) = \frac{1}{Z}\sum_{z=1}^{Z}\sqrt{\frac{1}{N^{(z)}}\sum_{n=1}^{N_C^{(z)}}(\log\{P_i^{(z)}(n)^{\gamma_z}\} - \log\{P_j^{(z)}(n)^{\gamma_z}\})^2}. \tag{5.3}$$

Unter Einbeziehung der Merkmalsgewichtungen γ_z lassen sich für das Clustering zusätzlich noch Gewichtungen der einzelnen Merkmalsströme berücksichtigen. Als Standardeinstellung gilt jedoch $\gamma_z = 1$.

Zur Initialisierung wird zunächst jedes Cluster mit genau einem Zustand besetzt. Um festzustellen welche Cluster zusammengefasst werden, muß zuerst für jedes Cluster-Paar aus den Zustandsabständen nach Glg. 5.3 ein Clusterabstand ermittelt werden. Der Abstand zweier Cluster ist die maximale Distanz, die beliebige Zustände innerhalb der beiden Cluster untereinander aufweisen. D. h. dass die Distanz zweier Cluster i und j, welche die Zustände k und l beinhalten aus der Maximumsuche unter den Abständen aller beteiligter Zustandsabstände ermittelt werden kann. Der Maximalabstand

$$d(k^*, l^*) = \max_{k,l} d(k,l). \tag{5.4}$$

zwischen den Zuständen k^* und l^* führt somit zu der gesuchten Distanz d_C zwischen den Clustern i und j mit

$$d_C(i,j) = d(k^*, l^*). \tag{5.5}$$

Eine anschließende Minimumsuche

$$(i^*, j^*) = \arg\min_{i,j} d_C(i,j) \tag{5.6}$$

unter den Abständen d_C aller Cluster-Paare i, j führt schließlich auf die zu verschmelzenden Cluster i^* und j^*. Abb. 5.4 illustriert diese Vorgehensweise.

Würde dieser Algorithmus sukzessive in der Form weitergeführt werden, hätte dies zur Folge, dass am Ende wieder alle Zustände in einen Cluster kollabieren würden. Darum ist es zudem erforderlich, eine Abbruchbedingung einzubringen. Dies läßt sich z. B. in der Form realisieren, dass eine minimale Anzahl verbleibender Cluster vorgegeben wird. Da dies aber nicht unbedingt eine Vorhersage über die Clustergröße zuläßt, kann alternativ dazu eine maximale Clusterdistanz $d_{C,max}$ definiert werden, bei deren Erreichen der Clustering-Algorithmus abgebrochen wird. Weiterhin kann das Clustering in der Weise verfeinert werden, dass in einem letzten Durchlauf besonders gering frequentierte Cluster entsprechend des definierten Abstandsmaßes mit ihrem jeweils nächsten Nachbarn zusammengefasst werden. In Kombination sorgen beide Maßnahmen für eine ausbalancierte Clustergröße, wodurch ein guter Kompromiß zwischen Trainierbarkeit und Genauigkeit erreicht werden kann.

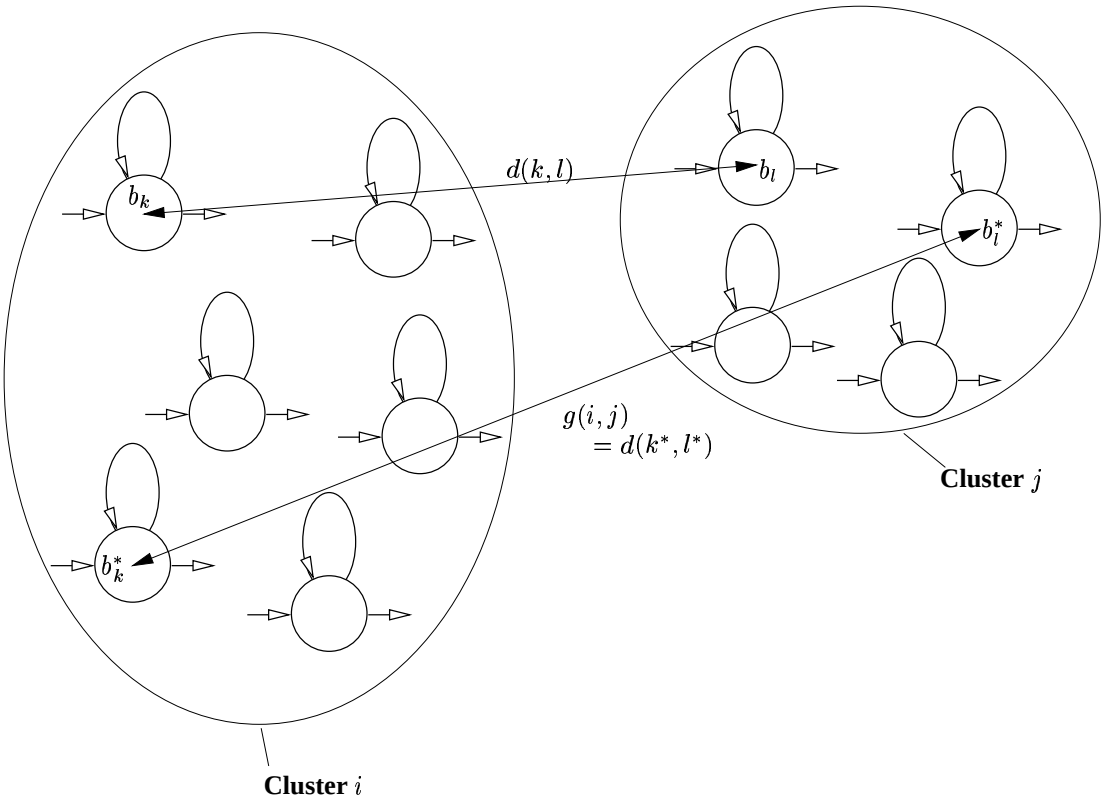


Abbildung 5.4: Datengetriebenes State-Clusterings

Nachdem bekannt ist, zu welchen Gruppen die HMM-Zustände jeweils gehören, können diese nun miteinander verschmolzen werden. Sämtliche Zustände innerhalb eines Clusters greifen dann auf einen repräsentativen Zustand zu, der die Eigenschaften aller beteiligten Zustände vereinigen sollte. Dies wird dadurch realisiert, dass ein beliebiger Zustand aus dem Cluster als physikalischer Zustand beibehalten wird, während alle anderen Zustände lediglich als Zeiger auf diesen einen physikalischen Zustand weiter geführt werden. Per Definition sind alle in einem Cluster enthaltenen Zustände und insbesondere deren Verteilungen recht ähnlich, wodurch diese vereinfachte Auswahl des physikalischen Zustandes möglich ist. Zudem werden nach dem Clustering die Modelle mit einigen Iterationen des Forward-Backward Algorithmus nachtrainiert. Die geclusterten Zustände werden dabei auf Grund der Verknüpfung von logischen und physikalischen Zuständen automatisch mit all den Zustands-bezogenen Beispielen trainiert die dem gesamten Cluster zuzuordnen sind. Damit wird schließlich garantiert, dass der den Cluster repräsentierende physikalische Zustand alle Charakteristika der beteiligten Zustände - in Abhängigkeit der jeweiligen Frequentierung - berücksichtigt.
Der Algorithmus zum datengetriebenen Zustands-Clustering läßt sich also in folgender Form zusammenfassen:

1. Erzeuge n Cluster für n Zustände

2. Suche ein (i, j) mit $(i, j) \in \{1, 2, \cdots, n\}$ welches $d_C(i, j)$ minimiert

3. Falls $d_C(i, j) < d_{C,max}$ und $n > 1$

   (a) verschmelze Cluster i mit Cluster j

   (b) dekrementiere n

   (c) weiter mit 2.

Theoretisch sind mit diesem Verfahren Verknüpfungen von Verteilungen oder Zuständen über verschiedene Zustandspositionen oder sogar Graphemklassen möglich. Da sich aber i. A. die diskriminativen Eigenschaften der Modelle verschlechtern, wenn deren Parameter über Graphembasisklassen hinweg verknüpft werden, ist es sinnvoll das Clustering nur auf verschiedene Modelle des gleichen Basisgraphems zu beschränken.
Mit dem datengetriebenen Clustering sollen die Ähnlichkeiten von Zuständen gefunden werden. Da sich rechtsseitige (linksseitige) Kontexteinflüsse insbesondere an den Modellanfängen (Modellenden) widerspiegeln, ist es zudem naheliegend, nur die Zustände zu verknüpfen, die an gleicher Position innerhalb verschiedener Grapheme des gleichen Basistyps liegen. Wie das Beispiel in Abb. 5.3 zeigt, werden nur Zustände in Position 1 mit anderen Zuständen in Position 1 verknüpft, während potenziell nur Zustände der Positionen 2 miteinander verknüpft werden usw.
Wenngleich dieses Verfahren bereits eine elegante Möglichkeit zur Konstruktion verallgemeinerter Allographeme darstellt, bleibt eine wesentliche Frage zunächst offen: was geschieht mit den kontextabhängigen Modellen, für die keine (oder nur sehr wenig) Trainingsdaten in entsprechender Kontextkombination existieren? Die Antwort auf diese Frage wird im nachfolgenden Abschnitt gegeben.
Um dies jedoch zu klären, sollte zunächst einmal der gesamte Ablauf zur Erzeugung geclusterter Allographeme zusammengefasst werden.

- Ausgehend von vollständig trainierten Monographemen werden auf Basis des später bei der Erkennung einzusetzenden Lexikons die Monographeme in entsprechende Allographeme kopiert. Praktischer Weise kann bereits hier ein Tying der Transitionsmatrizen von Allographemen des gleichen Basistyps erfolgen.

- Es folgen einige Iterationen des Forward-Backward-Algorithmus auf den Allographemen.

- Die Allographeme werden geclustert.

- Es folgen erneut einige Iterationen des Forward-Backward-Algorithmus auf den geclusterten Allographemen.

Die Expansion von Monographemen zu Allographemen wird zunächst ohne Kenntnis der Trainingsdaten und der damit verbundenen Graphemhäufigkeiten durchgeführt. Bei den nachfolgenden Forward-Backward Iterationen werden die Parameter der Allographeme mit unzureichendem oder fehlendem Trainingsmaterial unverändert mitgeführt und in dieser Form auch in den Clustering-Prozeß eingebracht. D. h., verglichen mit Monographemen ist der Gewinn an Genauigkeit bezogen auf die Testmenge auf solche Allographeme beschränkt, die sich in einer Schnittmenge aus Trainings- *und* Testdaten befinden. Sicherlich läßt sich annehmen, dass die am häufigsten vorkommenden Graphemkombinationen auch in der Trainingsmenge wiederzufinden sein müßten. Dennoch nimmt das Verhältnis von Schnittmenge aus Trainings- und potenziellen Testgraphemen zur Gesamtanzahl potenzieller Testgraphemen bei wachsendem Lexikon stark ab (vergl. Tab. 5.1).

5.4 Entscheidungsbaum basiertes State-Clustering

Es zeigt sich also, dass es bei der Verwendung kontextabhängiger Modelle in Verbindung mit einem sehr großen Vokabular vorteilhaft ist, die Vorzüge kontextueller Modellierung bei vorgegebener Trainingsmenge auch auf ungesehene Kontextkombinationen zu übertragen. Ein vielversprechender Ansatz dafür ist das Entscheidungsbaum-basierte State-Clustering [You94]. Mit der Verwendung von Entscheidungsbäumen wird es möglich, in kodierter Form a priori Wissen über den Einfluß von Kontextkombinationen auf die Graphemrealisierung in den Clustering-Prozeß einzubringen.
Verglichen mit dem datengetriebenen Clustering geht man hier genau einen entgegengesetzten Weg: Anstatt ähnliche Zustände zusammenzufassen, wird durch ein *split and merge*-Algorithmus versucht, möglichst unterschiedliche Zustände zuerst zu trennen und am Schluß die in einem Cluster verbleibenden Zustände zu verschmelzen (Abb. 5.5). Man nimmt dazu zunächst an, dass alle Zustände s in einem Cluster zu einem Zustand zusammengefasst werden. Anhand von Fragen q, die bestimmte Kontextgrapheme c der in einem Cluster enthaltenen Zustände betreffen, werden die Zustände dann in zwei Unter-Cluster aufgeteilt.
Eine solche aufspaltende Frage könnte z. B. lauten: 'Ist der rechte Kontext ein $/E/$ oder ein $/F/$?'. Zustände, für die eine angewendete Frage mit 'ja' ('nein') zu beantworten ist, werden den Teil-Clustern $\mathcal{S}^{(\mathcal{J})}$ (bzw. $\mathcal{S}^{(\mathcal{N})}$) zugeordnet. Der entstandene Cluster $\mathcal{S}^{(\mathcal{J})}$ enthält Zustände aus Graphemen die den nachgefragten Kontext c aufweisen. Der Cluster $\mathcal{S}^{(\mathcal{N})}$ nimmt alle übrigen Zustände der Menge $\mathcal{S}$ auf, d. h. die Zustände, deren Allographeme nicht den Kontext c besitzen. Nach wiederholter Aufteilung der entstehenden Unter-Cluster werden die in einem Ausgangsknoten des Baumes verbleibenden Zustände verschmolzen.
Jede Aufteilung sollte so vorgenommen werden, dass möglichst unterschiedliche Zustände getrennt werden. Nimmt man an, dass alle in einem Cluster enthaltenen Zustände zu einem

repräsentativen Zustand $\bar{s}$ verknüpft werden, bedeutet jede Aufsplittung des Clusters $\mathcal{S}$ eine vergrößerte Anzahl Modellparameter. Diese größere Anzahl führt zu einer detaillierteren Abbildung der Trainingsdaten und damit i. d. R. auch zu einer erhöhten Likelihood der Modelle auf den zugrunde liegenden Trainingsdaten. Um die Relevanz einer Aufsplittung und der damit verbundenen Frage q zu bewerten, eignet sich daher der log-Likelihood-Gewinn ΔL_q, der durch die Aufsplittung des Clusters $\mathcal{S}$ durch Frage q erzielt wurde.

$$\Delta L_q = L(\mathcal{S}^{(J)}(q)) + L(\mathcal{S}^{(N)}(q)) - L(\mathcal{S}) \tag{5.7}$$

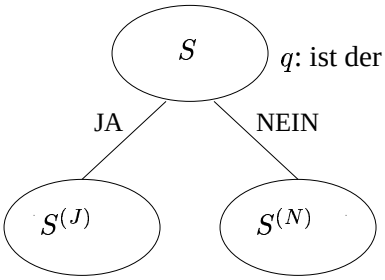
Die Bewertungsfunktion $L(\mathcal{S})$ eines Clusters ist die bedingte logarithmierte Wahrscheinlichkeit $\log p(X|\mathcal{S})$ dafür, dass die aus K Abtastwerten bestehende Trainingsmenge X durch das Cluster $\mathcal{S}$ erzeugt wurde. Dabei ist es zunächst unerheblich, ob X aus kontinuierlichen, oder aus diskreten, vektorquantisierten Merkmalssequenzen besteht.
Natürlich hängt $L(\mathcal{S})$ maßgeblich von den beteiligten Zuständen s ab. Wären innerhalb eines Clusters alle Zustände in gleichem Maße relevant, ließe sich $L(\mathcal{S})$ durch Summation der einzelnen bedingten log-Wahrscheinlichkeiten $\log p(X|s)$ bestimmen:

$$L(\mathcal{S}) = \sum_{s \in \mathcal{S}} \gamma \cdot \log p(X|s). \tag{5.8}$$

Dem Faktor γ kommt dabei die Rolle einer (konstanten) Gewichtung der beteiligten Wahrscheinlichkeiten $p(X|s)$ zu. Verallgemeinert man die Zusammensetzung des Clusters in der Form, dass die beteiligten Zustände unterschiedliche Relevanz für $L(\mathcal{S})$ einbringen können, kann die konstante Gewichtung durch eine zustandsabhängige Gewichtung γ_s ersetzt werden. $L(\mathcal{S})$ ließe sich dann approximieren durch

$$L(\mathcal{S}) = \sum_{s \in \mathcal{S}} \gamma_s \log p(X|s) = \sum_{s \in \mathcal{S}} \gamma_s \sum_{k=1}^{K} \log p(x_k|s). \tag{5.9}$$

Naheliegend ist es, die zustandsabhängige Gewichtung γ_s über die jeweilige Zustandsfrequentierung zu definieren [You94]. Unter der Voraussetzung, dass die Segmentgrenzen der



q: ist der rechte Kontext ein $/E/$ oder $/F/$?

$S^{(J)}$: Zustände aus Modellen, deren rechter Kontext ein $/E/$ oder ein $/F/$ ist

$S^{(N)}$: Zustände aus Modellen, deren rechter Kontext kein $/E/$ und kein $/F/$ ist

Abbildung 5.5: Cluster-Splitting

Zustände durch das Clustering quasi nicht verschoben werden, ließe sich die jeweilige Zustandsfrequentierung aus einem - dem Clustering vorausgehenden - Viterbi-Alignment bestimmen. Als $\gamma_s(x_k)$ kann damit die Wahrscheinlichkeit betrachtet werden, dass das Merkmal x_k von dem Zustand s erzeugt wurde. Unter Anwendung der Viterbi-Segmentierung ergäbe sich ein $\gamma_s(x_k)$ von 1 oder 0, abhängig davon, ob ein Frame x_k dem Zustand s zugeordnet wurde oder nicht.
Führt man sich den gesamten allgemeinen Ablauf des Clusterings vor Augen (vergl. Abschnitt 5.3), ergibt sich eine sehr elegante Alternative zu dem zusätzlichen (relativ teuren) Viterbi-Alignment: Da vor dem eigentlichen Clustering ohnehin einige Iterationen des Forward-Backward-Algorithmus durchgeführt werden, sollte es doch möglich sein, die Frame-Zustands-Zuordnung aus der letzten Iteration des FB-Algorithmus vor dem Clustering zu nutzen. Damit verließe man den deterministischen Standpunkt, den eine Viterbi-Segmentierung mit sich brächte und erhielte so 'echte', d. h. kontinuierliche Wahrscheinlichkeitswerte für $\gamma_s(x_k)$ zwischen $0,0$ und $1,0$.
Werden die Wahrscheinlichkeiten $\gamma_s(x_k)$ nun über alle K Abtastwerte aufsummiert, ergibt sich die Zustandsfrequentierung γ_s mit

$$\gamma_s = \sum_{k=1}^{K} \gamma_s(x_k) \tag{5.10}$$

für den Zustand s, mit der Normierungsbedingung

$$\sum_{s} \gamma_s(x_k) = 1, \tag{5.11}$$

wobei über alle Zustände s des Gesamtsystems summiert wird. D. h. γ_s ist letztlich die Häufigkeit mit der Zustand s besucht wurde. Wird die log-Likelihood des Clusters $\mathcal{S}$ durch

$$L(\mathcal{S}) = \sum_{s \in \mathcal{S}} \log p(X|s) \sum_{k=1}^{K} \gamma_s(x_k) = \sum_{s \in \mathcal{S}} \sum_{k=1}^{K} \log p(x_k|s) \gamma_s(x_k) \tag{5.12}$$

geschätzt, ergibt sich für den Fall kontinuierlicher Merkmale und Ausgabeverteilungen als weitere Vereinfachung, dass $L(\mathcal{S})$ mit

$$L(\mathcal{S}) = -\frac{1}{2}(\log[(2\pi)^n |\Sigma(\mathcal{S})|] + n) \sum_{s \in S} \sum_{k=1}^{K} \gamma_s(x_k), \tag{5.13}$$

nur noch von der Varianz $\Sigma(\mathcal{S})$ des Clusters $\mathcal{S}$, von $\gamma_s(x_k)$ und von der Dimensionalität n der Merkmalsvektoren abhängt [You94]. Der Wert für die Zustandsfrequentierung γ_s entsprechend Glg. 5.10 läßt sich für jeden Zustand s aus der vorangegangenen FB-Iteration aus den Vorwärts- und Rückwärtsvariablen berechnen und temporär speichern. $\Sigma(\mathcal{S})$ wiederum kann aus den Mittelwerten und Varianzen der beteiligten Zustände unter Berücksichtigung von γ_s bestimmt werden, wodurch eine effiziente Berechnung von $L(\mathcal{S})$ möglich wird.

Für den diskreten Fall kann wiederum von der in Glg. 5.12 skizzierten Grundidee ausgegangen werden, wobei der Merkmalsvektor $\vec{x}(k)$ zum Abtastzeitpunkt k von dem Vektorquantisierer in das diskrete Codewort $y_n(k)$ transformiert wird. Die n-te VQ-Partition bringt damit den Anteil $L_n(\mathcal{S})$ in die Gesamt-Likelihood der Menge $\mathcal{S}$ ein. Mit

$$L_n(\mathcal{S}) = \sum_{s\in\mathcal{S}} \sum_{k=1}^{K} \log p(y_n(k)|s)\gamma_s(y_n(k)) \tag{5.14}$$

resultiert für den diskreten Fall ein

$$L(\mathcal{S}) = \sum_{n=1}^{N_C} L_n(\mathcal{S}) = \sum_{s\in\mathcal{S}} \sum_{n=1}^{N_C} \sum_{k=1}^{K} \log p(y_n(k)|s)\gamma_s(y_n(k)). \tag{5.15}$$

Die Summe über k kann wiederum zusammengefasst werden, sodass die diskrete Ausgabeverteilung $b_s(y_n) = p(y_n|s)$, gewichtet mit der Zustandsfrequentierung $\gamma_s(y_n)$ des Zustands s durch Codewort y_n ein

$$L(\mathcal{S}) = \sum_{s\in\mathcal{S}} \sum_{n=1}^{N} \log b_s(y_n)\gamma_s(y_n) \tag{5.16}$$

ergibt. Hierbei ist zu betonen, dass es sich bei $L(\mathcal{S})$ um die Gesamt-Likelihood des Clusters *vor* dem Tying der einzelnen Zustände handelt. D. h. das Cluster ist zunächst nichts weiter als eine Gruppe einzelner Zustände mit separaten Parametersätzen. Durch ein Tying würden die einzelnen Verteilungen $b_s(y_n)$ durch eine repräsentative Verteilung $b_{\bar{s}}(y_n)$ ersetzt, die zu einer log-Likelihood

$$L(\bar{s}) = \gamma_{\bar{s}} \sum_{n=1}^{N_C} \log b_{\bar{s}}(y_n) \tag{5.17}$$

des Zustandes $\bar{s}$ führt.

Aus der Summation über alle N_C Codewörter in Glg. 5.16 resultiert des weiteren die totale Zustandsfrequentierung γ_s mit

$$\gamma_s = \sum_{n=1}^{N_C} \gamma_s(y_n). \tag{5.18}$$

Die Normierung der codewortbedingten Zustandsfrequentierung $\gamma_s(y_n)$ mit der totalen Zustandsfrequentierung γ_s liefert umgehend die diskrete Ausgabeverteilung $b_s(y_n)$ von Zustand s:

$$b_s(y_n) = \frac{\gamma_s(y_n)}{\gamma_s}. \tag{5.19}$$

In analoger Weise läßt sich die repräsentative Wahrscheinlichkeitsdichtefunktion $b_{\bar{s}}(y_n)$ berechnen

$$b_{\bar{s}}(y_n) = \frac{\sum_{s\in\mathcal{S}} \gamma_s(y_n)}{\sum_{s\in\mathcal{S}} \gamma_s}, \tag{5.20}$$

die sich praktischer Weise - unter Ausnutzung von Glg. 5.19 - als gewichtete Summe der Einzel-WDF bestimmen läßt:

$$b_{\bar{s}}(y_n) = \frac{\sum_{s\in\mathcal{S}} \gamma_s \cdot b_s(y_n)}{\sum_{s\in\mathcal{S}} \gamma_s}. \tag{5.21}$$

Umgestellt nach $\gamma_s(y_n)$ kann der Zusammenhang aus Glg. 5.19 in Glg. 5.16 verwendet werden, wodurch sich folgende interessante Berechnungsvorschrift für $L(\mathcal{S})$ mit

$$L(\mathcal{S}) = \sum_{s \in \mathcal{S}} \gamma_s \underbrace{\sum_{n=1}^{N_C} b_s(y_n) \log b_s(y_n)}_{-H_s}, \tag{5.22}$$

bzw. für $L(\bar{s})$ eines geclusterten Zustandes $\bar{s}$ zeigt:

$$L(\bar{s}) = \gamma_{\bar{s}} \underbrace{\sum_{n=1}^{N_C} b_{\bar{s}}(y_n) \log b_{\bar{s}}(y_n)}_{-H_{\bar{s}}}. \tag{5.23}$$

Die zweite Summe von Glg. 5.22 läßt sich eindeutig als negative Entropie H_s der diskreten Verteilung aus Zustand s identifizieren. Auch hier schließt sich also der Kreis: Der Likelihood-Verlust, der sich aus der Verschmelzung der Zustände $\mathcal{S}$ zu dem repräsentativen Zustand $\bar{s}$ ergibt, entspricht einem Informationsverlust

$$\Delta H = -\sum_{s \in \mathcal{S}} \gamma_s H_s + \gamma_{\bar{s}} H_{\bar{s}}. \tag{5.24}$$

Umgekehrt gilt natürlich, dass der Likelihood-Gewinn durch eine Aufspaltung von $\mathcal{S}$ in $\mathcal{S}^{(J)}$ und $\mathcal{S}^{(N)}$ mittels Frage q durch

$$\Delta L_q = \gamma_{\mathcal{S}^{(J)}} H_{\mathcal{S}^{(J)}} + \gamma_{\mathcal{S}^{(N)}} H_{\mathcal{S}^{(N)}} - \gamma_{\mathcal{S}} H_{\mathcal{S}} \tag{5.25}$$

ermittelt werden kann. Auch hier zeigt sich also, dass die für das Clustering erforderlichen Berechnungen allein auf der Grundlage der Ausgabeverteilungen der beteiligten Zustände und deren Frequentierungen durchgeführt werden können. Zusammenfassend kann die log-Likelihood $L(\bar{s})$ eines aus der Menge $\mathcal{S}$ hervorgegangenen Zustandes $\bar{s}$ mittels der Gleichungen 5.23 und 5.21 berechnet werden.

Um eine optimierte Modellstruktur zu finden, ist es sinnvoll eine Reihe optionaler Fragen für das Splitting eines Knotens anzubieten. Unter den möglichen Fragen q wird die für die nächste Aufspaltung anzuwendende Frage q^* ausgewählt, indem unter allen möglichen Fragen diejenige selektiert wird, die den Likelihood-Gewinn ΔL maximiert:

$$q^* = \arg\max_q \Delta L_q. \tag{5.26}$$

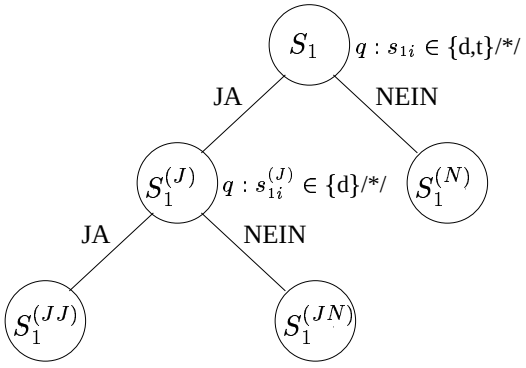
Die ausgewählte Frage kommt dann zur Anwendung, wenn der Likelihood-Gewinn einen gewissen minimalen Schwellwert überschreitet:

$$\Delta L_{q^*} \geq \hat{\Delta L}. \tag{5.27}$$

Ist diese Bedingung für eine optimale Frage nicht mehr erfüllt, wird das Splitting für den aktuellen Knoten abgebrochen und die dort verbliebenen Zustände werden verknüpft.

Ähnlich wie bei dem datengetriebenen Clustering kann auch bei dem Baum-basierten Clustering eine minimale Zustandsanzahl pro Cluster gefordert werden, um die Trainierbarkeit der generalisierten Modelle zu gewährleisten. Diese minimale Zustandsanzahl kann als zusätzliches Abbruchkriterium für das Clustering genutzt werden, indem für die potenziellen Unter-Cluster $\mathcal{S}^{(J)}$ und $\mathcal{S}^{(N)}$ die Prüfung auf minimale Zustandsanzahl in logischer UND-Verknüpfung positiv ausfallen muß. Andernfalls wird auch hierbei das Splitting abgebrochen.
Wurde nun ein Baum für eine Menge von Zuständen durch sukzessive Aufspaltung expandiert, kann die Struktur weiter verfeinert werden, indem für jede Paarung der entstandenen Terminalknoten geprüft wird, ob deren Verknüpfung mit einem vertretbaren Likelihood-Verlust erkauft werden könnte. Ist dies der Fall, erfolgt deren Zusammenfassung. Dazu wird wiederum der Schwellwert $\hat{\Delta L}$ verwendet, der schon als Abbruchkriterium bei der Aufspaltung verwendet wurde.

Nachdem die Frage nach der Bewertung einer Frage und deren Auswahl geklärt ist, ist die praktische Vorgehensweise bei dem Entscheidungsbaum-basierten Clustering noch näher zu beleuchten. Abb. 5.6 zeigt dazu den Vorgang an einem konkreten Beispiel. Auch bei dem Baum-basierten Clustering gilt im Regelfall, dass die Zustände positionsweise geclustert werden. Das Beispiel bezieht sich auf die Trigrapheme $d/e/r$, $d/e/s$, $i/e/s$ und $t/e/r$. Die Zustände aus den jeweiligen HMM mit der Position i werden in der Menge bzw. dem Knoten $\mathcal{S}_i$ gesammelt. Auf der Basis jedes Wurzelknotens wird anhand der Fragen $\mathcal{Q}$ ein Binärbaum gebildet. In Abb. 5.6 wird auf den Knoten $\mathcal{S}_1$ zuerst eine Frage q angewendet. In dem gezeigten Beispiel spaltet die Frage nach den rechten Kontextgraphemen $(q : s_{1i} \in \{d, t\}/*/)$ den Knoten $\mathcal{S}_1$ in eine Menge $\mathcal{S}_1^{(J)}$ und eine komplementäre Menge $\mathcal{S}_1^{(N)}$ auf. $\mathcal{S}_1^{(J)}$ enthält entsprechend der gewählten Frage alle Zustände, deren Grapheme die rechtsseitigen Kontextgrapheme $/d/$ oder $/t/$ aufweisen. Knoten $\mathcal{S}_1^{(N)}$ enthält alle übrigen Zustände aus Position 1 des Zentralgraphems $/e/$. In dem gezeigten Beispiel enthält $\mathcal{S}_1^{(J)}$ also die Zustände der Position 1 der Trigrapheme $d/e/r$, $d/e/s$ und $t/e/r$. $\mathcal{S}_1^{(N)}$ enthält Zustand 1 aus dem Trigraphem $i/e/s$.
Das Beispiel in Abb. 5.6 gibt zudem einen Einblick, wie sich welcher Kontext zusammen mit der Position der Zustände eines Knotens auf die Entwicklung des entsprechenden Baumes auswirkt.
Als erstes wurde für den Knoten $\mathcal{S}_1$, wie auch für alle übrigen Wurzelknoten, die Frage nach dem linken Kontext gestellt (ist linkes Kontextgraphem ein /d/ oder ein /t/?). Dies ist im übrigen bezeichnend: da im Allgemeinen auch bei anderen Zentralgraphemen zuerst Fragen nach dem linken Kontext gewählt werden (weil diese einen entsprechend großen Likelihood-Gewinn einbringen), kann davon ausgegangen werden, dass das vorausgehende Zeichen einen besonders starken Einfluß auf die Realisierung der Zentralgrapheme ausübt.

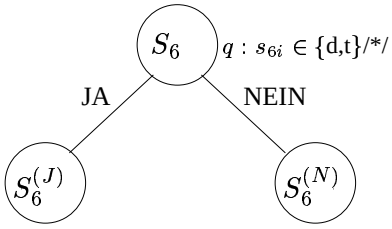


$S_1^{(J)}$ = {d,t}/e/* = {d/e/r, d/e/s, t/e/r}

$S_1^{(N)}$ = {i/e/s}

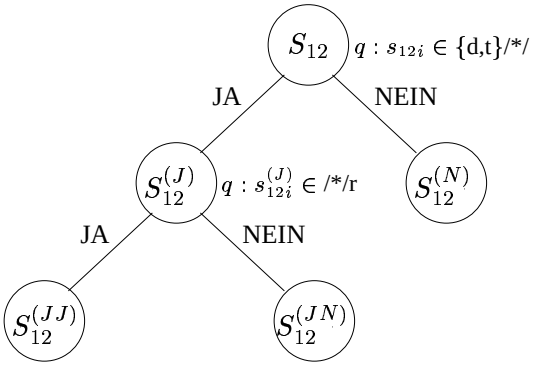
$S_1^{(JJ)}$ = {d/e/r, d/e/s}

$S_1^{(JN)}$ = {t/e/r}



$S_6^{(J)}$ = {d,t}/e/* = {d/e/r, d/e/s, t/e/r}

$S_6^{(N)}$ = i/e/s



$S_{12}^{(J)}$ = {d,t}/e/*

$S_{12}^{(N)}$ = i/e/s

$S_{12}^{(JJ)}$ = {d,t}/e/r = {d/e/r, t/e/r}

$S_{12}^{(JN)}$ = {d/e/s}

Abbildung 5.6: Prinzip des Entscheidungsbaum basierten State-Clusterings

Des weiteren ist zu beobachten, dass die Relevanz kontextbedingter Entscheidungen zur Aufspaltung von Clustern zur Modellmitte (6. Zustand) hin zunächst abnimmt, um zum Modellende (12. Zustand) hin wieder anzusteigen. Dies wird durch die abflachende Tiefe der Bäume zur Modellmitte angezeigt. Eine allzu große Diversifizierung der Zustände ist dort scheinbar nicht sinnvoll. Dieser Effekt ist mit den Kontext-erfassenden Eigenschaften der Merkmalsextraktion zu begründen, da diese Eigenschaft gerade am Modellanfang und -ende zu einer höheren Inkonsistenz der Merkmale führen muß.

Darüber hinaus nimmt der oben erwähnte Einfluß der vorausgehenden Kontext-Grapheme

zum Modellende hin ab. Dies läßt sich daran ablesen, dass mit fortschreitender Position der Zustände in den Modellen zunehmend Fragen ausgewählt werden, die den rechten Kontext betreffen. In Abb. 5.6 ist dies exemplarisch an der Aufspaltung des Knotens $\mathcal{S}_{12}^{(J)}$ zu sehen.

5.4.1 Einbeziehung typographischer Eigenschaften

Bei sämtlichen Betrachtungen in dem vorangegangenen Abschnitt wurde davon ausgegangen, dass die für das Clustering notwendigen Fragen bekannt sind. Da diese Fragen und die damit verbundenen Graphemgruppen jedoch möglichst unterschiedliche Kontexteinflüsse berücksichtigen sollen, ist deren Formulierung keineswegs trivial. Während aus der Linguistik solche Kontexteinflüsse für phonetische Realisierungen der gesprochenen Sprache bekannt sind und dieses Wissen auch für den Aufbau kontextabhängiger Phonem-Modelle eingesetzt wird, gibt es in der Literatur bisher wenig Hinweise über Kontexteinflüsse von Sub-Worteinheiten auf handgeschriebene Sprache.
Einen wesentlichen Einfluß haben sicherlich Großbuchstaben auf die Ausführung nachfolgender Zeichen. Die im Abschnitt B.2 gezeigten Beispiele lassen einen solchen Zusammenhang vermuten, da gerade Großbuchstaben - auch bei einer ansonsten eher kursiven Schreibweise - tendenziell eher in Druckschrift ausgeführt werden. Zwei mögliche Graphemklassen aus denen sich Fragen nach dem Kontext ableiten lassen sind demnach Groß- und Kleinbuchstaben.
Eine weitere Unterteilung bietet sich bei den Kleinbuchstaben an. Hier ist es z. B. denkbar, dass die Zeichenhöhe und die vertikale Position - ähnlich wie bei den Großbuchstaben - zu Graphemvariationen führt. Dementsprechend ließen sich die Klassen der *in-line*-Zeichen (Zeichen ohne Ober- oder Unterlängen), der Oberlängen-behafteten Zeichen, und der Buchstaben mit Unterlängen definieren.
Im Sinne maximaler Benutzbarkeit wurde schon bei der Datenakquisition auf die Definition bestimmter Regeln für die Schriftgenerierung verzichtet. Bedingt dadurch ist es in den Datenbasen häufig zu beobachten, dass zuerst eine Sequenz von Buchstabenkörpern geschrieben wird, und erst einige Zeichen später sog. diakritische Zeichen gesetzt werden. Dazu gehören die Punkte über einem ’i’ oder einem ’j’, sowie Punkte über Umlauten und t-Striche. Ein Einfluß diakritischer Zeichen auf Umgebungsbuchstaben ist daher ebenfalls naheliegend.
Diese Überlegungen führen zusammengefasst zu den folgenden Graphemklassen, aus denen sich kontextbezogene Fragen ableiten lassen:

Klasse 1: A, B, C, . . . Z, Ä, Ö, Ü

Klasse 2: a, c, e, m, n, o, r, s, u, v, w, x, z

Klasse 3: b, d, h, k, l, ß

Klasse 4: f, g, p, q, y

Klasse 5: i, j, t, ä, ö, ü

Beispielsweise kann eine aus diesen Klassen abgeleitete Frage demzufolge lauten: 'Ist der rechte Kontext ein Graphem der Klasse 1'?
Im Vorgriff auf die später dargestellten Ergebnisse ist anzumerken, dass dieser Ansatz eine eher mäßige Erkennungsgenauigkeit liefert. Von diesem Standpunkt aus wäre darum vergleichsweise eher der datengetriebene Ansatz zu favorisieren. Jedoch kann andererseits auch nicht garantiert werden, dass mit den oben dargestellten Kontextklassen eine optimale Grundlage für das Entscheidungsbaum-basierte Clustering gefunden wurde. Vielmehr ist anzunehmen, dass diese relativ einfache Unterteilung die Kontexteinflüsse noch nicht in vollem Maße berücksichtigt.
Im folgenden sollen daher verstärkt selbstorganisierende Verfahren entwickelt werden, die es ermöglichen, auf Basis der vorhandenen Daten das fehlende graphonomische Wissen zu generieren. Diese Verfahren werden in den nachfolgenden Abschnitten näher beschrieben.

5.4.2 Clustering Markov'scher Quellen

Grundlegende Idee dieses Ansatzes ist es, gewisse Ähnlichkeiten unter Graphemen zu finden, die anhand des Schriftbildes nicht ohne weiteres ersichtlich sind, sich aber in der gewählten Merkmalsrepräsentation wiederfinden. Naheliegend wäre es also, die Merkmalssequenzen verschiedener Grapheme in der Trainingsdatenbasis zu vergleichen. Ein direkter Vergleich der Merkmalssequenzen brächte jedoch die folgenden zwei wesentlichen Probleme mit sich:

- Zum einen ist zu beachten, dass es relativ schwierig ist, einzelne Merkmalssequenzen, die zum Vergleich herangezogen werden könnten, als repräsentativ zu bezeichnen. Die stark variierenden Realisierungsformen verhindern dies.

- Zum anderen erschweren aber auch die äußerst uneinheitlichen Merkmalssequenz-Längen einen direkten Vergleich durch herkömmliche Abstandsmaße.

Eine interessante Alternative bietet sich allerdings mit einem indirekten Vergleich der Merkmalssequenzen über die Monograph-HMM, da diese - in statistisch abstrahierter Form - ein gemitteltes aber dennoch relativ detailiertes Abbild der Trainingsdaten darstellen. Hierzu werden die HMM zu Markov'schen Quellen umfunktioniert [Kos98d]. Abb. 5.7 zeigt dazu eine Möglichkeit wie sich HMM als Markov'sche Quellen nutzen lassen. Werden die Tran-

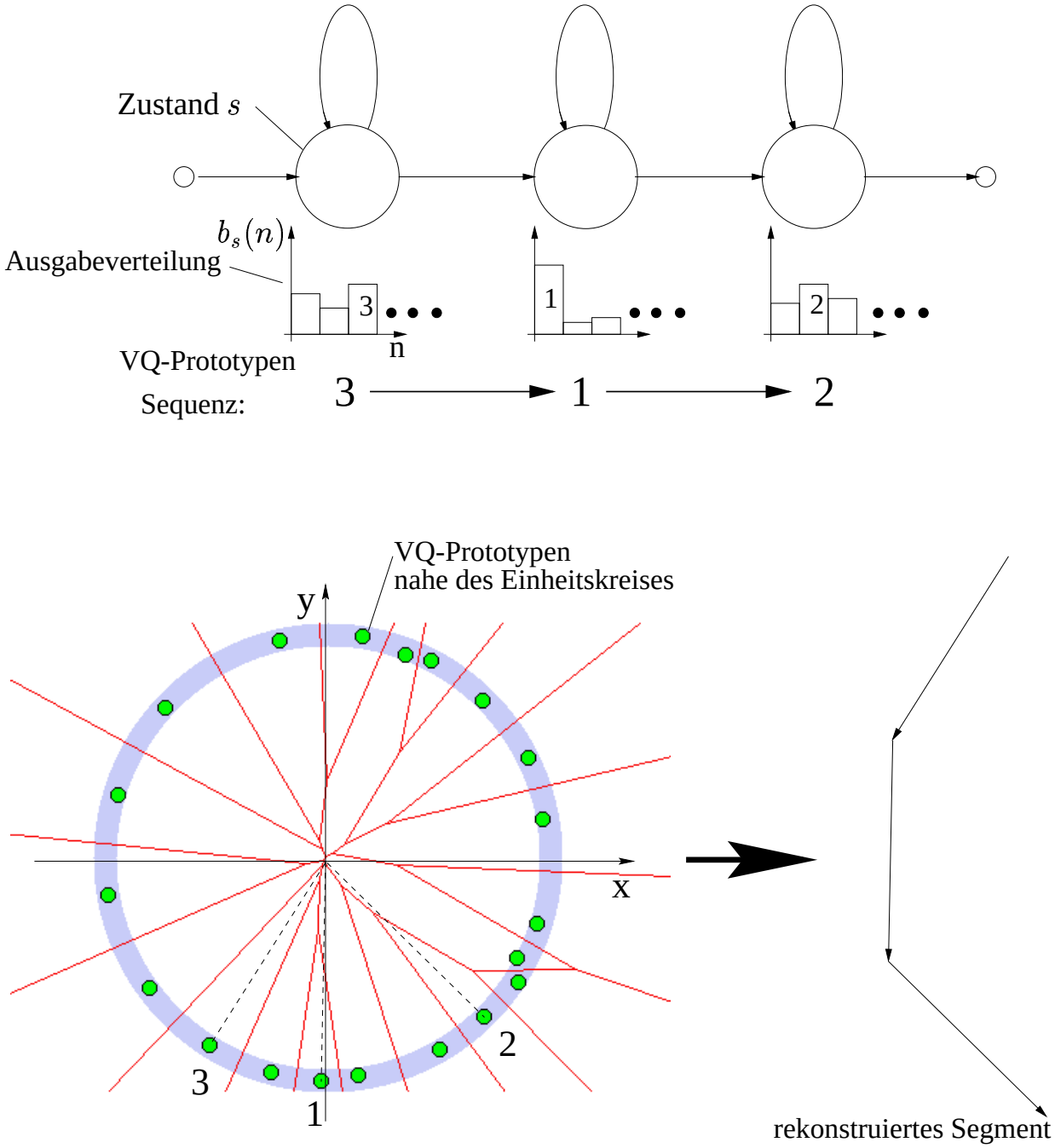


Abbildung 5.7: Clustering Markov'scher Quellen

sitionswahrscheinlichkeiten der HMM und die damit verbundenen Verweilzeiten in den einzelnen Zuständen vernachlässigt, lassen sich für jedes Graphem typische, längennormierte Trajektorien ermitteln, die anschließend problemlos einem selbstorganisierenden Minimum-Abstands-Klassifikator zugeführt werden können.

Zur Rekonstruktion der Stifttrajektorie anhand der als Quelle genutzten HMM bietet sich insbesondere die als Merkmal genutzte Orientierung der Tangente der Stifttrajektorie (α-Merkmalsstrom) an. Dies ist dadurch zu erklären, dass die Stifttrajektorie zunächst räumlich unterabgetastet wird. Die Abtastvektoren interpolieren dabei quasi-äquidistante Abtastpunk-

te. Im Umkehrschluß kann also die Trajektorie lediglich anhand der Winkelsequenz der Abtastvektoren angenähert werden, da ja der Abstand aufeinander folgender Abtastpunkte als Konstante bekannt ist. Da jede dieser Orientierungen wiederum in je einen $\cos$- und $\sin$-Wert kodiert wird (um die Unstetigkeit bei 2π zu beheben), sind die Merkmale demzufolge auf dem Einheitskreis verteilt. Ein weiterer Vorteil der Nutzung der α-Merkmale ist, dass die nur zweidimensionalen Eingangsvektoren durch ein Codebuch der Größe 30 quantisiert werden. Der entstehende nachteilige Quantisierungsfehler hält sich damit in Grenzen, was sich letztendlich positiv auf das Rekonstruktionsergebnis auswirkt.
Um die Rekonstruktion der Trajektorie anhand der Abb. 5.7 zu verdeutlichen, sei zunächst angenommen, dass die HMM aus drei Zuständen bestehen. Anhand der Maximumsuche

$$n_s^* = \arg\max_n b_s(n) \tag{5.28}$$

wird das Codewort n_s^* mit der höchsten Wahrscheinlichkeit aus der Ausgabeverteilung $b_s(n)$ für jeden Zustand s innerhalb eines HMM gesucht. Aus den oben dargelegten Gründen wird diese Suche ausschließlich auf den Ausgabeverteilungen der α-Merkmale durchgeführt. Die Wahrscheinlichkeit für ein Codewort n ist natürlich ein direktes Maß für die Häufigkeit des korrespondierenden n-ten Codebuchvektors (im betreffenden Zustand). Darauf basierend läßt sich nun eine Sequenz von VQ-Prototypen für jedes HMM bestimmen. Diese synthetisierte VQ-Sequenz läßt sich in Verbindung mit dem Codebuch des α-Merkmals nutzen um die Stifttrajektorie des modellierten Graphems zu rekonstruieren. Die Codewörter stellen die (numerierten) Cluster des partitionierten Merkmalsraumes dar. Im Falle des α-Merkmals besteht jeder Codebuchvektor aus zwei Komponenten, die den $\sin$- und $\cos$-Werten der Abtastvektororientierungen in dem Einheitskreis entsprechen. Diese Codebuchvektor-Komponenten, multipliziert mit der konstanten Abtastvektorlänge ergeben die Beiträge in x- und y-Richtung zu dem interpolierten Trajektorienstück. Die Trajektorienstücke lassen sich dann zustandsweise zu einem synthetischen Graphem zusammenfügen indem die Beiträge in x- und y-Richtung inkrementell an das Ende des jeweils vorherigen Vektors angefügt werden.
Abb. 5.8 zeigt einige der auf diese Weise rekonstruierten Grapheme. Jede dieser rekonstruierten Grapheme repräsentiert eine charakteristische Kurvenform. Die uniforme Länge erlaubt es schließlich Standard-Clustering Methoden, wie z. B. den k-means Algorithmus anzuwenden, um durch unüberwachtes Lernen Graphemklassen zu identifizieren.
Das hier zugrundeliegende Prinzip zur Bestimmung von Kontextklassen beruht auf der Annahme, dass Nachbargrapheme, die auf Grund bestimmter gemeinsamer Eigenschaften als ähnlich gelten, auch ähnliche Auswirkungen auf ein Zentralgraphem zeigen. Ist dies der Fall, lassen sich aus den ermittelten Graphemklassen natürlich Fragen für das Baum-basierte Clustering ableiten, indem danach gefragt wird, ob der Kontext zu einer bestimmten Graphemgruppe gehört.

Die folgende Tabelle zeigt das Ergebnis des Clusterings.

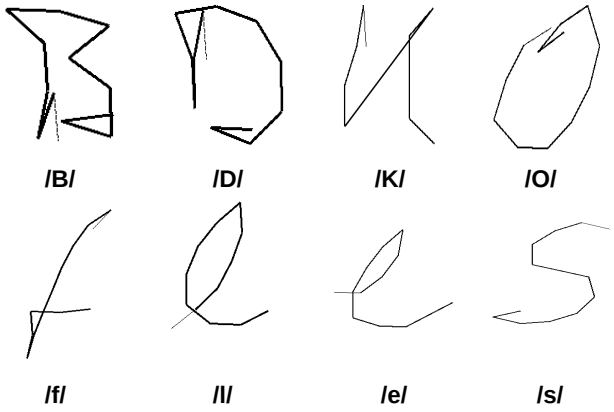


Abbildung 5.8: Beispiele rekonstruierter Grahpeme

Klasse 1: A, D, J, M, Y

Klasse 2: B, C, F, G, H, I, K, L, P, R, S, U, Z

Klasse 3: E, N, O, Q, T, V, W, X, Ä, Ö, Ü

Klasse 4: a, d, j, k, n, o, p, q, r, s, t, v, w, x, ö, ü, ß

Klasse 5: b, c, f, h, i, l, u, ä

Klasse 6: e, g, m, y, z

Hierbei wurden die repräsentativen Sequenzen mittels des k-means Algorithmus zu 2x3 Gruppen eingeordnet, wobei Groß- und Kleinbuchstaben jeweils getrennt gruppiert wurden. Verglichen mit dem in Abschnitt 5.4.1 vorgestellten heuristischen Ansatz stehen durch den selbstorganisierenden Ansatz insgesamt sechs anstatt der fünf zuvor definierten Klassen zur Verfügung. Auch die Gruppenaufteilung scheint eine komplett andere Struktur hervorgebracht zu haben als bei der heuristischen Aufteilung. Ein Teil der Ergebnisse des Clusterings scheint aus graphonomischer Sicht einleuchtend zu sein. Dies ist z. B. der Fall bei den Graphemen /C/ und /G/ der Klasse 2, deren Trajektorien über weite Strecken durchaus Ähnlichkeiten zugeschrieben werden könnten. Gleiches ließe sich beispielsweise auch für das /O/ und /Q/ als eine Teilklasse, sowie für das /N/, das /V/ und das /W/ als eine weitere Teilgruppe aus Klasse 3 postulieren.
Selbst unter der Berücksichtigung der Bildung von Teilgruppen scheinen andere Klassenzuordnungen wiederum weniger offensichtlich. Jedoch ist hier nochmals zu unterstreichen, dass die Gruppierungen auf der Basis der beobachteten Merkmale durchgeführt wurden, und dies nicht zwangsweise mit dem Schriftbild gleichzusetzen ist, das sich dem menschlichen Betrachter offenbart.

Zusammenfassend läßt sich feststellen, dass die Ähnlichkeiten aus einem Teil des Merkmalsraumes abgeleitet werden konnten, der auch bei der allgemeinen Modellbildung kontextabhängiger wie kontextunabhängiger HMM relevant ist und darum gegenüber einem heuristischen Ansatz gewisse Vorteile bieten sollte.

5.4.3 Tree-Sweeping

Ein weiterer untersuchter Ansatz zur Generierung von Kontextklassen ist ein zweistufiger, ebenfalls selbstorganisierender Prozeß, der wiederum auf dem Entscheidungsbaum-basierten Clustering beruht. In einem ersten Clustering-Durchlauf werden auch hier die Graphemklassen zusammengefasst, die in einer zweiten Stufe dann als Grundlage für das eigentliche Clustering der HMM-Zustände verwendet werden (Abb. 5.9). In der ersten Stufe wird zunächst ein Baum-basiertes Clustering auf die vorhandenen Monographeme angewendet. In der zweiten Stufe werden anhand der generierten Fragen die Zustände der expandierten n-Grapheme verknüpft.
Wesentlicher Unterschied zu dem im vorherigen Abschnitt beschriebenen Verfahren der Markov'schen Quellen ist, dass die Zusammenfassung der Monographeme zu Kontextklassen entsprechend einer Maximum-Likelihood Optimierung durchgeführt wird. Damit ist es möglich über den Vergleich der Kurveneigenschaften hinaus zu gehen und die Betrachtungen auf sämtliche zur Verfügung stehenden Merkmale auszuweiten.
Die Ausgangssituation in der ersten Stufe ist, dass zunächst alle Zustände der vorhandenen Monographeme in einem Wurzelknoten akkumuliert werden. Im Unterschied zu dem allgemeinen Baum-basierten Verfahren werden die Entscheidungsbäume nicht getrennt zustandsweise expandiert. Ausgehend von dem Wurzelknoten wird das Splitting so lange vorangetrieben, wie ein bestimmter Wert für $\hat{\Delta L}$ überschritten wird. Die Aufspaltung wird dabei anhand primitiver Fragen vorgenommen, welche die Zustände der enthaltenen Monograph-

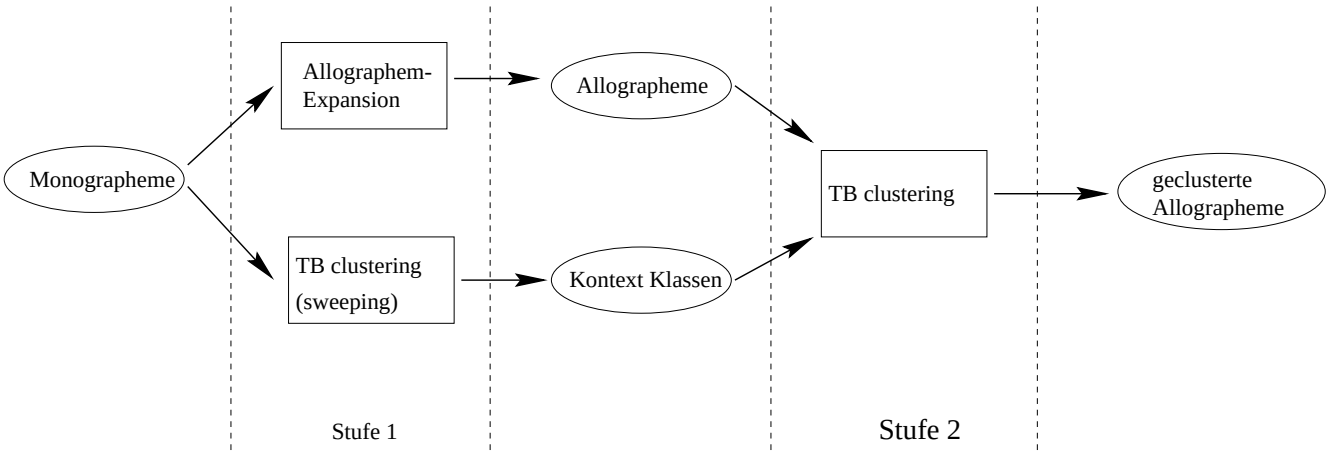


Abbildung 5.9: Übersicht zum Entscheidungsbaum basierten Clustering

eme betreffen. Z. B. ist eine solche potenzielle Frage, ob ein Zustand zum Monographem /E/ gehört ($q : s_{ij} \in /E/$). Betrachtungen des Kontextes bleiben in dieser Stufe zunächst außen vor. Die Auswahlmechanismen der Fragen sind identisch mit denen des allgemeinen Baum-basierten Clusterings.

Die sukzessive Anwendung solcher Fragen bringt die im oberen Teil in Abb. 5.10 gezeigte Baum-Struktur hervor, in der an jedem Knoten die Zustände des nachgefragten Monographems abgespalten werden (wenn ein entsprechender Likelihood-Gewinn gegeben ist). Diese aus den Zuständen eines Monographems bestehenden Ausgangsknoten seien als primitive Knoten bezeichnet. Nach vollständiger Aufspaltung, d. h., wenn der durch $\hat{\Delta L}$ spezifizier-

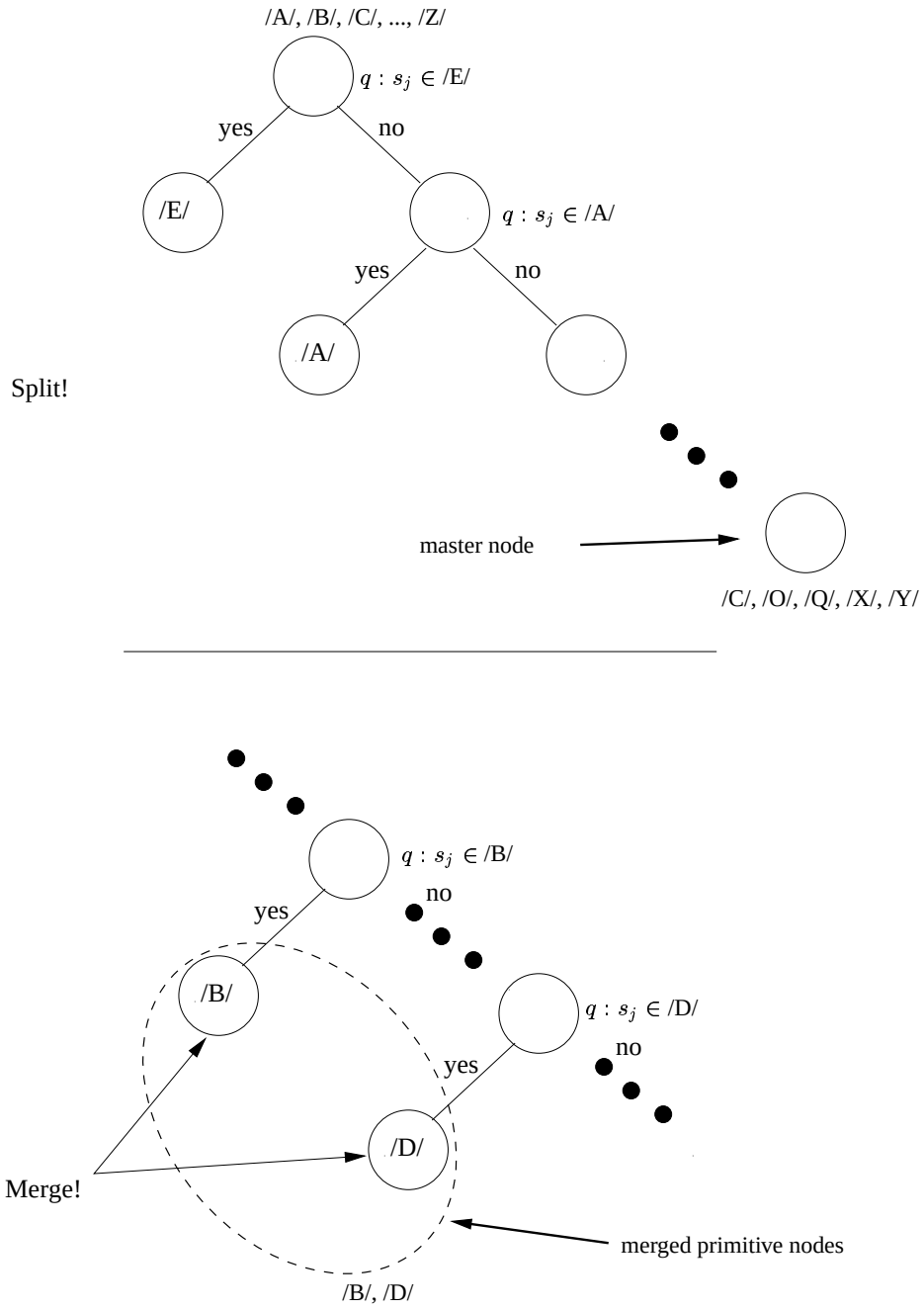


Abbildung 5.10: Split-and-Merge von Zustands-Clustern

te minimal geforderte Likelihood-Gewinn durch weitere Abspaltungen nicht mehr erreicht werden kann, verbleibt ein nicht-primitiver Ausgangsknoten auf unterster Baum-Ebene. Dies sei der sog. Master-Knoten.
Um zusätzliche nicht-primitive Cluster mit einem Baum zu generieren, wird wie im unteren Teil in Abb. 5.10 gezeigt, in einem *Merging*-Durchlauf versucht die primitiven Cluster weiter zusammenzufassen. Für jede mögliche Cluster-Paarung wird geprüft, ob deren Verknüpfung einen Likelihood-Verlust unterhalb der spezifizierten Schranke $\hat{\Delta L}$ mit sich bringt. Ist dies der Fall, erfolgt die Verknüpfung der beteiligten Cluster. Die Verknüpfung der Zustände in den nicht-primitiven Knoten des entstandenen Baumes zu einem allgemeinen, Graphemklassen-übergreifenden HMM würde auf diese Weise zu einem vergleichsweise geringen Likelihood-Verlust führen.
Die ausgebildete Struktur des Baumes - insbesondere dessen Tiefe - hängt unter anderem entscheidend von dem gewählten Parameter $\hat{\Delta L}$ ab. Durch das sog. *Sweeping* des $\hat{\Delta L}$-Parameters innerhalb eines bestimmten Bereiches ist es möglich eine Vielzahl verschiedener Baumstrukturen, bzw. Graphemkombinationen zu erzeugen. Aus allen erzeugten Baum-Strukturen zwischen den Extremen der völligen Verknüpfung aller Modelle und der totalen Aufspaltung in jeweils einzelne HMM werden alle auftretenden nicht-primitiven Cluster gespeichert. Diese Sammlung nicht-primitiver Graphemklassen läßt sich anschließend wiederum als Grundlage für Kontext-bezogene Fragen verwenden.
In dieser Weise wird für Groß- und Kleinbuchstaben getrennt verfahren. Ein Teil der so erzeugten Graphemklassen ist in Abb. 5.11 gezeigt. Bei den dargestellten Graphemgruppen handelt es sich um die entstandenen Master-Knoten. Die Pfeile in der Abbildung deuten die Abhängigkeit der Zusammensetzung eines Master-Knotens von dem Parameter $\hat{\Delta L}$ an. Abb. 5.11 verdeutlicht darüber hinaus, dass Master-Knoten, die unter einem kleineren $\hat{\Delta L}$ entstanden sind, stets eine Teilmenge größerer Master-Knoten sind.

5.5 Ergebnisse

Zur Evaluierung kontextbezogener Modellierungstechniken sind umfangreiche Datenbasen erforderlich. Es ist zu bedenken, dass die kontextbedingten Effekte in möglichst vielfältigen Kombinationen mehrfach bei einem Schreiber zu beobachten sein sollten. Da bei der schreiberabhängigen Datenbasis die Wortanzahl pro Schreiber im Trainings-Set um den Faktor 10 größer ist, als bei der schreiberunabhängigen Datenbasis, werden die Experimente zunächst an dem schreiberabhängigen System durchgeführt. Von jedem der drei Schreiber wurden zu Trainingszwecken ca. 2000 Wörter gesammelt. Es besteht damit die Hoffnung, dass beobachtbare Kontexteinflüsse in gewissem Maße als repräsentativ gelten.
Tab. 5.2 zeigt die erzielten Ergebnisse. Zu Vergleichszwecken ist in Tab. 5.2 in Zeile 1

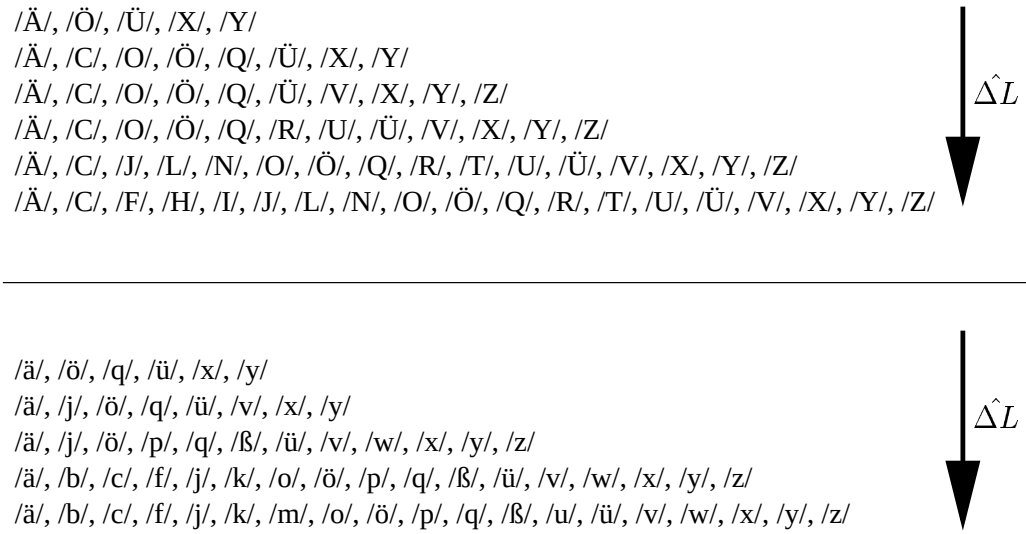


Abbildung 5.11: Entwicklung der Master-Knoten durch Tree-Sweeping

das Ergebnis des diskreten Basissystems, mit kontextunabhängigen Monographemen aufgeführt. Die Spalten 2 bis 4 geben die Erkennungsraten für die einzelnen Schreiber wieder. Die 5. Spalte zeigt die durchschnittliche Erkennungsrate mit der relativen Fehlerreduktion in Klammern, bezogen auf das Monographemsystem in Zeile 1. Spalte 6 gibt Auskunft über die durchschnittliche Anzahl von Modellen *nach* dem Clustering. Die Modellanzahl in Spalte 6 ist mit den entsprechenden Werten aus Tab. 5.1 zu vergleichen, in der die Modellanzahl in Abhängigkeit des verwendeten Lexikons und des berücksichtigten Kontextes aufgelistet ist. Die Ergebnisse in Tab. 5.2 sind mit dem 30k-Lexikon erzielt worden. Die Zeilen 2-6 zeigen die Ergebnisse unter Verwendung von Trigraphemen. Die Anzahl der Trigrapheme vor dem Clustering beträgt demnach 8900. Die 2. Zeile in Tab. 5.2 gibt die erzielten Ergebnisse für die selektiv hinzugefügten Trigrapheme wieder (Abschnitt 5.2.1). Die Ergebnisse mit kontextabhängigen Modellen geben die maximalen Erkennungsraten, die mit optimaler Einstellung der jeweiligen Parameter erzielt werden konnten, wieder. Mit dem datengetriebenen Clustering der Trigraphem-Zustände (Abschnitt 5.3) konnte die durchschnittliche Er-

	ank	jmr	vdm	ø	Anzahl HMMs
Monographeme	93.0 %	77.9 %	83.4 %	84.8 %	80
selektives Verfahren	93.6 %	79.1 %	89.8 %	87.5 (17.8) %	138
datengetrieben	96.2 %	82.9 %	94.7 %	91.3 (42.8) %	96
Baum, heuristisch	95.7 %	90.9 %	93.6 %	93.4 (56.6) %	416
Baum (WI), Markov-Quelle	96.2 %	92.0 %	93.6 %	93.9 (59.9) %	403
Baum (WD), Markov-Quelle	96.2 %	91.4 %	94.6 %	**94.1** (61.1) %	434

Tabelle 5.2: Erkennungsergebnisse - Vergleich verschiedener Reduktionsansätze, schreiberabhängig, 30K Lexikon

kennungsrate auf 91,3 % verbessert werden. Die drei letzten Zeilen in Tab. 5.2 zeigen die Ergebnisse mit Entscheidungsbaum-basiertem Clustering. An der Anzahl verbleibender Modelle ist bei dem Entscheidungsbaum-basierten Clustering ein interessanter Effekt abzulesen. Die größere Anzahl läßt darauf schließen, dass die kontextabhängigen Modelle deutlich detailierter ausgeführt werden, als bei den übrigen Verfahren, was schließlich zu einer vergleichsweise hohen Erkennungsrate führt.
Zunächst ist in Zeile 4 das Ergebnis wiedergegeben, welches mit heuristisch angesetzten Fragen (Abschnitt 5.4.1) zu erreichen war. Die Ergebnisse in Zeilen 5 und 6 beziehen sich auf die Generierung von Fragen durch Markov'sche Quellen (Abschnitt 5.4.2). Interessant sind die geringen Differenzen zwischen der letzten und der vorletzten Zeile. Für die Ergebnisse der letzten Zeile wurden für jeden Schreiber individuelle Fragen, bzw. Bäume für das Clustering generiert. Die Generierung der Fragen wurde dabei mit den schreiberabhängigen Monographemen durchgeführt. Die vorletzte Zeile beschreibt die Ergebnisse, die mit einer allgemeinen Menge von Fragen erzielt wurden. Dieser allgemeine Satz von Fragen wurde wiederum aus einem schreiberunabhängigen System generiert und einheitlich für jeden Schreiber verwendet. Dieser Vergleich legt den Schluß nahe, dass es selbst für schreiberabhängige Systeme möglich ist, eine allgemeingültige Menge von Fragen zu entwickeln, ohne dafür nennenswerte Verluste an Genauigkeit hinnehmen zu müssen.
Die selbstorganisierende Generierung von Fragen zeigt gegenüber den übrigen Verfahren die höchste Korrektheit. Es zeigt sich darüber hinaus, dass über die zur Verfügung gestellten Fragen ein gewisses Optimierungspotenzial gegeben ist. Um schließlich dass optimale Verfahren zu finden, soll ein abschließender Vergleich der Markov'schen Quellen und des Tree-Sweepings anhand des 200k-Lexikons durchgeführt werden. Wie Tab. 5.1 zeigt, stellt eine solche Lexikongröße aufgrund der sehr stark steigenden Modellanzahl eine beträchtliche Herausforderung an kontextorientierte Modellierungsformen.
Bei einer derartigen Größe des Lexikons stellt sich zudem die Frage, ob die generelle Verwendung von Trigraphemen in jedem Fall die beste Lösung darstellt. Die Ergebnisse in Tab.

		ank	jmr	vdm	ø	Anzahl HMMs
Bigrapheme links	Markov'sche Quelle	94.62	84.49	89.84	89.65	284
	Tree-Sweeping	95.70	83.42	89.84	89.65	351
Bigrapheme rechts	Markov'sche Quelle	93.55	83.42	89.84	88.93	341
	Tree-Sweeping	93.55	82.89	88.77	88.40	292
Trigrapheme	Markov'sche Quelle	95.16	84.49	89.84	89.83	603
	Tree-Sweeping	94.62	86.63	90.37	90.54	633

Tabelle 5.3: Vergleich verschiedener Kontextbereiche, schreiberabhängig, 200k Lexikon

5.3 sollen diese Fragen beantworten. Die erste Spalte in Tab. 5.3 gibt Auskunft über den berücksichtigten Kontext. Die zweite Spalte bezeichnet das Verfahren zur Generierung von Fragen. Sowohl mit den Markov'schen Quellen, wie auch mit dem Tree-Sweeping wurde je ein generell angewendeter Satz von Fragen für alle drei Schreiber aus dem schreiberunabhängigen System generiert. Bei einem Vergleich zwischen links-abhängigen Bigraphemen und rechts-abhängigen, zeigt sich eine höhere Korrektheit bei der Berücksichtigung des linken Nachbargraphems. Dies ist in sofern einsehbar, da der aktuell geschriebene Buchstabe offensichtlich stärker von dem vorausgehenden Nachbarn beeinflußt wird, als von seinem Nachfolger. Bei der Verwendung links-abhängiger Bigrapheme läßt sich zunächst kein Einfluß auf die verwendeten Fragen feststellen. Markov'sche Quellen zeigen hingegen einen Vorteil bei rechts-abhängigen Bigraphemen. Die höchsten Erkennungsraten liefert schließlich die Verwendung von Trigraphemen, wobei sich hier insbesondere die aus dem Tree-Sweeping hervorgegangenen Fragen als geeignet erweisen. Die höhere Erkennungsrate bei Trigraphemen läßt sich wiederum damit begründen, dass Trigrapheme die Vorteile von rechts- und links-abhängigen Bigraphemen vereinen. Wenn auch links-Bigrapheme dem dominierenden Einfluß vorausgehender Buchstaben gerecht werden, können diese weniger gut die Abhängigkeiten bei Großbuchstaben erfassen, da diese in der Regel lediglich einen rechten Kontext aufweisen.
Mit dem hier beschriebenen Verfahren des Tree-Sweepings läßt sich bei optimiertem $\hat{\Delta L}$ die Erkennungsrate des schreiber*un*abhängigen Systems mit einem Wortschatz von 2000 Wörtern von 90,6 % auf 93,2 % verbessern.

5.6 Kapitelzusammenfassung

Mit der Einführung kontextabhängiger Modelle eröffnet sich die Möglichkeit die Vielfältigkeit der Handschrift auf komplexere Modellstrukturen abzubilden, als es mit kontextunabhängigen Modellen möglich wäre. Durch entsprechende Verknüpfung von Modellparametern wird auf selbstorganisierende Weise eine optimierte individuelle Modellstruktur gefunden. Mit der Parameterverknüpfung wird gleichzeitig die Parameteranzahl auf ein sinnvolles Maß reduziert, sodass die Trainierbarkeit der Modelle - auch bei einer begrenzten Anzahl von Trainingsbeispielen - gegeben ist.
Insbesondere bei der Verwendung sehr großer Lexika spielt die Parameterverknüpfung eine entscheidende Rolle. Die meisten potenziell möglichen Graphemkombinationen sind bei sehr großem Vokabular nicht in der Trainingsmenge vorhanden. Das Entscheidungsbaumbasierte Clustering bietet dabei gegenüber dem datengetriebenen Clustering besondere Vorteile, da es durch die entstehende Baumstruktur auch 'ungesehene' Graphemkombinationen synthetisiert.
Für die Entwicklung der Baumstruktur ist wiederum die Bildung von Graphemklassen erfor-

derlich, die unterschiedliche Kontexteinflüsse widerspiegeln. Diese Graphemklassen wurden anhand verschiedener Verfahren generiert. Ein abschließender Vergleich der Verfahren hat trotz der relativ kleinen Datenbasen die Überlegenheit kontextueller Modellierung gezeigt.

Kapitel 6

Textmodellierung

Bisher realisierte Ansätze für die Online-Handschrifterkennung ermöglichen entweder

1. die Erkennung ohne vorgegebenen Wortschatz, was dafür aber die Eingabe *einzelner* Druckschrift-ähnlicher Buchstaben erfordert. Hier werden die Buchstaben einzeln erkannt und später aneinandergereiht. Oder sie ermöglichen

2. die verbundene Eingabe von Buchstabenfolgen, mit der Beschränkung auf bestimmte Worte (festes Vokabular).

Bei der Handschrifterkennung müßte demnach grundsätzlich zwischen einem Lexikon-freien Modus bei Eingabe segmentierbarer Einzelzeichen, und einem Lexikon-basierten Modus mit der Option zur Eingabe verbundener Zeichen ausgewählt werden.
Die zweite Variante wurde in den bisherigen Kapiteln verfolgt. Um diese Beschränkung soweit wie möglich abzuschwächen, wurde dabei versucht die Lexikongröße zu maximieren. Gerade im Deutschen mit der intensiven Nutzung von Komposita wird es hingegen immer wieder auffallen, dass ein Lexikon nie vollständig ist.
Der entscheidende Vorteil des hier vorgestellten Verfahrens ist nun die Kombination der Vorteile beider Modi [Kos99c]. Es kann Text unter Verwendung verbundener Buchstaben eingegeben werden. Der Benutzer ist dabei jedoch nicht auf ein festes Lexikon, bzw. auf einen definierten Wortschatz beschränkt. D. h., dass die Kombination von HMM und Textmodellen hier erstmals eine automatische schreiberunabhängige Handschrifterkennung ermöglicht ohne besondere Voraussetzungen bezüglich des verwendeten Vokabulars oder der Schriftart (Font). Als Textmodelle werden hier statistische Modelle (N-Gramme) bezeichnet, die nicht wie üblich auf Wortbasis, sondern auf Einzelzeichenbasis trainiert werden.

6.1 Verknüpfung von Graphem- und Textmodellen

Für die Erkennung von Wörtern oder Sätzen mit großem Vokabular (einige tausend Wörter) gilt folgender klassischer Ansatz als Stand der Technik:
Die einzelnen Buchstaben werden mit HMM anhand von Trainingsbeispielen modelliert. In einem Lexikon, welches den erlaubten Wortschatz definiert, ist ein gültiges Wort durch die Folge seiner Buchstaben bzw. HMM beschrieben. Mit einer Viterbi-Dekodierung kann bei der Erkennung eines unbekannten Wortes, welches durch die Abtastsequenz X repräsentiert ist, das wahrscheinlichste Wort W^* aus allen Worten W gefunden werden.

$$W^* = \arg\max_W P(X|W) \tag{6.1}$$

Die Suche erfolgt hier unter Berücksichtigung des gewählten Vokabulars um den Berechnungsaufwand zu begrenzen und die Fehlerrate zu minimieren. Für die Erkennung ganzer Sätze oder Wortfolgen werden zusätzlich Sprachmodelle eingesetzt [Mak94, Nat95]. Diese Sprachmodelle sind im wesentlichen Wahrscheinlichkeiten für bestimmte Wortfolgen W:

$$P(W) = \prod_{i=1}^{m} P(w_i|w_{i-N+1}, ..., w_{i-1}). \tag{6.2}$$

$P(w_i|w_{i-N+1}, ..., w_{i-1})$ ist dabei die Wahrscheinlichkeit, dass dem Wort w_i die Wortfolge $w_{i-N+1}, ..., w_{i-1}$ voraus ging. N ist die Kontexttiefe dieses sog. N-Grams. Mit der Viterbi-Dekodierung kann bei der Erkennung das Sprachmodell und die HMM gleichzeitig ausgewertet werden. Der erkannte Satz W^* wird dabei wie folgt bestimmt:

$$W^* = \arg\max_w P(W) \cdot P(X|W) \tag{6.3}$$

Die wesentliche Innovation ist nun die Verwendung eines Textmodells an Stelle eines festen Lexikons und eines optionalen Sprachmodells. Das Textmodell ist im wesentlichen strukturiert wie ein Sprachmodell, beschreibt aber nicht die Wahrscheinlichkeiten von Wortfolgen sondern die von Buchstabensequenzen ($P(c_i|c_{i-N+1}, ..., c_{i-1})$). Diese Beschreibung kann allerdings über eine wesentlich größere Kontexttiefe erfolgen als bei Wort-basierten Sprachmodellen. Während bei Sprachmodellen gewöhnlich Kontexttiefen von zwei oder drei eingesetzt werden, sind bei Textmodellen Kontexttiefen von 12 oder höher realisierbar. Die höheren Kontexttiefen bei Textmodellen beziehen sich allerdings auf die Länge von Buchstabenfolgen, während sich die Kontexttiefe bei Sprachmodellen auf die Wortfolgenlänge bezieht.
Ein systemimmanentes Problem des Viterbi-Dekoders ist der Umgang mit N-Grammen mit großer Kontexttiefe ($N \geq 3$). Um eine effizientere Auswertung solch komplexer Textmodelle zu erzielen, kann z. B. ein *Stack-Dekoder* [Wil98] verwendet werden. Das Textmodell - insbesondere mit hoher Kontexttiefe - macht auf diese Weise ein festes Lexikon überflüssig.

Indem das Textmodell mit entsprechend großer Kontexttiefe über Wortgrenzen hinweg berechnet wird, sind die Auftrittswahrscheinlichkeiten der Wortgrenzen im Textmodell enthalten. D. h., dass sowohl eine Erkennung von Sätzen bzw. Wortfolgen möglich ist, wie auch die Erkennung einzelner Worte. Das potentielle Auftreten von Wortgrenzen wird, wie in Abschnitt 6.2.2 beschrieben, durch eine entsprechende Aufbereitung der Trainingsmenge gesteuert.

6.2 Modellebenen

Wie bereits erwähnt, werden HMM und Textmodelle wie in Glg. (6.3) angegeben, gemeinsam während der Erkennung anhand der beobachteten Handschriftmerkmale X, bzw. im diskreten Fall anhand der vektorquantisierten Merkmale Y, evaluiert. Bei der Dekodierung, bzw. der Erkennung sind daher lediglich die zwei Modellarten Graphemmodell und Textmodell involviert, wie sie in den folgenden Abschnitten beschrieben werden.

6.2.1 Graphemmodelle

Bisher war das Graphem- oder Zeichenmodell als Hidden Markov Modell dargestellt. Anzumerken ist hier, dass statt des HMMs bei diesem Ansatz ebenso ein allgemeines statistisches Graphemmodell verwendet werden kann, welches die bedingte Wahrscheinlichkeit (Likelihood) $P(X|C)$ der Beobachtungsfolge X unter der Bedingung der Wort- bzw. Zeichenfolge C liefert. Das Verfahren ist demnach nicht an die Art der Zeichenmodelle gebunden. Es muß lediglich sichergestellt werden, dass während der Erkennung die verwendeten Graphemmodelle effizient gemeinsam mit dem Textmodell ausgewertet werden können.

6.2.2 Textmodelle

Allgemein formuliert beschreibt ein Textmodell die Wahrscheinlichkeit $P(c_i|c_{i-N+1}, ..., c_{i-1})$ eines Buchstabens c_i unter der Bedingung einer bestimmten ihm vorausgehenden Buchstabenkette oder Historie $c_{i-N+1}, ..., c_{i-1}$ (Glg. (6.2)). Die Struktur ähnelt demnach der eines Sprachmodells, bzw. einer statistischen Grammatik, welche auf Wortfolgen basiert. Mit zunehmender Kontexttiefe vergrößert sich exponentiell die Anzahl der Textmodellparameter. Bei einer Kontexttiefe von $N = 3$ und einem Zeichenvorrat N_c von 80 verschiedenen Zeichen, ergibt sich theoretisch pro Zeichen eine Kontextmenge von $N_c^{N-1} = 80^2 = 6400$ verschiedener Kombinationen. Für eine Kontexttiefe von $N = 6$

erhöht sich dieser Wert bereits auf $80^5 \approx 3,3 \cdot 10^9$. Um die Vielzahl der Parameter zuverlässig zu schätzen, werden solche Modelle i. d. R. auf sehr großen Textdatenbasen trainiert, die mehrere Millionen Sätze beinhalten. Zudem werden zur Glättung der Wahrscheinlichkeiten und zur Vermeidung von *sparse data*-Effekten, *Cut-Off-, Diskontierungs*- und *Back-Off*-Strategien [Cla97] bei der Parameterschätzung der Textmodelle angewendet. Prinzipiell können hier sämtliche Verfahren zur Verfeinerung und Glättung der Textmodelle verwendet werden, wie sie auch von der Optimierung von Sprachmodellen bekannt sind.
Der Unterschied zwischen Textmodellen zur Satz- und zur Worterkennung soll nachfolgend anhand eines Beispiels erläutert werden. Dazu wird angenommen, dass die Trainingsmenge für das Textmodell aus folgendem Satz besteht:

```
Ein Trigramm-Sprachmodell besteht aus relativen
Trigramm-, Bigramm- und Unigramm-Häufigkeiten.
```

Für die Satzerkennung wird jeweils eine gesamte Textpassage verarbeitet. Satzanfang und -ende werden speziell behandelt, was hier nicht weiter vertieft werden soll. Bei einem Trigramm ($N = 3$) ergeben sich für die exemplarische Trainingsmenge beginnend bei dem ersten 'n' folgende Kontextkombinationen:

```
(n|i,E), (<sp>|n,i), (T|<sp>,n), (r|T,<sp>), (i|r,T),
(g|i,r), (r|g,i), (a|r,g), (m|a,r), (m|m,a),
(-|m,m), (S|-,m), (p|S,-), ...
```

Dieses Beispiel macht deutlich, dass bei entsprechend hoher Kontexttiefe statistische Zusammenhänge über Wortgrenzen (hier mit $< sp >$ bezeichnet) hinweg erfasst werden können. Sogar der Einfluß von Satzzeichen und Wortgrenzen wird in diese spezielle Form der statistischen Orthografie einbezogen.
Für den Einzelwortmodus wird die Datenbasis in modifizierter Form zum Training verwendet. Die einzelnen Textpassagen werden dazu an den auftretenden *white spaces* in einzelne Wörter zerlegt:

```
Ein
Trigramm-Sprachmodell
besteht
aus
relativen
Trigramm-,
Bigramm-
und
Unigramm-Häufigkeiten.
```

Unter spezieller Behandlung der Wortanfänge und -Enden werden für jedes einzelne Wort mit anhängenden Satzzeichen die Kontextkombinationen extrahiert und deren Häufigkeiten analysiert. Unter Vernachlässigung der Sonderbehandlung von Wortanfang und -Ende ergeben sich im Vergleich zum ersten Beispiel folgende Kontextkombinationen:

```
(n|i,E), (i|r,T), (g|i,r), (r|g,i), (a|r,g), (m|a,r),
(m|m,a), (-|m,m), ...
```

Ein Wortende ($< sp >$) inmitten einer Äußerung wird somit nicht ermöglicht, wohl aber die zusätzliche Eingabe von Satzzeichen nach den Einzelworten. Die letztendliche Struktur des Textmodells bleibt die gleiche, da es sich hier lediglich um Häufigkeiten, bzw. um Wahrscheinlichkeiten von Kontextkombinationen handelt. In den Erkenner muß bei einer Umschaltung zwischen den beiden Modi lediglich das relevante Textmodell nachgeladen werden.

6.3 Dekodierung

Wie bereits oben angedeutet, stellt die Verarbeitung komplexer N-Gramme eine besondere Herausforderung an den als Erkenner eingesetzten Dekoder. Zur Verarbeitung komplexer N-Gramme bietet sich z. B. der Einsatz eines *Stack-Decoders* an, wie er in [Ren95a, Wil98] beschrieben wurde. Exemplarisch soll nachfolgend eine Übersicht zur Arbeitsweise des Dekoders im zeitsynchronen Modus gegeben werden. Abb. 6.1 stellt den grundlegenden Algorithmus des Stack-Decoders mit zeitsynchroner Stackexpandierung dar. Wie in Schritt a) beschrieben, wird zunächst jedem Zeitpunkt k, zu dem ein Merkmalsvektor existiert, ein

a) Initialisierung des nullten Stacks mit der leeren Hypothese und der Stacks 1 bis K als leere Stacks, $k := 0$
b) Auswahl des Stacks des Zeitpunktes k
c) Falls das Äußerungsende erreicht ist ($k = K$), ist die beste Hypothese des Stacks aus b) die gesuchte Wortfolge
d) Erweitere die Hypothesen des gewählten Stacks um weitere Worte unter Berücksichtigung der bedingten Wortwahrscheinlichkeiten und der Graphemmodelle. Potentielle Wortenden definieren neue Hypothesen, sie werden dem jeweiligen Stack zugefügt.
e) $k := k + 1$
f) weiter bei b)

Abbildung 6.1: Stackalgorithmus mit zeitabhängigen Stacks und zeitsynchroner Stackbearbeitung

leerer Stack zugeordnet. Dieser Stack wird später im Laufe der Dekodierung mit Hypothesen aufgefüllt. Als Hypothese H wird eine Teildekodierung der Beobachtungssequenz X bis zum Zeitpunkt k_H bezeichnet. Die Hypothese H besteht im wesentlichen aus der hypothesenspezifischen Historie der Zeichenfolge C_H und wird als Objekt zusammen mit dem Wahrscheinlichkeitswert der Hypothese S_{C_H} (*Score*) auf dem Stack des entsprechenden Zeitpunktes k_H abgelegt (Abb. 6.2). Der Score S_{C_H} wird entsprechend Glg. (6.2) und (6.3) für Einzelzeichenfolgen berechnet: $S_{C_H} = P(C) \cdot P(X|C)$. Die Auftrittswahrscheinlichkeit $P(X|c)$ für ein einzelnes Zeichen kann z. B. mit einer Viterbi-Approximation und einer zeitsynchronen Strahlsuche [Rab89] bestimmt werden. Die Hypothesen werden auf den jeweiligen Stacks nach ihrem Score in aufsteigender Reihenfolge abgelegt, sodass die wahrscheinlichste Hypothese ($H_i^{(k_H)}$ in Abb. 6.2) oben aufliegt. Anschließend wird ein zu bearbeitender Zeitpunkt ausgewählt (Schritt b)). Ist das Ende der Beobachtungssequenz erreicht (Schritt c)), wird die oberste Hypothese zum Zeitpunkt K als das Erkennungsergebnis ausgegeben. Andernfalls wird mit der Stackexpandierung fortgefahren, indem die Hypothese des nächsten Stacks unter Berücksichtigung von Graphem- und Textmodell um ein Zeichen $c_i^{(k')}$ verlängert wird ($||c_i^{(k')}$). Die so entstandenen neuen Hypothesen werden - entsprechend der geschätzten zusätzlichen Zeichendauer - auf nachfolgenden Stacks abgelegt (Schritt d)). Die Schleife wird anschließend für nachfolgende Stacks wiederholt (Schritte e) und f)). Dabei ist jedoch zu beachten, dass i. A. nicht jeder nachfolgende Stack untersucht werden muß. Es können, wie in Schritt b) angedeutet, gezielt Auswahlmechanismen eingesetzt werden, die den Dekodierungsprozeß wesentlich beschleunigen [Wil98]. So werden z. B. erst die nachfolgenden Stacks selektiert, deren beste Hypothesen einen einstellbar abfallenden Wahrscheinlichkeitswert überschreiten. Außerdem kann davon ausgegangen werden, dass sich in zeitlich benachbarten Stacks überwiegend ähnliche Hypothesen befinden, deren Auswertung vernachlässigt werden kann ohne allzu große Fehler zu erzeugen, was wiederum zu einer beschleunigten Erkennung führt. Zusätzliche Pruningmechanismen werden auch in der Strahlsuche zur Einzelzeichenerkennung eingesetzt, sodass auch auf dieser Ebene eine Beschleunigung erzielt werden kann.

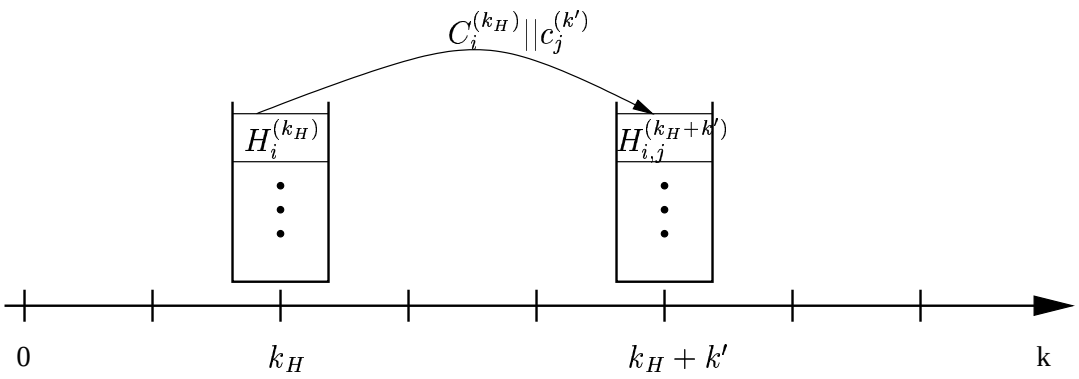


Abbildung 6.2: Anordnung der Stacks und Zuordnung der Hypothesen

6.4 Ergebnisse und Kapitelzusammenfassung

Abschließend bleibt festzuhalten, dass der Vorteil des Stack-Dekoders für diese Aufgabe konzeptionell zu begründen ist, wobei die vielen verschiedenen Pruningmöglichkeiten in adäquater Adjustierung das Dekodierverhalten bezüglich der Laufzeit positiv beeinflussen. Ein wesentlicher Vorteil, den dieses Konzept mit sich bringt, ist die Tatsache, dass der Zugriff auf das Textmodell unabhängig von der Strahlsuche zur Einzelzeichenerkennung erfolgt. Bei einer Expandierung einer Hypothese wird zunächst die Wahrscheinlichkeit der Graphemmodelle $P(X|C)$ zu dem bekannten aktuellen Score multipliziert und abschließend wird der Textmodellanteil aufmultipliziert. Im Vergleich dazu würde der Speicher- und Berechnungsaufwand exponenziell wachsen, wenn derart komplexe N-Gramme direkt in die Strahlsuche integriert würden, da der Suchraum über sämtliche gegebene Kontextkombinationen aufgebaut werden müßte. Hier ließe sich die Auswertung von Graphem- und Textmodell nicht so elegant entkoppeln. Diese Tatsache dürfte im wesentlichen begründen, warum mit einer Strahlsuche maximal Kontexttiefen von $N = 3$ verarbeitet werden können.
Auf der anderen Seite zeigt Tab. 6.1, dass die Ausweitung des betrachteten Kontextes auf mehr als drei Zeichen unerlässlich ist, um die Erkennungsrate zu maximieren. Der Bedarf nach effizienten Dekodierverfahren, die in der Lage sind solche Kontextbereiche zu verarbeiten, wird anhand dieser Ergebnisse deutlich. Während bei dem schreiberunabhängigen System bei einer Kontexttiefe von $N = 3$ die Korrektheit der insgesamt 27025 Einzelzeichen im Testset der 4134 Einzelwörter noch bei 81,3 % liegt, kann dieser Wert bei der Verwendung von 9-Grammen auf 91,4 % gesteigert werden. Bei weiterer Vergrößerung der Kontexttiefe auf 12 nimmt die Erkennungsrate wiederum leicht ab. Die Verbesserung der Akkuratheit, bei der zusätzlich zur Korrektheit (Glg. (3.34)) noch die Anzahl der fehlerhaften Einfügungen berücksichtigt wird, spielt sich in der gleichen Größenordnung ab. Die jeweils etwa 50-prozentige relative Fehlerreduktion bei Erhöhung der Kontexttiefe von 3 auf 9 wird insbesondere bei der Worterkennungsrate relevant. Hier kann die Korrektheit von 39,2 % auf 73,5 % erhöht werden.

	3-Gramm	6-Gramm	9-Gramm	12-Gramm
Korrektheit (27025 Zeichen)	81.3 %	90.6 %	91.4 %	91.3 %
Akkuratheit (27025 Zeichen)	77.1 %	88.4 %	89.3 %	89.2 %
Worterkennungsrate (4134 Wörter)	39.2 %	70.7 %	73.5 %	73.4 %

Tabelle 6.1: Erkennungsergebnisse - verschiedene Kontexttiefen, schreiberunabhängig, Lexikon-frei

Kapitel 7

Formelerkennung

Neben der reinen Texterkennung eröffnet die Erkennung handgeschriebener Formeln eine Reihe weiterer interessanter Anwendungen und Fragestellungen. Führt man sich vor Augen, wie unzureichend bis heute die Eingabemöglichkeiten an einem Computer für mathematische Formeln realisiert sind, wird der Bedarf nach dieser Form von Mensch-Maschine-Schnittstellen schnell deutlich.
Für einen handschriftlichen Formeleditor sind im Desktop-Bereich Textverarbeitungssysteme das wichtigste Anwendungsszenario. Hier sind derzeit im wesentlichen zwei konkurrierende Ansätze gebräuchlich. Der erste ist die klassische Menü-gesteuerte Variante, wie sie bei WYSIWYG- (What You See Is What You Get) Textverarbeitungssystemen eingesetzt wird. Hierbei muß der Benutzer die einzelnen Symbole mittels Tastatur eingeben oder (z.B. bei Sonderzeichen oder Operatoren) über Auswahlmenüs selektieren. Der zweite Ansatz geht über die Verwendung von Satzsystemen wie z. B. LaTeX, bei dem die verschiedenen Symbole durch Befehle in einer Quelltextdatei gesetzt werden. LaTeX bietet somit die Möglichkeit auch ausgefallene Ausdrücke darzustellen, erinnert dafür aber stark an eine höhere Programmiersprache und zwingt den Benutzer sich eine Vielzahl verschiedener Befehle zu merken.
Ein weiteres Anwendungsszenario ergibt sich im Desktop-Bereich bei Mathematik-Tools und Gleichungslösern. Die Problematik der Mensch-Maschine-Schnittstelle ist hier ähnlich. In beiden Fällen ließe sich eine deutliche Verbesserung des Bedienkomforts durch eine handschriftliche Formeleingabe erzielen.
Ebenso ließen sich bei Mobilcomputern der nächsten Generation (Palm-Tops, PDAs und PIMs) in eleganter Weise über die HSE-Schnittstelle komplexe wissenschaftliche Taschenrechnerfunktionen steuern. Der zu berechnende Ausdruck könnte so direkt auf das Display geschrieben werden. Nach der Erkennung der handschriftlichen Eingabe und der Ergebnisberechnung kann das Ergebnis dann in gewohnter Form ausgegeben werden.
Um die spezielle Darstellung mathematischer Ausdrücke und die zweidimensionale Anordnung zu erkennen und zu interpretieren, wird das Gesamtproblem in zwei Teilprobleme zer-

legt. Diese Teilprobleme lassen sich wiederum in insgesamt sechs Prozesse aufteilen, die in dieser oder ähnlicher Form in den meisten Ansätzen wiedergefunden werden [Blo97, Blo95]:

1. Erkennung einzelner Symbole
   (a) Vorverarbeitung, Rauschfilter, Skew-Reduktion
   (b) Segmentierung
   (c) Symbolerkennung
2. Analyse und Interpretation der Symbolanordnung
   (a) Identifikation räumlicher Symbolbeziehungen
   (b) Identifikation logischer Symbolbeziehungen
   (c) Konstruktion der Bedeutung (z. B. in Syntaxform oder Baumstruktur)

Ein wesentliches Problem ist hier die Aufteilung der Segmentierung und der Symbolerkennung in zwei getrennte Prozesse. Speziell für die Handschrifterkennung gilt, dass eine korrekte Segmentierung häufig nur mit Kenntnis der Klassenzugehörigkeit möglich ist. Ebenso wie eine unvollständige oder falsche Segmentierung zwangsläufig zu einer falschen Symbolerkennung führt. Dieses Henne-Ei-Problem läßt sich effizient durch die Verwendung von HMMs umgehen, da mit der Viterbi-Dekodierung neben dem eigentlichen Erkennungsergebnis auch immer die Segmentierung bekannt ist.
Die folgenden Abschnitte stellen einen Ansatz dar, mit dem die Erkennung handschriftlicher Formeln realisiert werden kann. Nach einer generellen Übersicht im folgenden Abschnitt, werden in den weiteren Abschnitten die einzelnen Verarbeitungsschritte, insbesondere die Strukturanalyse und das Parsing beschrieben. Im Abschnitt 7.6 werden über die eigentliche Erkennung hinaus einige integrierte handschriftliche Korrekturfunktionen beschrieben, bevor in den letzten beiden Abschnitten die Ergebnisse dargestellt werden und die Zusammenfassung dieses Kapitel abschließt.

7.1 Grundlegende Funktionsweise

Das hier beschriebene System ist in einem schreiberabhängigen Modus in der Lage, zusammenhängende mathematische Ausdrücke bestehend aus einem Zeichenvorrat von ca. 100 verschiedenen Zeichen zu erkennen. Neben Ziffern, Klein- und Großbuchstaben können die wichtigsten mathematischen Symbole und Operatoren erkannt werden ($+ - \cdot : / \text{——} \hat{} \sqrt{} \sum \prod \int , ' = < > \leq \geq \rightarrow \leftrightarrow \approx ! \infty$), wie auch eine Reihe weiterer griechischer Buchstaben ($\alpha, \beta, \gamma, \lambda, \mu, \Delta, \pi, \omega, \epsilon, \tau, \phi$). Zur Modellierung der Symbol-Zwischenräume

wird - ähnlich wie bei einer Satzerkennung - ein *Space*-Modell eingesetzt. Auch bei der Formelerkennung findet die Initialisierung anhand von vorsegmentierten Trainingsbeispielen statt, wie sie beispielhaft in Abb. B.1 dargestellt sind. Das anschließende Training der Modelle, sowie die Evaluierung wird dann anhand ganzer zusammenhängender Ausdrücke, wie sie in Abschnitt B.3 beispielhaft dargestellt sind, durchgeführt.
Um zum einen eine möglichst konsistente Parameterschätzung des Space-Modells zu erzielen, zum anderen aber auch um den Aufwand für das zweidimensionale Parsing des Erkennungsergebnisses zu begrenzen, werden insgesamt vier Benutzungsregeln definiert, die bei der handschriftlichen Eingabe der Formeln beachtet werden sollen. Diese Benutzungsregeln - dargestellt in einer graphischen Übersicht in Abb. 7.1 - sind jedoch so formuliert, dass sie einer möglichst natürlichen Schreibweise entgegenkommen [Bec97, Kos98b] und die Benutzbarkeit des Systems damit nicht nennenswert einschränken:

1. *left-right, top-down*

   Ganz allgemein formuliert müssen die handschriftlichen Eingaben von links nach rechts und von oben nach unten erfolgen. Am Beispiel eines Bruchs bedeutet diese Regel, dass zunächst der Zähler zu schreiben ist, anschließend der Bruchstrich gezogen werden muß, bevor schließlich der Nenner geschrieben werden kann. Klammerausdrücke sind entsprechend dieser Regel in folgender Form einzugeben: *linke Klammer, Ausdruck, rechte Klammer*.

2. *Integrale, Summen, Produkte*

   Aus Beobachtungen ergab sich, dass es in einigen Fällen günstig ist, eine Ausnahme von Regel 1 zu erlauben. Dies ist beispielsweise der Fall bei Integralen, Summen oder Produkten. Tritt im Zusammenhang mit diesen Symbolen eine Untergrenze und/oder Obergrenze auf, so neigt der Benutzer dazu, nach dem Schreiben des Hauptsymbols zuerst die Untergrenze und erst dann die Obergrenze zu schreiben. Ganz ähnlich verhält es sich bei tiefgestellten Zeichen (Subskripte) in Kombination mit hochgestellten Zeichen (Superskripte), einfachen Anführungsstrichen oder Vektor- bzw. Matrixpfeilen. Auch hier wird es scheinbar als effizienter empfunden nach der Eingabe

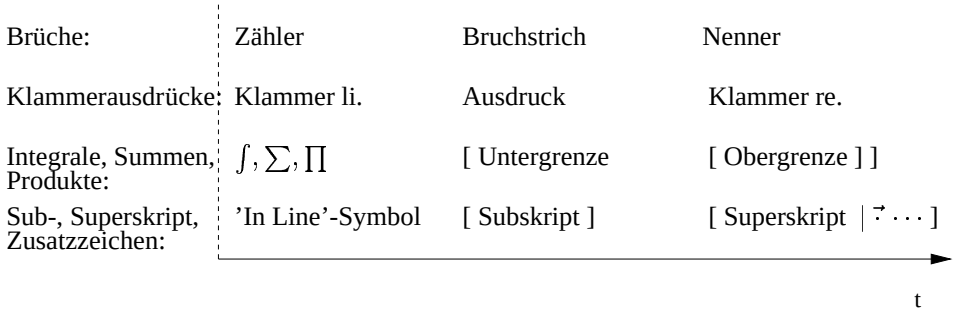


Abbildung 7.1: Benutzungsregeln zur zeitlichen Abfolge handschriftlicher Eingaben

des 'in-line'-Symbols das Subskript vor dem Superskript zu schreiben. Konkret bedeutet diese Ausnahmeregelung, dass z. B. ein Integral mit entsprechenden Grenzen in der Reihenfolge $< Integral,\ untere\,Grenze,\ obere\,Grenze >$ zu schreiben ist.

3. *Verlängerung von Brüchen und Wurzelzeichen*
   In einigen Fällen kann es sinnvoll sein, Zeichen wie Bruchstriche oder Wurzelzeichen zu verlängern, nachdem der Nenner bzw. der Radikand geschrieben wurde und dieser länger ausgefallen ist als ursprünglich vermutet. Aufgrund der Verarbeitung dynamischer Eingangsdaten ist eine Modifikation in der Form nicht möglich, da auf den unteren Systemebenen, d. h. während der Symbolerkennung, die räumlichen Beziehungen noch nicht berücksichtigt werden. Zeitliche Beziehungen sind jedoch bei dieser Form von Korrekturen im Allgemeinen nicht gegeben, sodass Symbol und Modifikation unter Umständen mit großem zeitlichen Abstand auftreten.

4. *Subskript, Superskript*
   Die Erkennung von geschachtelten Subskripten oder Exponenten wie z. B. x^{y^z} sind in diesem System derzeit nicht berücksichtigt.

Unter Beachtung dieser vier Benutzungsregeln ist es nun möglich, den zu erkennenden Ausdruck in kontinuierlicher Form einzugeben. Kontinuierlich bedeutet in diesem Zusammenhang, dass die einzelnen Symbole weder spatial noch temporal separiert geschrieben werden müssen. Vielmehr läßt sich der hier verfolgte Ansatz wie eine Satzerkennung ohne vorhergehende Wortsegmentierung betrachten, wobei die Identifikation von Wort- bzw. Symbolgrenzen einen integralen Bestandteil des HMM-Ansatzes unter Verwendung des 'space'-Modells darstellt. Ein weiterer Vorteil der Verwendung von HMMs ist die Möglichkeit, bei Bedarf gewisse syntaktische Randbedingungen in den Dekodierprozeß direkt zu integrieren. Abb. 7.2 gibt eine Übersicht über das grundlegende System zur Formelerkennung, wie es in [Kos98c, Kos99b] vorgestellt wurde. Die Funktionsweise der einzelnen Verarbeitungsstufen ist in den nachfolgenden Abschnitten dargestellt.

7.2 Vorverarbeitung, Merkmalsextraktion und Dekodierung

Wie bereits in Abschnitt 2.1 beschrieben wurde, erfolgt auch für die Formelerkennung eine räumliche Neuabtastung der Stifttrajektorie. Auch hier gilt es durch die räumlich äquidistante Abtastung die Geschwindigkeitsinformation zu eliminieren und die virtuellen Sequenzen zwischen den Strokes linear zu interpolieren. Dabei wird die Sequenz der *zeitlich äquidistant* abgetasteten Kartesischen Koordinaten $(x, y)(n \cdot \Delta T)$ in die *räumlich äquidistante* Abtast-

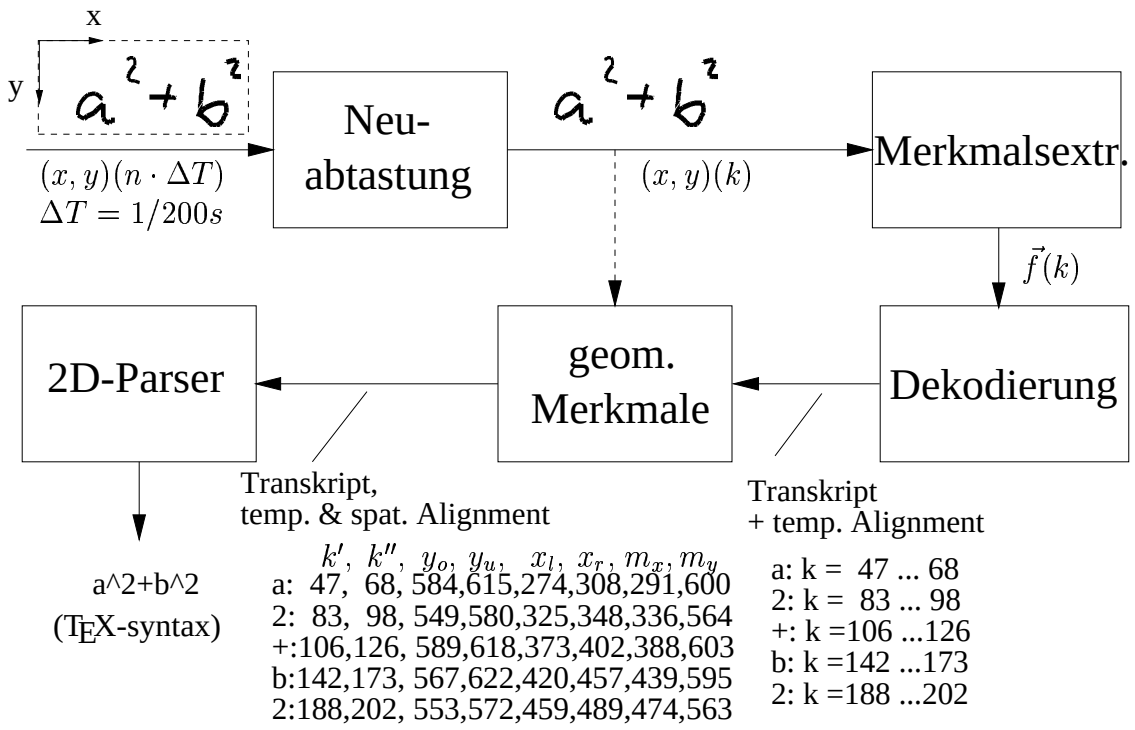


Abbildung 7.2: Systemübersicht des Formeleditors

squenz $(x, y)(k)$ überführt.

Wie auch bei dem Standardsystem zur Wort- bzw. Satzerkennung, werden bei der Formelerkennung Trajektorien- und Bitmap-Merkmale verarbeitet (vergl. Abschnitte 3.1.1 und 3.2.1), sowie zusätzlich die Information über den binären Stiftdruck.

Nach einer Vektorquantisierung, die i. d. R. mittels eines MMI-Netzes - ausgeführt in Multi-Codebuch-Technik - realisiert wird, werden die diskreten, multiplen Merkmalsströme mit Hidden Markov Modellen modelliert bzw. zu Erkennungszwecken dekodiert. Da die Länge der Beobachtungssequenz aufgrund der verschieden großen Symbole bei der Formelerkennung sehr stark schwankt, werden HMMs mit variierender Zustandsanzahl eingesetzt. Zur Modellierung von Großbuchstaben und größeren mathematischen Operatoren und Symbolen (Summe, Integral, Produkt) wurden HMMs mit 12 Zuständen eingesetzt. Kleinbuchstaben und Symbole mittlerer Größe (z. B. $+,\ (,\ ),\ \rightarrow$) wurden mit HMMs bestehend aus acht Zuständen modelliert. Sehr kleine Symbole hingegen, werden schließlich durch HMMs mit drei Zuständen repräsentiert. Für alle genannten Gruppen wurde eine lineare HMM-Topologie verwendet, d. h. es existieren von jedem Zustand nur Selbsttransitionen und Übergänge zum Nachfolgezustand.

Die initialisierten und anschließend trainierten Modelle λ werden schließlich bei der Erkennung verwendet, um eine Viterbi-Approximation in Kombination mit einer Strahlsuche, auf eine unbekannte Beobachtungssequenz Y anzuwenden. Diese Viterbi-Dekodierung liefert nun - im Sinne der Viterbi-Approximation - die wahrscheinlichste Zustandssequenz S^* einer

Menge von HMMs.

$$P(Y, S^*|\lambda) = \max_S P(Y, S|\lambda) \tag{7.1}$$

Die Aussage über die wahrscheinlichste Zustandssequenz S^*, welche die Dekodierung liefert, ist von entscheidender Bedeutung für die Realisierung des Gesamtansatzes. Zum einen ist durch diese Zustandssequenz festzulegen, um welche Symbole und deren Reihenfolge es sich im einzelnen bei der Beobachtungssequenz handelt. Zum anderen erfolgt für die Berechnung der Wahrscheinlichkeiten auch eine Zuordnung der einzelnen 'Frames' zu den jeweiligen Zuständen. Mit anderen Worten heißt das, dass zu *jedem* Frame der Beobachtungssequenz Y bekannt ist, zu welchem Abschnitt dieser Frame der Symbolfolge zuzuordnen ist. Im Rückschluß läßt sich somit die Folge der erkannten Symbole angeben mit jeweils einer exakten Indizierung k von Start- und Endframe.

7.3 Strukturanalyse

Wie bereits angemerkt wurde, liefert die Viterbi-Dekodierung neben dem eigentlichen Erkennungsergebnis auch die Information zu den einzelnen Segmentgrenzen innerhalb der Gesamtsequenz. Die Information zu den Segmentgrenzen (gegeben in zeitdiskreten Abtastschritten k) läßt sich zusammen mit den neuabgetasteten Vektordaten auswerten, um die räumliche Position der einzelnen Segmente zu bestimmen. Die Kenntnis über das aktuelle Symbol, sowie dessen räumliche Zuordnung wird anschließend benötigt um die erkannte Symbolsequenz zu parsen. Anhand des Beispiels in Abb. 7.3 soll das Vorgehen zur Bestimmung der geometrischen Merkmale, d. h., der räumlichen Position erklärt werden.
Dazu wird für das erkannte Symbol 'a' aus der gesamten Sequenz neuabgetasteter Vektoren zunächst der Bereich betrachtet, der ausgehend von den Segmentgrenzen das entsprechende Symbol enthält. In dem Beispiel lieferte die Viterbi-Dekodierung die Segmentgrenzen $k' = 35$ und $k'' = 59$. Beginnend bei $k = 35$, wird das Segment für jeden diskreten Zeitschritt k auf Minima und Maxima in x- und y- Richtung geprüft. Indem diese Extremwerte für jedes Segment gespeichert werden, erhält man am Segmentende eine exakte Information über das umschreibende Rechteck zu dem betrachteten Segment. Die Koordinaten und Abmessungen dieses umschreibenden Rechtecks werden für das anschließende Parsing als geometrische Merkmale weiterverarbeitet. Zuzüglich zu den Kartesischen Koordinaten der Kanten des umschreibenden Rechtecks wird der Mittelpunkt des Segments bestimmt, der als geometrischer Mittelpunkt des umschreibenden Rechtecks approximiert wird ($m_x = (x_l - x_r)/2$ und $m_y = (y_u - y_o)/2$). Mit dieser räumlichen Zuordnung der zeitdiskreten Abtastfolge sind nun Aussagen über Position und Größe der erkannten Symbole verfügbar, die während des Parsings ausgewertet werden.
Die Sequenz der erkannten Zeichen wird zusammen mit den einzelnen zeitlichen Segment-

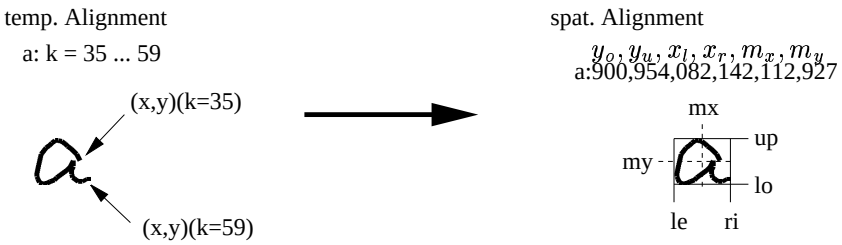


Abbildung 7.3: Bestimmung der räumlichen Segmentierung aus zeitlicher Segmentierung

grenzen und den nun vorhandenen geometrischen Merkmalen zur weiteren Verarbeitung in Form einer doppelt verketteten Liste zusammengefasst. In dem Beispiel in Abb. 7.2 entspricht ein Listenelement demnach einer Zeile der Ausgabe der geometrischen Merkmalsextraktion. Die verkettete Listenstruktur bietet sich deshalb an, da während des nachfolgenden Parsings oder auch um manuelle Korrekturbefehle umzusetzen Elemente an entsprechender Stelle aus der Liste entfernt bzw. hinzugefügt werden müssen. Des weiteren ist so ein einfacher Zugriff auf Vorgänger- oder Nachfolgezeichen möglich.

7.4 Parsing – LRTD-Ansatz

Basierend auf einer erkannten Symbolsequenz und deren geometrischen Merkmalen erfolgt die schrittweise Umsetzung des Erkennungsergebnisses in eine syntaxbeschreibende Form. Der in diesem Abschnitt beschriebene LRTD- (Left-Right-Top-Down-) Ansatz basiert auf den im Abschnitt 7.1 beschriebenen grundlegenden Benutzungsregeln. Als Ausgabeformat wurde hierbei LaTeXgewählt, da das LaTeX-Format zum einen äußerst vielfältige Möglichkeiten der Syntaxbeschreibung bietet und zum anderen die Ergebnisvisualisierung mittels DVI- oder Postscript-Konvertierung in einfacher und übersichtlicher Form ermöglicht. Darüber hinaus ließen sich die Erkennungsergebnisse in dieser Form leicht als Dokumentbausteine im Hinblick auf ein zu realisierendes Gesamtsystem weiterverwenden.
Im wesentlichen setzt sich der LRTD-Parser aus den folgenden sequentiell abzuarbeitenden Modulen zusammen:

- Fehlerkorrektur,
- Parsing spezieller mathematischer Operatoren und
- Detektion und Umsetzung von Sub- und Superskript.

Die Funktionsweise dieser Module wird in den jeweils folgenden Abschnitten beschrieben.

7.4.1 Fehlerkorrektur

Bei komplexen Mustererkennungsaufgaben, wie der Handschrifterkennung, ist eine perfekte, d. h. stets fehlerfreie Erkennung mit heutigen bekannten Ansätzen nicht realisierbar. Dies gilt insbesondere für solche Fälle, bei denen Ambiguitäten auf den unteren Systemebenen nicht aufgelöst werden können. Typisch für diese Fälle ist das in Abb. 7.4 gezeigte Beispiel. Die einzelnen Symbole aus dem oberen und unteren Beispiel ähneln sich dabei sehr stark. Zudem ist die Reihenfolge, in der die jeweilige handschriftliche Eingabe erfolgte, identisch. Dennoch tragen die gezeigten Ausdrücke eine völlig unterschiedliche Bedeutung. Verdeutlicht man sich an dieser Stelle einmal den Ablauf der gesamten Erkennung, wie er in Abb. 7.2 dargestellt ist, wird klar, dass die Dekodierung aufgrund fehlender Möglichkeiten zur Auswertung von Kontextinformation solche Grenzbereiche nicht unterscheiden kann. Mit Kontextinformation sind an dieser Stelle die geometrischen Merkmale der einzelnen Symbole gemeint, die erst nach der Dekodierung bestimmt werden.
Die Fehlerkorrektur des Parsers bezieht sich somit auch nur auf offensichtliche Fehler des Dekoders, wie z. B. die gezeigte Vertauschung eines kurzen Bruchstrichs mit einem Minuszeichen im unteren Beispiel von Abb. 7.4.
Eine weitere korrigierbare Fehlerklasse ist die Vertauschung von einfachen Anführungszeichen bei 'gestrichenen' Variablen mit Kommata ($x' \leftrightarrow x,$) oder auch die fälschliche Einfügung eines Punktes nach den Worten sin oder lim. Diese können durch sog. 'delayed strokes' entstehen, indem zunächst die Buchstabenfolge in verbundener Form geschrieben wird, und anschließend der i-Punkt über den Wortkörper gesetzt wird.
Die Korrektur falsch eingefügter i-Punkte läßt sich einfach realisieren, indem im Erkennungsergebnis auftretende Punkte nach bestimmten Schlüsselwörtern, wie den Worten $\sin$ oder $\lim$ pauschal gelöscht werden. Die Löschung kann vorgenommen werden, da eine solche Zeichenfolge keinen regulären Ausdruck formen kann.
Um die Vertauschung von Minuszeichen und kurzem Bruchstrich zu korrigieren (Bsp. in Abb. 7.4), wird jede Transkribierung zunächst auf vorhandene Brüche oder Subtraktionen überprüft. Tritt ein solches kritisches Zeichen auf, so werden die geometrischen Merkmale

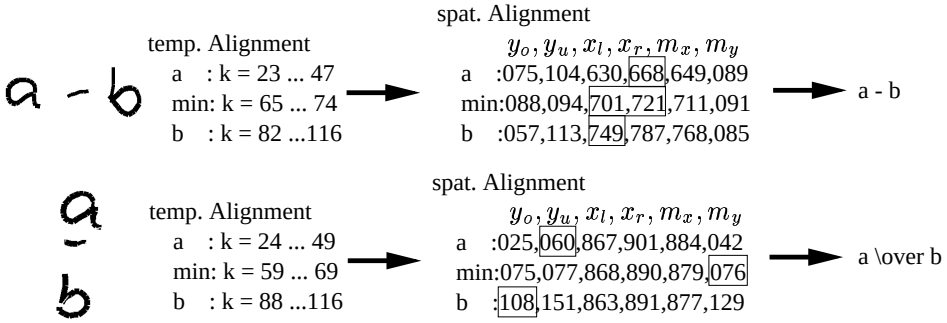
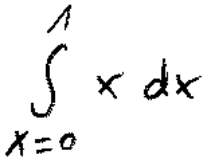


Abbildung 7.4: Beispiel zur Fehlerkorrektur

des Bruchstrichs bzw. des Minuszeichens mit den geometrischen Merkmalen des (zeitlichen) Vorgängersymbols und des (zeitlichen) Nachfolgesymbols verglichen. Der Vergleich bezieht sich dabei auf die in Abb. 7.4 markierten geometrischen Merkmale. Ein Minuszeichen liegt genau dann vor, wenn das Merkmal x_r des Vorgängersymbols (im Bsp. der Buchstabe 'a') kleiner ist als das Merkmal x_l des als 'min' erkannten Symbols. Zusätzlich muß überprüft werden, ob das Merkmal x_r des 'min'-Symbols kleiner als das Merkmal x_l des zeitlichen Nachfolgers ist. Da im oberen Beispiel beide Bedingungen erfüllt sind ($x_r('a') = 668 < x_l('min') = 701$ und $x_r('min') = 721 < x_l('b') = 749$), wird die Erkennung bestätigt. Es muß sich folglich um ein Minuszeichen handeln.
Im zweiten Beispiel in Abb. 7.4 zeigt dieser Vergleich, dass die Bedingungen für ein Minuszeichen nicht erfüllt sind. Vielmehr wird durch den Vergleich der geometrischen Merkmale von (zeitlichem) Vorgänger und Nachfolger mit den geometrischen Merkmalen des 'min'-Symbols klar, dass es sich um einen Bruchstrich handeln muß. Speziell sind bei diesem Vergleich die Merkmale $y_l('a')$, der Mittelpunkt in y-Richtung des 'min'-Symbols $m_y('min')$ und $y_l('b')$ relevant. Ein Bruchstrich liegt dann vor, wenn $y_u('a') < m_y('min)$ und $m_y('min') < y_o('b')$. Die markierten geometrischen Merkmale des unteren Beispiels in Abb. 7.4 bestätigen also, dass es sich um einen Bruchstrich handeln muß, was eine anschließende automatische Korrektur zur Folge hat.
Ein ganz ähnliches Prüfverfahren wird durchgeführt, wenn entweder Kommata oder Hochkommata in der Erkennung auftreten, um Vertauschungen eben dieser Symbole zu korrigieren. Auch hier sind beide Symbole ohne Kontextinformation kaum zu unterscheiden. Der Vergleich bezieht sich in diesem Fall jeweils auf die Merkmale m_y von Komma bzw. Hochkomma und dem Merkmal m_y des Vorgängersymbols. Bei einem Komma muß $m_y(',\ ') < m_y(Vorgänger)$ gelten, bei einem Hochkomma wären die Relationen genau umgekehrt.

7.4.2 Spezielle mathematische Operatoren

In einem weiteren Parsingschritt wird nach einigen speziellen Symbolen gesucht, deren zeitlicher Kontext eine besondere Bedeutung haben könnte. Dies sind insbesondere die Symbole $\int$, $\sum$, $\prod$ und $\lim$. Wird in der Transkribierung ein Summen-, Integral- oder Produktsymbol gefunden, so muß das Ergebnis auf mögliche Unter- oder Obergrenzen hin untersucht werden. Dabei können nun wieder die in Abschnitt 7.1 definierten Benutzungsregeln zur Hilfe genommen werden. Dort wurde mit Bedingung (2) festgelegt, dass eine Untergrenze - beispielsweise eines Integrals - zeitlich direkt nach dem Integral selbst geschrieben wird. Eine mögliche Obergrenze wird zeitlich nach der Untergrenze geschrieben.
Anhand des Beispiels in Abb. 7.5 läßt sich nun das Parsing dieser speziellen Operatoren erklären. Wie auch bei der oben beschriebenen Unterscheidung zwischen kurzen Brüchen



temp. Alignment

int	: k = 21 ... 56
x	: k = 108 ... 132
=	: k = 144 ... 264
0	: k = 172 ... 189
1	: k = 241 ... 260
x	: k = 330 ... 356
d	: k = 373 ... 409
x	: k = 417 ... 444

spat. Alignment

	$y_o, y_u, x_l, x_r, m_x, m_y$
int	:455,559,080,131,105,507
x	:583,618,041,068,054,600
=	:591,613,084,109,096,602
0	:587,609,121,142,131,598
1	:395,435,101,136,118,415
x	:498,532,178,209,193,515
d	:475,532,249,280,264,503
x	:495,534,295,334,315,514

Abbildung 7.5: Detektion von Unter- und Obergrenze am Beispiel eines Integrals

und Minuszeichen wird zunächst jeder Ausdruck auf das Vorhandensein dieser speziellen Operatoren (Integral, Summe etc.) hin untersucht. Tritt - wie im Beispiel gezeigt - ein solcher Operator auf, so wird zunächst die Position der Unterkante des umschreibenden Rechtecks $y_u('int')$ mit dem geometrischen Merkmal y_o des Nachfolgers verglichen (hier: $y_o('x')$). Aus dem spatialen Alignment wird sofort ersichtlich, dass sich das Nachfolgesymbol 'x' unterhalb des Integrals befindet ($y_u('int') = 559 < y_o('x') = 583$). Da eine Untergrenze nicht zwangsläufig aus einem einzigen Symbol bestehen muß, sind zusätzlich noch die weiteren Nachfolgezeichen zu untersuchen. Diese Nachfolgezeichen werden nacheinander solange zu einer Untergrenze zusammengefasst, wie deren Oberkante des umschreibenden Rechtecks unterhalb des Integrals liegt. In dem gezeigten Beispiel sind dies neben dem Zeichen 'x' noch das Gleichheitszeichen und die Null ($y_u('int') = 559 < y_o('=') = 591$ sowie $y_u('int') = 559 < y_o('0') = 587$). Bei dem fünften Zeichen der Transkribierung (hier: die '1') ist diese Bedingung jedoch nicht mehr erfüllt, sodass die Gruppierung zur Untergrenze abgebrochen werden muß. Ungeklärt ist an dieser Stelle allerdings noch, ob für das Integral eine Obergrenze angegeben wurde. Dazu muß zunächst für das nächste Symbol nach der Untergrenze die relative Position zum Integral betrachtet werden. Insbesondere werden dazu die Merkmale $y_o('int')$ und $y_u('1')$ verglichen, wobei sich zeigt, dass $y_u('1') = 435 < y_o('int') = 455$ und damit eine Obergrenze vorliegt. Bei einem positiven Test für eine Obergrenze wird zwecks Gruppierung nachfolgender Symbole in analoger Weise zur Gruppierung der Untergrenze verfahren. Nach Abbruch der Gruppierung zur Obergrenze werden die nachfolgenden Symbole zum Integranden zusammengefasst.

Wird das Wort 'lim' erkannt, so ist zu erwarten, dass nachfolgend die Variable genannt wird, auf die sich die Grenzwertbildung bezieht, gefolgt von einem Pfeil und dem Grenzwert selbst. Die Gruppierung erfolgt auch hier durch Vergleich der Unterkante des Wortes 'lim' mit den Oberkanten der Nachfolgesymbole.

Nach diesem Prinzip erfolgt auch die Gruppierung zu Teilausdrücken bei auftretenden Wurzeln oder Brüchen (Abb. 7.6). Bei diesen Strukturen ist es allerdings durchaus wahrscheinlich, dass ein Teilausdruck, d. h. der Radikand, der Zähler oder der Nenner eines Bruches über das Wurzelzeichen bzw. den Bruchstrich hinausragt. Dies kann insbesondere dann be-

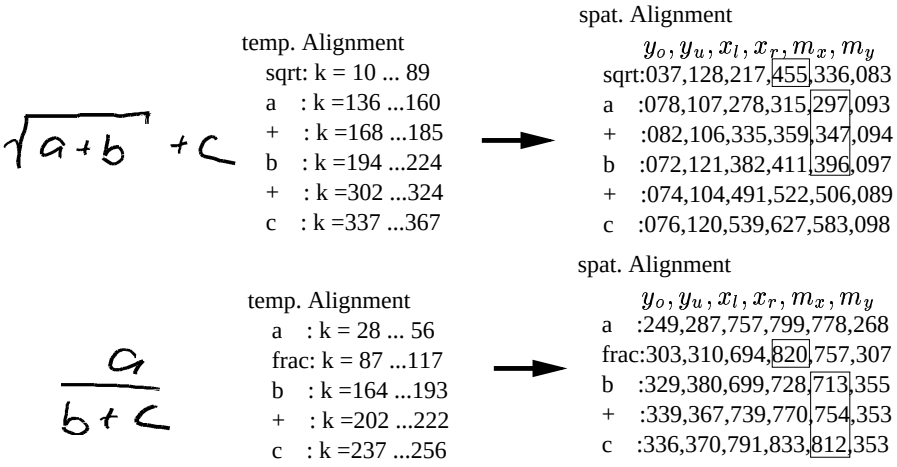


Abbildung 7.6: Gruppierung von Teilausdrücken

obachtet werden, wenn das letzte Zeichen eines Teilausdruckes noch mit Zusatzzeichen oder Indizes versehen wird. Zur Gruppierung von Zeichen zu Teilasudrücken wird daher nicht das umschreibende Rechteck als Entscheidungskriterium herangezogen, sondern der geometrische Mittelpunkt der Zeichen. Wie in Abb. 7.6 am Beispiel einer Wurzel gezeigt wird, werden die nachfolgenden Symbole zu einem Radikanden zusammengefasst, deren Mittelpunkt m_x unterhalb des Wurzelzeichens liegt ($m_x(Nachfolger) < x_r('sqrt')$). Im oberen Beispiel in Abb. 7.6 sind dies die Zeichen 'a', '+' und 'b'. Die Suchrichtung für den Radikanden ist hier wiederum aus der Bedingung (1) in Abschnitt 7.1 bekannt.
Das zweite Beispiel in Abb. 7.6 zeigt eine Situation, in der die Gruppierung, basierend auf den Merkmalen des umschreibenden Rechtecks, fehlschlagen würde, da das 'c' im Nenner etwas über den Bruchstrich hinausragt ($x_r('c') > x_r('-')$. Da jedoch auch bei der Gruppierung von Brüchen von den Mittelpunkten der Zeichen ausgegangen wird, kann der Nenner korrekt zusammengefasst werden ($m_x('c') < x_r('-')$). Die Suchrichtungen für Zähler und Nenner sind ebenfalls aus Bedingung (1) in Abschnitt 7.1 bekannt: einem Bruchstrich muß der Zähler vorausgehen und der Nenner folgen.

7.4.3 Sub- und Superskript

Die zuvor beschriebenen Methoden zur Strukturierung des Erkennungsergebnisses basierten stets auf der Existenz gewisser Schlüsselwörter, wie z. B. 'int' bei Integralen oder 'sqrt' bei Wurzeln. Auf solche Hinweise kann bei der Suche nach Sub- oder Superskripten nicht zurückgegriffen werden. Die Situation stellt sich hier vielmehr so dar, dass bis auf Ausnahmen jedes erkannte Zeichen zu einem Index oder Exponenten zählen könnte. Als ein wesentlicher Anhaltspunkt kann hier lediglich die in Abschnitt 7.1 definierte Bedingung (1) verwendet werden, wonach ein Index oder Exponent in zeitlicher Reihenfolge unmittelbar nach dem 'in-line' Zeichen einzugeben ist.

Um den Suchaufwand nach Indizes oder Exponenten einzuschränken und auch das Potential für mögliche Fehlinterpretationen zu reduzieren, ist es zunächst sinnvoll, mögliche Zeichentypen für Indizes und Exponenten zu spezifizieren. Für Indizes lassen sich die gültigen Ausdrücke so z. B. auf Zahlen, Buchstaben und deren Kombination beschränken. Für das Parsing kann weiterhin ausgenutzt werden, dass die Basis zu einem Exponenten oder das 'in-line' Zeichen eines Index - bis auf Ausnahmen - ebenfalls Variablen oder Zahlenwerte sind. Als Ausnahmen sind hierbei noch die Symbole $), \ ], \ \}, \ \sqrt{}, \ !, \ \infty$ zu berücksichtigen, welche darüber hinaus als 'in-line' Symbol auftreten können. Für den Exponenten gelten prinzipiell jedoch keine Restriktionen, sodass auch komplexe Strukturen, wie z. B. Brüche als Exponenten möglich sind. Die einzige Einschränkung hier ist, dass keine Exponenten höherer Ordnung (d. h. Exponent vom Exponent) zugelassen sind. Diese Einschränkung ist notwendig, da die verschiedenen Ebenen bei handschriftlicher Eingabe kaum mehr zu unterscheiden sind. Ein durchschnittlicher Exponent ist zudem nur zwei bis drei Millimeter groß. Bei einer Auflösung von 300 dpi bedeutet das schließlich, dass solche kleinen Zeichen nach der Neuabtastung lediglich durch 10-20 Abtastpunkte repräsentiert werden. Dies erschwert nicht nur die eigentliche Erkennung der Zeichen und die Unterscheidung zwischen Groß- und Kleinschreibung bei Buchstaben, sondern auch die zuverlässige relative Positionsbestimmung der einzelnen Zeichen und damit die eindeutige Zuordnung zu den verschiedenen Ebenen des Exponenten.

Aufgrund der großen Anzahl potentieller Kandidaten für Indizes oder Exponenten muß die Suche nach Sub- bzw. Superskript demnach für eine relativ große Symbolmenge durchgeführt werden. Die Strukturierung stützt sich dabei lediglich auf die relativen Positionen der einzelnen Symbole. Zu diesem Zweck wird jedes Zeichen, beginnend mit dem ersten Zeichen in der Liste erkannter Symbole, untersucht. Erfüllt ein erkanntes Zeichen in der Liste das logische Kriterium, d. h. es ist vom zulässigen Typ, so werden durch Positionsvergleiche von Symbolmittelpunkt und umschreibendem Rechteck zwischen dem aktuell untersuchten Zeichen und deren Nachfolger entsprechende Gruppierungen zu Indizes oder Exponenten vorgenommen, wenn zudem die geometrischen Kriterien erfüllt werden. Für Indizes (Exponenten) gelten $m_y(Basis) < y_o(Nachfolger)$ ($m_y(Basis) > y_u(Nachfolger)$). Zudem muß der Nachfolger in beiden Fällen rechts von der Basis liegen ($m_x(Basis) < x_l(Nachfolger)$).

Da der geometrische Mittelpunkt eines Zeichens mit dem Mittelpunkt des umschreibenden Rechtecks approximiert wird, liegen die Mittelpunkte von Buchstaben mit Unterlängen (f, g, j, p, q, y) tendenziell etwas tiefer als die übrigen Buchstaben oder Zahlen. Um zu vermeiden, dass nun nachfolgende in-line Zeichen fälschlicherweise als Exponent erkannt werden, wird zu Beginn des Parsings die Position m_y eines jeden Zeichens mit Unterlänge um einen gewissen Bias in y-Richtung nach oben korrigiert.

Matrix- bzw. Vektorpfeile werden von dem Parser als eine spezielle Art von Exponenten

behandelt. Die Strukturierung des Erkennungsergebnisses kann bei dieser Form von Symbolen unter Berücksichtigung geometrischer Merkmale allerdings wieder auf die Existenz von Schlüsselworten zurückgreifen.
Nachdem schließlich alle Teilausdrücke gruppiert wurden, erfolgt die abschließende Umsetzung in das LaTeXFormat [Kop94]. Dazu ist es lediglich notwendig die gefundenen Teilausdrücke jeweils mit logischen Klammern zu versehen und diese zusammen mit den übrigen erkannten Zeichen und Operatoren entsprechend der LaTeX-Syntax in eine Ausgabedatei zu schreiben. Funktionen werden im Gegensatz zu Variablen mit einem '\' gekennzeichnet. Die in Abb. 7.6 gezeigten Beispiele resultieren somit nach abgeschlossener Erkennung in zwei reinen (ACSII-) Textdateien mit einer sehr kompakten Beschreibung der Syntax: `\sqrt{a+b}+c` bzw. `a \over{b+c}`.

7.5 2D-Parsing mit kontextuellen Graph-Grammatiken

Neben dem in den vorangegangenen Abschnitten beschriebenen LRTD-Ansatz, wurde in Zusammenarbeit mit dem *Insitut National de Recherche en Informatique et en Automatique* (INRIA) ein weiterer Ansatz realisiert [Kos99a], der auf Benutzungsregeln, wie sie zunächst für den LRTD-Ansatz formuliert wurden weitgehend verzichtet. Dieser Ansatz basiert im wesentlichen auf der Kombination zweier existierender Systeme für die Formelerkennung, bei deren unabhängiger Entwicklung jedoch zunächst unterschiedliche Ziele verfolgt wurden.
Bei dem einen System handelt es sich um den bereits beschriebenen handschriftlichen Formeleditor. Das zweite System ist das am INRIA entwickelte *OFR* (Optical Formula Recognition), welches für die Offline-Erkennung von Formeln in gedruckten Dokumenten entwickelt wurde. Dieses System besteht aus den drei weitgehend unabhängigen Hauptkomponenten OCR, Graph-Builder und Graph-Parser [Lav97, Lav98].
Das OCR-Modul liefert dabei die Eingabe für den Graph-Builder in Form von erkannten Einzelzeichen, und deren Position und Größe. Da es sich um eine Offline-Erkennung handelt, existiert eine *zeitliche* Zeichenfolge im Sinne der erfolgten Zeicheneingabe nicht. Der auf der OCR-Ausgabe aufsetzende Graph-Builder und der nachfolgende Graph-Parser werten lediglich Position und Größe der erkannten Zeichen für die Strukturierung aus. An dieser Stelle wird bereits klar, dass sich die Vorteile beider Systeme elegant miteinander verknüpfen lassen, wodurch sich eine handschriftliche Formelerkennung ohne vorgegebene Schreibrichtung realisieren läßt. Als Schnittstelle zwischen beiden Systemen dient hier lediglich die Transkribierung der handschriftlichen Eingabe und deren geometrische Merkmale.
Aus der Transkribierung wird zunächst zusammen mit den geometrischen Merkmalen

ein Graph konstruiert, der die geometrischen Zusammenhänge zwischen den einzelnen initialen lexikalischen Einheiten repräsentiert. Die gerichteten Kanten, welche die Knoten miteinander verbinden, tragen somit die Information über die relativen Positionen der Knoten. Der Graph wird in der Form konstruiert, dass zunächst versucht wird, jedes Symbol der Formel in acht Richtungen (rechts, links, oben, unten, oben rechts, oben links, unten links, unten rechts) mit seinen nächsten Nachbarn zu verbinden, wobei - in Abhängigkeit des untersuchten Zeichens - bestimmte Gültigkeitstests bezüglich der Richtung potentieller ein- und ausgehender Kanten durchgeführt werden. Darüber hinaus wird bei diesem Test ein elliptisches Potenzialmodell verwendet, welches der bevorzugten Schreibrichtung (von links nach rechts) entspricht. Damit läßt sich modellieren, dass in horizontaler Richtung ein rechts liegendes Zeichen beispielsweise über weitere Abstände mit einer ausgehenden Kante verbunden werden kann, als ein links liegendes Zeichen (vergl. [Lav98]).
Nachdem auf diese Weise ein initialer Graph der Formel konstruiert wurde, kann damit begonnen werden diese zweidimensionale Beschreibung der Formel zu parsen. Dabei wird eine kontextabhängige Graph-Grammatik [Pfa69] verwendet, die bestimmte Syntaxregeln in Abhängigkeit der verwendeten Symbole auswertet. In iterativer Weise werden aus dem eingangs generierten Graphen Teilgraphen zu einem Knoten zusammengefasst. Der resultierende Knoten enthält eine abstrahierte Prefix-Syntaxbeschreibung des Teilausdrucks. Die geometrischen Merkmale dieses Knotens werden nach der Zusammenfassung auf die Parameter des umschreibenden Rechtecks des gesamten Teilausdrucks gesetzt. Bei diesem Ansatz, der in der Literatur auch als *Graph-rewriting* [Blo96] bekannt ist, wird diese Zusammenfassung von Teilausdrücken iterativ wiederholt, bis schließlich ein verbleibender Knoten die Syntaxbeschreibung der gesamten Formel enthält.

7.6 Manuelle Korrekturfunktionen

Die bisher vorgestellten Ansätze zur Realisierung eines handschriftlichen Formeleditors basierten stets auf dem Prinzip eine ganze Formel vollständig einzugeben und anschließend die Erkennung zu starten. Nachträgliche Modifikationen oder auch partielle Korrekturen von möglichen Erkennungsfehlern waren dabei ausgeschlossen. Das bedeutet, dass ein Benutzer ohne die Unterstützung manueller Korrekturfunktionen gezwungen ist, selbst bei kleinsten Erkennungsfehlern eine u. U. sehr komplexe Formel komplett neu einzugeben. Der erhöhte Gebrauchswert bei Einführung manueller Korrekturfunktionen zeigt sich jedoch nicht nur bei auftretenden Erkennungsfehlern. Gerade bei einem handschriftlichen Formeleditor kann es zudem ausgesprochen nützlich sein, nach der Erkennung der Eingabe noch Erweiterungen oder Veränderungen vorzunehmen, um die gewünschte Formel auf dem ’elektronischen Pa-

pier' schrittweise zu entwickeln. Zu diesem Zweck wurden die Editierfunktionen 'Löschen', 'Ersetzen', 'Einfügen', 'Rückgängig und Wiederholen' und 'Neuzeichnen' in den Demonstrator integriert.
Im Prinzip ist die Umsetzung von Editierfunktionen bei einem handschriftlichen Formeleditor auf zwei Arten denkbar:

- Zum einen könnten Modifikationen auf dem Erkennungsergebnis selbst erfolgen. Bei der hier gewählten Ausgabeform wäre das sehr leicht umzusetzen, da es sich bei der Ausgabe um reine Textdateien handelt.
- Zum anderen ist es aber auch denkbar die Änderungen auf dem temporär gespeicherten Schriftbild vorzunehmen, anschließend die handschriftlichen Editierbefehle und Korrekturzeichen erkennen zu lassen, die Befehle auszuführen und die Änderungen dann schließlich mit der zuvor erzeugten Ausgabe zusammenzuführen.

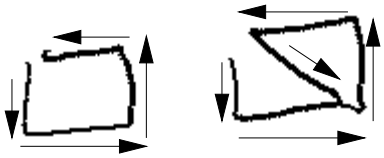
Während erste Arbeiten zur Integration von Erkennung und automatischer Ausführung handschriftlicher Korrekturen in OCR-Systemen bekannt sind [Mor97] blieb der zweite o. g. Ansatz bis auf die an der Gerhard-Mercator-Universität - Duisburg durchgeführten Untersuchungen [Kos00, Mes99] völlig unberücksichtigt. Doch gerade dieser Ansatz bietet eine Reihe von interessanten Fragestellungen und Vorteilen bezüglich der Benutzbarkeit. Änderungen bzw. Korrekturen können so vom Benutzer durchgeführt werden ohne einen lästigen Wechsel der Eingabemodalitäten vorzunehmen (vom Stift zur Tastatur oder Maus). Bei Endgeräten, die konsequent auf die Stifteingabe setzen, ließen sich mit einem solchen Verfahren in konsistenter Form z. B. Taschenrechnerfunktionen integrieren.

7.6.1 Funktionsweise manueller Korrekturfunktionen

Für die Erkennung und Ausführung manueller Korrekturfunktionen ist es zunächst notwendig Steuersymbole zu definieren. Diese Symbole werden wie die übrigen Zeichen, Buchstaben und Ziffern anhand von Trainingsbeispielen gelernt und durch ein oder mehrere HMMs repräsentiert. Wird anschließend ein solches Steuersymbol erkannt, können daraufhin entsprechende Editieroperationen auf dem Original vorgenommen werden.
Ein solches Steuersymbol sollte daher gewisse Eigenschaften aufweisen. Zunächst ist es wichtig, mit einem Steuersymbol die relevanten Bereiche, auf die sich die Änderungen beziehen, exakt markieren zu können. Eine weitere wichtige Eigenschaft des Steuersymbols sollte die gute Unterscheidbarkeit zu sämtlichen übrigen Zeichen sein, da Vertauschungen zwischen regulären Zeichen und Steuersymbolen zwangsweise einen starken unerwünschten Einfluß auf das Gesamtergebnis zur Folge haben. Erste Tests mit unterschiedlichen Editiersymbolen [Kos00] zeigten, dass die in Abb. 7.7 dargestellten Zeichen die erforderlichen

Eigenschaften aufweisen. Da bei der Online-Handschrifterkennung auch die zeitliche Abfolge der ’Strokes’ relevant ist, kann dies als zusätzliches Unterscheidungsmerkmal einbezogen werden, indem die in Abb. 7.7 mit den Pfeilen angedeutete Schreibrichtung eingehalten wird. Mit entsprechenden Variationen dieser Symbole hinsichtlich Ausdehnung und Kantenverhältnis in der Trainingsdatenbasis sind natürlich auch verschiedene Realisierungsformen bezüglich der Form und Größe in der Erkennungsphase möglich.

Wenngleich beide Varianten in Abb. 7.7 von dem Demonstrator unterstützt werden, ist den-



(1) (2)

Abbildung 7.7: Implementierte Editierbefehle

noch Variante (2) zu bevorzugen, da dieses Symbol sehr sicher erkannt wird. Wie in den nachfolgenden Abschnitten noch im Detail gezeigt wird, können diese Editiersymbole universell für die Funktionen ’Löschen’, ’Ersetzen’ und ’Einfügen’ verwendet werden.

Für die Funktion ’Löschen’ sowie für die übrigen Editierfunktionen gilt, dass diese sich ohne Änderungen an der gewählten Gesamtarchitektur des Demonstrators integrieren lassen. Die Änderungen beschränken sich lediglich auf die Einführung neuer Modelle für die Steuersymbole in den Dekoder und einen erweiterten Parser, der bei erkannten Steuersymbolen entsprechende Änderungen auf vorherigen Versionen des Dokumentes vornimmt. Dazu ist es natürlich erforderlich, dass der Parser über eine entsprechende Verwaltung der Historie verfügt, sowie über eine Synchronisation der verschiedenen Versionen von handschriftlichen Eingabedokumenten, Transkribierungen mit geometrischen Merkmalen und Ausgabedokumenten. Die Historie ist als Stapel mit vorzugebender Maximalgröße ausgeführt. Ein Stapelelement wiederum entspricht jeweils einer Eingabe mit Erkennungsergebnis und enthält demnach einen Verweis auf die handschriftlichen Eingabedaten, sowie die verkettete Liste mit den Transkribierungen, Segmentgrenzen und geometrischen Merkmalen.

Der Vorteil der Listenstruktur kommt auch bei der Ausführung manueller Korrekturbefehle zum tragen, da hier teilweise ganze Zeichensequenzen zu entfernen oder durch neue Sequenzen zu ersetzen sind.

7.6.2 Löschen

Um bestimmte Teile einer Formel zu löschen, ist der betreffende Bereich mit dem Steuersymbol zu markieren. Nach erneuter Erkennung wird der entsprechende Bereich aus dem Ausgabedokument entfernt. Abb. 7.8 verdeutlicht dies an einem Beispiel. Der erste Schritt

zeigt das Original der Eingabe, wie es auf dem *Canvas* des Systems (s. auch Abb. 8.1) angezeigt wird. Eine erstmalige Erkennung generiert ein entsprechendes Ausgabedokument (`mx+b=y`). Im zweiten Schritt wird dann der zu löschende Bereich markiert und die Erkennung erneut gestartet. Das Ergebnis ist eine Ausgabedatei mit dem Inhalt `mx=y`.

Während im Canvas sämtliche Stiftaktionen visualisiert werden, schickt die Benutzerober-

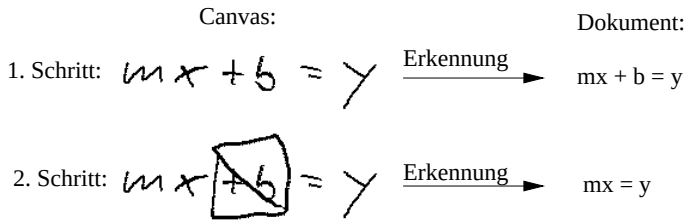


Abbildung 7.8: Löschen von Teilausdrücken

fläche immer nur die Daten in den Erkennungsprozeß, die seit dem *letzten* Erkenneraufruf hinzugekommen sind. In dem gezeigten Beispiel bedeutet dies also, dass der Dekoder lediglich das (hinzugefügte) Steuersymbol zu sehen bekommt. Die einzige Ausgabe des Dekoders ist somit die Transkribierung des Steuersymbols 'ctrl' und das temporale Alignment der Abtastvektoren. Daraufhin werden wiederum geometrische Merkmale (umschreibendes Rechteck und Mittelpunkt) des Steuersymbols bestimmt und dies zusammen mit der Transkribierung 'ctrl' an den Parser weitergeleitet.

Bei dem nun aufgetretenen Steuerbefehl wird von dem Parser zunächst das letzte, d. h. oberste Stapelelement (Verweis auf handschriftliche Eingabe und verkettete Liste) vom Stapel geholt, auf welches die Operation anzuwenden ist. Zusammen mit den Parametern des umschreibenden Rechtecks des Steuersymbols können die zu löschenden Symbole nun bestimmt werden, indem die Liste erkannter Zeichen nach solchen Einträgen durchsucht wird, deren Mittelpunkt (m_x, m_y) innerhalb des umschreibenden Rechtecks des Steuersymbols liegt. Die auf diese Weise bestimmten Zeichen werden anschließend aus der Liste gelöscht.

Um größere Lücken zwischen den Formelelementen zu vermeiden, werden die geometrischen Merkmale der nachfolgenden verbleibenden Zeichen um einen gewissen Offset verschoben. Dieser Offset wird bestimmt, indem ein umschreibendes Rechteck für die gesamte Sequenz zu löschender Zeichen berechnet wird. Das umschreibende Rechteck muß dabei jedoch nicht mehr aus den Vektordaten ermittelt werden, sondern kann durch Maximum- bzw. Minimumbildung über alle enthaltenen umschreibenden Rechtecke bestimmt werden. Die Verschiebung der Zeichen betrifft an dieser Stelle zunächst die Korrektur der geometrischen Merkmale um den ermittelten Offset. Für eine spätere Angleichung des Schriftbildes (Funktion 'Neuzeichnen') wird der Offset für das jeweilige verschobene Zeichen in einer zusätzlichen Variablen innerhalb des Listenelementes gespeichert.

Abschließend erfolgt die Ausgabe des aktualisierten Dokumentes und die Ablage der aktua-

lisierten Liste als neues Stapelelement.
Auf diese Weise ist es natürlich auch möglich mehrere Korrekturzeichen in einem Arbeitsschritt zu verarbeiten (Abb. 7.9), indem die Suche nach von Steuersymbolen überdeckten Zeichen für jedes auftretende Steuersymbol durchgeführt wird.

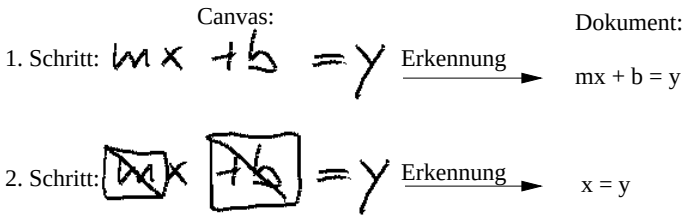


Abbildung 7.9: Löschen mehrerer Teilausdrücke

7.6.3 Ersetzen

Die Funktion ’Ersetzen’ basiert wie die Funktion ’Löschen’ ebenfalls auf dem Prinzip, die von der Modifikation betroffenen Zeichen mittels eines handgezeichneten Steuersymbols zu selektieren (Abb. 7.10). Beachtlich ist, dass trotz der deutlich unterschiedlichen Größen der in Abb. 7.10 und Abb. 7.8 gezeigten Steuersymbole, beide Realisierungen durch nur ein einziges HMM repräsentiert werden.
Die Handhabung der Funktion ’Ersetzen’ soll an dem in Abb. 7.10 gezeigten Beispiel verdeutlicht werden. Der erste Schritt zeigt das eingegebene Original (Canvas) mit der entsprechenden Ausgabe nach erfolgter Erkennung (Dokument). Im zweiten Schritt wurde dann der zu ersetzende Bereich mittels des Steuersymbols selektiert. Schritt drei zeigt, wie nach dem Einfügen des Steuersymbols handschriftlich der ersetzende Ausdruck hinzugefügt wurde.

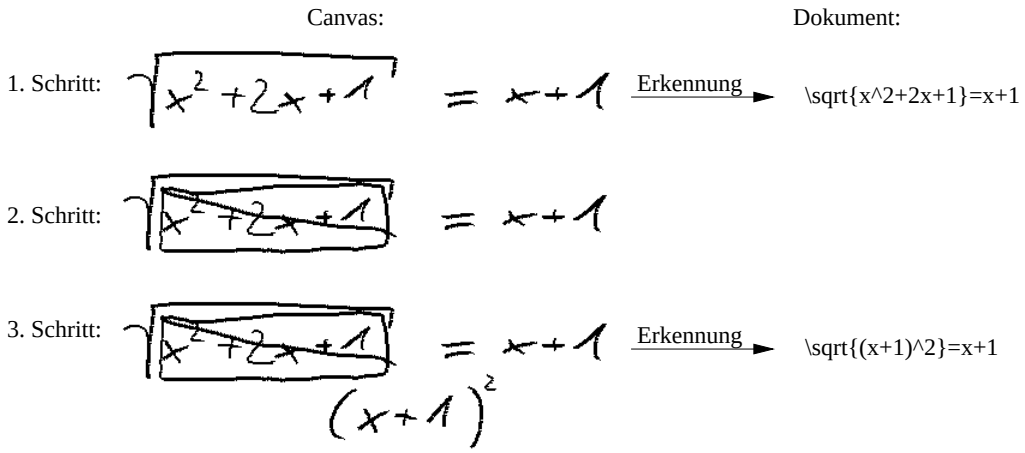


Abbildung 7.10: Ersetzen von Teilausdrücken

Nach einer abschließenden Erkennung in Schritt drei werden die Änderungen in das Ausgabedokument übertragen.
Die im Beispiel gezeigte Reihenfolge der Eingabe von

1. Steuersymbol und

2. ersetzendem Ausdruck bestehend aus Nicht-Steuersymbolen

ist ein wesentliches Merkmal für die Umsetzung der Funktion ’Ersetzen’. Wie bereits erwähnt, werden im zweiten Erkenneraufruf lediglich die hinzugekommenen Daten von der Oberfläche an den Erkenner übergeben. In dem gezeigten Beispiel heißt dies, dass der Parser nach der Ermittlung der geometrischen Merkmale die folgende Zeichensequenz mit entsprechenden geometrischen Merkmalen empfängt: `’ctrl ( x + 1 ) 2’`. Tritt - wie in dem vorliegenden Fall - ein Steuersymbol auf, ist dies grundsätzlich ein Signal für den Parser Änderungen an einem vorherigen Dokument vorzunehmen. Im Unterschied zu dem Beispiel in Abb. 7.8, wo auf ein Steuersymbol kein Nicht-Steuersymbol folgte, erkennt der Parser im Beispiel in Abb. 7.10, dass eine Substitution vorzunehmen ist daran, dass dem Steuersymbol eine Zeichenkette aus Nicht-Steuerzeichen folgt. Die Bestimmung der zu ersetzenden Zeichen verläuft zunächst analog zu der Vorgehensweise bei der Funktion ’Löschen’. Im einem weiteren Schritt wird dann das Erkennungsergebnis strukturiert, welches dem Steuersymbol folgt. Dies erfolgt in der beschriebenen Weise, wie bei einer Erkennung ohne manuelle Korrekturen. Um mehrere Substitutionen in einem Arbeitsschritt vornehmen zu können, ist es auch hier möglich wiederholt die Folge ’Steuersymbol, ersetzender Ausdruck’ einzugeben.
Die eigentliche Ersetzung kann nun erfolgen, indem der durch das führende Steuerzeichen selektierte zu ersetzende Ausdruck aus der verketteten Liste entfernt wird und an dieser Stelle der ersetzende Ausdruck eingefügt wird. Für multiple Substitutionen wird dieser Vorgang analog für jedes auftretende Steuersymbol wiederholt. Um auch hier Überlappungen oder Lücken zu vermeiden, werden die nachfolgenden verbleibenden Zeichen um einen Offset verschoben. Dieser Offset wird aus den Differenzen der umschreibenden Rechtecke von ersetzendem und zu ersetzendem Ausdruck ermittelt. Das Ergebnis wird wieder ausgegeben und aktualisiert auf dem Stapel abgelegt.

7.6.4 Einfügen

Bei der Funktion ’Einfügen’ handelt es sich im Grunde im eine spezielle Variante einer Substitution. Wie das Beispiel in Abb. 7.11 zeigt, wird mittels eines sehr schmalen Steuersymbols die Stelle markiert, an die der ersetzende Ausdruck eingefügt werden soll. Der Parser erkennt den Korrekturbefehl ’Einfügen’ daran, dass wie bei einer Substitution ein Steuersymbol von Nicht-Steuersymbolen gefolgt wird, wobei das Steuersybmol keinen Mit-

telpunkt irgend eines anderen Zeichens des vorausgegangenen Erkennungsergebnisses überdecken darf. Nach erfolgter Strukturierung der nachfolgenden Nicht-Steuerbefehle werden diese wiederum an entsprechender Stelle in die Liste eingefügt. Die ursprünglich vorhandenen Zeichen werden wiederum um einen Offset verschoben, um Überlappungen zu vermeiden. Auch hier gilt, dass mehrere Korrekturanweisungen in einem Erkenneraufruf abgearbeitet werden können.

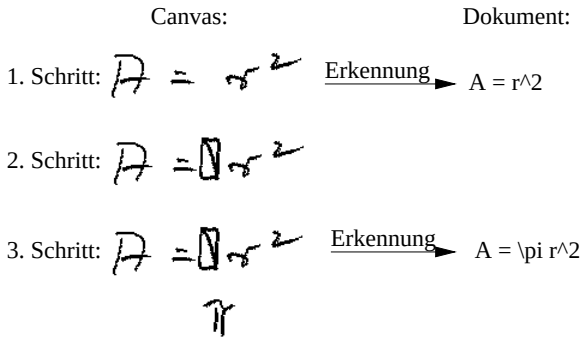


Abbildung 7.11: Einfügen von Teilausdrücken

7.6.5 Rückgängig und Wiederholen

Um während der Entwicklung einer Formel den Zugriff auf verschiedene Versionen zu ermöglichen, wurden die Funktionen ’Rückgängig’ und ’Wiederholen’ realisiert. Mit Hilfe dieser Funktionen können vom Benutzer eingeführte Änderungen zurückgenommen bzw. wieder hinzugefügt werden.
Die Umsetzung dieser Funktionen ist aufgrund der gewählten Struktur für die Historie relativ effizient umzusetzen. Ausgehend von der aktuellen Version eines Dokumentes entspricht der Aufruf von ’Rückgängig’ (’Wiederholen’) dem absteigen (aufsteigen) innerhalb des Stapels. Bei wiederholtem Aufruf der Funktion ’Rückgängig’ liegt das aktuelle Stapelelement natürlich unterhalb des obersten Stapeleintrags. Modifikationen der jetzt aktuellen Version lassen alle späteren Einträge auf dem Stapel ungültig werden. Diese Einträge werden somit aus dem Stapel gelöscht.

7.6.6 Neuzeichnen

Wie die vorangegangenen Beispiele zeigen, kann bereits nach einer Korrekturmaßnahme die handschriftliche Eingabe recht unleserlich werden. Ein weiterer Effekt durchgeführter Modifikationen ist, dass die Divergenz zwischen Eingabe- und Ausgabedokument mit jeder weiteren Änderung zunimmt. Sollen nun zusätzliche Änderungen auf einem bereits modifizierten Dokument vorgenommen werden, könnte diese Divergenz zu Problemen führen, da

vom Benutzer gesetzte Steuersymbole - bedingt durch Verschiebungen aus früheren Ersetzungen oder Einfügungen - nicht mehr exakt die nun zu ersetzenden Zeichen treffen. Die visuelle Rückkopplung zur Stiftführung an den Benutzer ist nicht mehr gegeben. Zu diesem Zweck wurde die Funktion 'Neuzeichnen' eingeführt, welche die Handschrifteingabe an das Ausgabedokument anpasst und automatisch nach jeder manuellen Korrektur ausgeführt wird. Optional kann anschließend das automatisch angeglichene handschriftliche Eingabedokument geladen und im Canvas visualisiert werden.
Die Funktion 'Neuzeichnen' erzeugt aus den aktuellen Änderungen und dem vorangegangenen Erkennungsergebnis mit entsprechender Segmentierungsinformation und geometrischen Merkmalen der handschriftlichen Eingaben eine aktualisierte Synthese der handgeschriebenen Formel.
Die einzelnen Segmente einer Formel werden dazu zunächst anhand der zeitlichen Segmentgrenzen aus dem Eingabedatenstrom ausgeschnitten. Wenn nun Verschiebungen aufgrund von Löschungen oder Einfügungen vorgenommen wurden, existiert für das betroffene Listenelement ein von Null verschiedener Offset, der aus jeder Verschiebung ggf. individuell mitgeführt wurde. Dieser Offset kann nun auf die Kartesischen Koordinaten der Abtastvektoren eines jeden Segmentes aufaddiert werden.
Diese Vorgehensweise soll anhand des Beispiels in Abb. 7.12 verdeutlicht werden. Schritt eins zeigt bereits die im Ausgabedokument durchgeführte Änderung und die handschriftliche Eingabe mit hinzugefügtem Steuersymbol und einzufügendem π. In dem gezeigten Beispiel ist das 'r' sowie die hochgestellte '2' von der Verschiebung betroffen, da diese sich rechts neben dem Steuersymbol befinden (vergl. LRTD-Bedingungen). Die das 'r' und die '2' betreffenden Listenelemente enthalten also einen von Null verschiedenen Offset, der anzeigt um wieviel und in welche Richtung die geoemtrischen Merkmale korrigiert wurden. Um

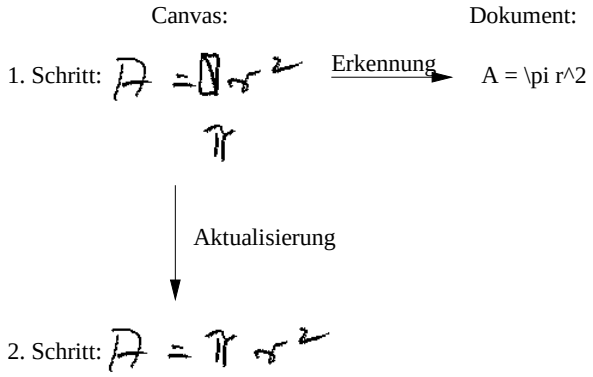


Abbildung 7.12: Aktualisierung des Schriftbildes

das an das Ausgabedokument angeglichene Schriftbild zu generieren, werden anhand der zeitlichen Segmentierung die einzelnen Symbole aus der Abtastsequenz ausgeschnitten. Die Kartesischen Koordinaten dieser Abtastvektoren enthalten zunächst noch die ursprünglichen

Werte. Da für die Zeichen 'A' und '=' kein Offset existiert, wird die originale Abtastfolge übernommen. Die Positionen des nach dem Steuersymbol auftretenden Zeichens π sowie das 'r' und die '2' wurden jedoch in der Funktion 'Einfügen' korrigiert und mit einem entsprechenden Offset versehen, welcher auf die Abtastfolge $(x, y)(k)$ des jeweiligen Segmentes aufaddiert wird.
Für die weitere Verarbeitung insbesondere für eine wiederholte Erkennung ist es notwendig die Abtastfolge der Zwischenräume zu rekonstruieren, da diese ja bei der Erkennung einem speziellen 'space'-Modell zugeordnet werden. Das Fehlen der Zwischenräume in der zeitlichen Abfolge würde zwangsweise zu Erkennungsfehlern führen. Diese Zwischenräume können durch lineare Interpolation zwischen Endpunkt eines Zeichens und Anfangspunkt des Nachfolgezeichens rekonstruiert werden. Die 'Spaces' am Anfang vor dem ersten Zeichen und am Ende nach dem letzten Zeichen können aus dem jeweiligen Original kopiert werden. Die Länge der Abtastvektoren dieser Zwischenräume bzw. deren Anzahl kann willkürlich gewählt werden, da bei jeder Erkennung zunächst eine Neuabtastung vorgenommen wird, die für äquidistante Abtastpunkte sorgt.
Das so generierte Schriftbild, repräsentiert durch die rekonstruierte Abtastfolge, kann schließlich visualisiert werden und für eine Weiterverarbeitung verwendet werden.

7.7 Ergebnisse

Bei der Bewertung des Formeleditors ist es zweckmäßig die Bereiche Ersteingabe und Modifikation zu trennen. Die Erkennungsgenauigkeit bei der Ersteingabe kann, analog zu der Worterkennung, wieder anhand eines Test-Sets geschätzt werden. Die entsprechenden Werte für die drei Schreiber 'ank', 'bec' und 'sla' sowie die Duchschnittswerte sind in Tab. 7.1 zusammengestellt.
Zu unterstreichen ist hierbei jedoch, dass die Erkennungsraten auf der Ausgabe des Dekoders berechnet wurden. Die Bewertung der Formelerkennung auf Basis der Erkennungsraten kann daher nur eine ungefähre Idee von der eigentlichen Benutzbarkeit des Systems geben, da in einfacher Form nur die Fehler der Dekoderausgabe meßbar sind. Um Fehler des Parsers zu messen, müßte ein umfangreiches mathematisches Regelwerk für die Bewertung genutzt werden, da hierbei wiederum verstärkt die kontextbezogene Bedeutung der einzelnen Symbole einzubeziehen wäre.

	ank	bec	sla	ø
Korrektheit	97.4 %	97.5 %	89.5 %	94.8 %
Akkuratheit	96.6 %	97.1 %	89.4 %	94.4 %

Tabelle 7.1: Erkennungsraten des schreiberabhängigen Formeleditors

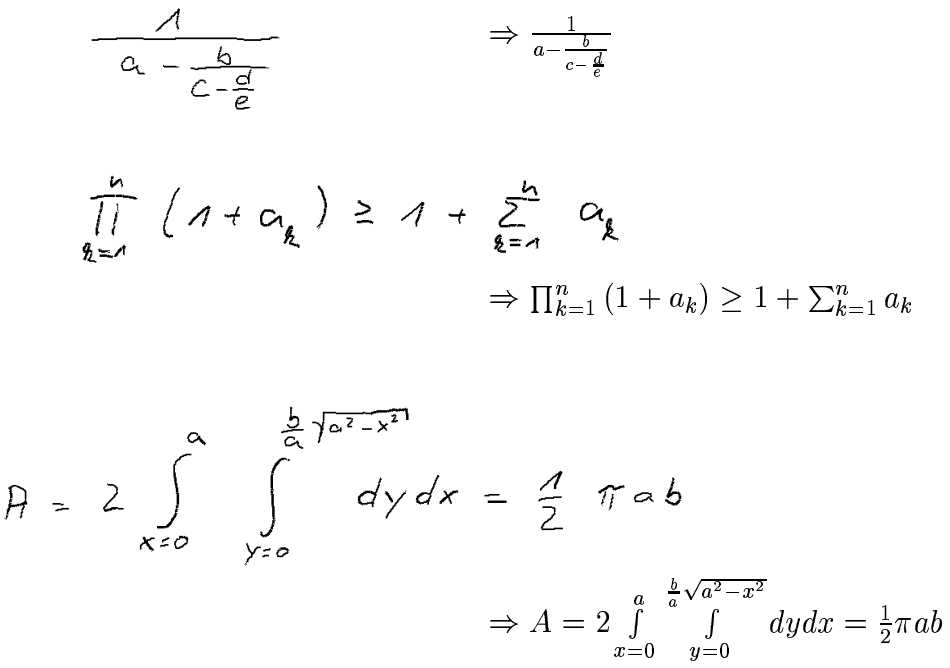


Abbildung 7.13: Korrekt erkannte Beispiele

Dazu betrachte man die folgenden Beispiele. Wird eine Zahl, die als Index durch den Schreiber tiefgestellt wurde, durch den Dekoder korrekt erkannt, durch den nachfolgenden Parser aber fehlerhaft als *in-line* gesetzt, statt als Index, so wird dieser Fehler in der Berechnung der Erkennugsrate außer Betracht bleiben. Ein Durch den Dekoder gelöschter Multiplikationspunkt zwischen einem Koeffizienten und einer Funktion würde hingegen als Fehler gezählt werden, obwohl beide Ausdrücke gültig und darüber hinaus auch gleichwertig sind.
Dennoch ist festzuhalten, dass der Parser um so genauere Ergebnisse liefert je genauer das Alignment von dem Dekoder geliefert wird. Das Alignment wiederum ist dann exakt, wenn die Modelle genau sind. Die Genauigkeit der Modelle spiegelt sich wiederum in der Erkennungsrate - meßbar nach dem Dekoder - wieder, was die Betrachtung der Erkennungsraten in gewisser Weise auch bei dem Formeleditor rechtfertigt. Ergänzend zu den Ergebnissen in Tab. 7.1 sind in Abb. 7.13 einige korrekt erkannte Beispiele des Schreibers 'ank' wiedergegeben, die durchaus komplexe Strukturen aufweisen und von dem System korrekt umgesetzt werden.
Die Ergebnisdiskussion hat gezeigt, dass sich die Erfassung meßbarer Ergebnisse bei der Erkennung von Ersteingaben schwierig gestaltet. Dies gilt um so mehr bei der Bewertung der Editierfunktionen. Die Interaktionen sind hier mit Dialogen vergleichbar, bei denen sich Aktionen und Reaktionen von Mensch und Maschine gegenseitig beeinflussen. Derartige Systeme sind lediglich durch sehr aufwendige Benutzbarkeitsstudien an einer größeren Anzahl Benutzer zu evaluieren und im Rahmen dieser Arbeit nicht realisierbar.

7.8 Kapitelzusammenfassung

Ein Schwerpunkt aktueller Forschungsarbeiten in dem Bereich der Formelerkennung bildet nach wie vor die Untersuchung der unteren Systemebenen, wie Segmentierung und anschließende Erkennung der Einzelzeichen. In dieser Arbeit konnte hingegen gezeigt werden, dass die integrierte Erkennung und Segmentierung durch einen HMM-basierten Ansatz wesentliche Vorteile gegenüber klassischen Verfahren bietet. Das Segmentierungsproblem als solches tritt explizit nicht mehr auf. Des weiteren konnte die Segmentierungsinformation des Dekoders für weitere Verarbeitungsschritte genutzt werden. Insbesondere wurde die Segmentierungsinformation für das Parsing der Dekoder-Ausgabe verwendet.

Im Bereich der handschriftlichen Formelerkennung war bisher die Erkennung und Ausführung manueller Korrekturbefehle unberücksichtigt. Mit der Einführung universell verwendbarer Editiersymbole konnten die Befehle ’Löschen’, Einfügen’ und ’Ersetzen’ realisiert werden. Die anschließende Synthese des Schriftbildes aus dem aktualisierten Erkennungsergebnis sorgt für die nötige Synchronisation zwischen Eingabe und Ausgabedokument. Schließlich wurde die gesamte Funktionalität in einen nahezu echtzeitfähigen Demonstrator integriert, dessen Oberfläche in Abb. 8.1 gezeigt ist.

Kapitel 8

Zusammenfassung

In den vorangegangenen Kapiteln wurden verschiedene Aspekte zur Online-Handschrifterkennung beleuchtet, mit denen die gesamte Verarbeitungskette von der Vorverarbeitung, Merkmalsextraktion, Modellierung bis hin zur Erkennung erfasst wird. Neben der reinen Texterkennung konnten weiterhin Ansätze für die Erkennung handgeschriebener Formeln präsentiert werden, die zusammen mit diversen Editierfunktionen in einen interaktiven Demonstrator integriert werden konnten.

Den Schwerpunkt in Kapitel 2 bildete die Vorverarbeitung der digitalisierten Stifttrajektorie mit der Neuabtastung und der Normalisierung. Insbesondere konnten hier Histogrammverfahren für die Online-Handschrifterkennung adaptiert werden, mit denen sich sowohl die Zeilenneigung, wie auch die Schriftneigung automatisch korrigieren lassen. Durch die Minimierung eines Entropiemaßes konnte speziell für die als problematisch geltende Schriftneigungskorrektur eine Alternative zu den bisher verwendeten Richtungshistogrammen erarbeitet werden. Die iterative Bereichsdetektion für die Schriftgrößenskalierung komplementiert die für eine vollständige Normalisierung notwendigen Funktionen.

Kapitel 3 stellt eine Auswahl der für die Online-Handschrifterkennung untersuchten Merkmale dar. Hierbei wurden insbesondere zwei Konzepte zur Merkmalsentnahme intensiver untersucht. Dies sind zum einen die Trajektorienmerkmale und zum anderen die Bitmap-Merkmale. Beide Ansätze machen von der dynamischen Information der Stifttrajektorie Gebrauch, wobei die Bitmap-Merkmale in zeitlich globaler Weise alle für einen geometrisch begrenzten Bereich relevanten Ereignisse berücksichtigen. Insgesamt wurden zahlreiche Trajektorien- und Bitmap-Merkmale untersucht, von denen eine Auswahl von 5 Trajektorien- und 3 Bitmap-Merkmalen näher präsentiert wurde. Über die Betrachtung von Einzelmerkmalen hinaus wurde eine optimale Merkmalskombination gefunden. Aufgrund der erzielten Erkennungsleistungen sind dabei die Trajektorienapproximation durch Bézierkurven in Kombination mit der DCT-Kodierung der gleitenden Bitmap besonders hervorzuheben. Beide Verfahren waren aus der Literatur bisher nicht bekannt. Speziell mit

der Singulärwertzerlegung konnte für die Trajektorienapproximation durch Bézierkurven ein sehr effizienter und genauer Algorithmus gefunden werden.

Unter der Berücksichtigung der dynamischen Eigenschaften der online abgetasteten Handschrift bietet ein Hidden Markov Modell basierter Ansatz besondere Vorteile. In Kapitel 4 wurden erstmals für die Online-Handschrifterkennung verschiedene HMM-basierte Modellierungsverfahren eingehender untersucht. Dazu wurden diskrete, kontinuierliche und hybride Ansätze anhand schreiberabhängiger Systeme mit sehr großem Vokabular miteinander verglichen. Als überraschendes Ergebnis konnte der erstmalig für die Handschrifterkennung eingesetzte hybride Ansatz als beste Modellierungsform identifiziert werden. Bei dieser hybriden Modellierungsform werden neuronale Vektorquantisierer für die Merkmalsquantisierung eingesetzt. Als Bewertungsfunktion für das Training der VQ wird die Transinformation maximiert, die zwischen Sender (Schreiber) und Empfänger (Erkenner) über die Verarbeitungskette übertragen wird. Durch eine Verallgemeinerung dieses Ansatzes ist es möglich, mehrere VQ anhand der Verbund-Transinformation simultan zu optimieren. Die Verbund-Transinformation konnte darüber hinaus auch für die Vorauswahl von Merkmalskombinationen genutzt werden, da hierbei die Korrelation der Einzelmerkmale einfließt.

Einen weiteren Modellierungsaspekt stellt die Einbeziehung von Kontextinformation dar (Kapitel 5), die im Bereich der Handschrifterkennung bisher weitgehend unberücksichtigt blieb. Im Rahmen dieser Arbeit wurden zum ersten Mal umfassende Untersuchungen zur Nutzung von Kontextinformation für die automatische Handschrifterkennung durchgeführt. Der Vorteil kontextbezogener Graphemmodelle liegt klar in der höheren Auflösung der Modelle. Kontraproduktiv wirkt sich auf die Trainierbarkeit der Modelle hingegen die z. T. dramatisch ansteigende Parameteranzahl aus. Im Gegenzug dazu sind diverse Clustering-Verfahren für HMM bekannt, mit denen sich die Parameteranzahl durch Verknüpfungen reduzieren läßt. Zur Adaption dieser Verfahren auf die Handschrifterkennung war es in erster Linie erforderlich Wissen über Kontexteinflüsse zu generieren. Dazu wurden in Kapitel 5 verschiedene leistungsfähige Ansätze entwickelt, deren Eignung anhand schreiberabhängiger Systeme mit Lexika von bis zu 200000 Wörtern getestet wurden.

Ein Erkennungssystem, bei dem die Eingabe verbundener Handschrift zugelassen wird, ist im allgemeinen auf ein vorgegebenes Lexikon angewiesen. Zur Abschwächung dieser Beschränkungen wurden die Forschungsarbeiten von den Bestrebungen begleitet eine maximale Lexikongröße zu realisieren. Wie in Kapitel 6 gezeigt wird, kann alternativ zu einem vom Umfang begrenzten, fest vorgegebenen Lexikon auch eine statistische Beschreibung von Buchstabenabhängigkeiten herangezogen werden. Der Vorteil ist, dass eine deterministische Definition gültiger Buchstabensequenzen mittels eines Lexikons damit überflüssig wird. Die durchgeführten Experimente belegen, dass sich eine brauchbare Genauigkeit jedoch erst mit einem relativ großen Kontext von mehr als 6 vorausgehenden

Zeichen einstellt. Solche Kontexttiefen sind bisher nicht berücksichtigt worden, da sie mit gebräuchlichen Dekodierverfahren nicht mehr beherrschbar sind. Als sinnvolle Alternative hat sich hier die Verwendung eines Stack-Decoders erwiesen.
Wird in einem System die Text- oder Befehlseingabe über einen Stift vorgesehen, erschließen sich weitere interessante Szenarien, wie die handschriftliche Eingabe von Formeln zur Dokumenterstellung oder auch für handschriftliche Taschenrechner. Methoden für die Realisierung dieses Szenarios wurden in Kapitel 7 entwickelt. Das Erkennungsergebnis wird dabei in einer LaTeX-Struktur ausgegeben. Darüber hinaus wurden erstmals Konzepte für eine automatische Erkennung und anschließende Ausführung handschriftlicher Korrekturbefehle für Formeln präsentiert.

Demonstrator
Die Evaluierung verschiedener Methoden anhand konkreter Erkennungsergebnisse geschieht in der Regel in Form einer Stapelverarbeitung, bei der eine möglichst aussagefähige Menge von Schriftproben verarbeitet, und die Erkennerausgaben abschließend mit den 'Solltexten' verglichen werden. Für die Präsentation der verschiedenen Erkennungssysteme hingegen wurden die entwickelten Methoden und Software-Tools in einen Demonstrator integriert. Abb. 8.1 zeigt die für X-Windows entwickelte Oberfläche des Systems [Kel95, Mes99].
Die über die serielle Schnittstelle von einem Digitalisiertablett abgetasteten Handschriftdaten werden während deren Erzeugung bereits im Canvas visualisiert. Nach einem automatischen oder auch manuellen Start der Erkennung werden die Rohdaten über eine Pipeline an ein weiteres Programm im Hintergrund gesendet, welches die eigentliche Erkennung ausführt. Aus Flexibilitätsgründen ist es natürlich auch möglich den Ablauf der Erkennung im Hintergrund in Form von Skripten zusammenzufassen. Die Ausgaben des jeweiligen Erkennungssystems werden an die Oberfläche zurückgesendet. Statusmeldungen, sowie der erkannte Text können somit durch die Oberfläche angezeigt werden (linkes unteres, bzw. linkes oberes Fenster des 'NeuroGraphen' in Abb. 8.1).
Im Falle der (dargestellten) Formelerkennung wird das Erkennungsergebnis nicht wie bei der Texterkennung im integrierten linken oberen Fenster des NeuroGraphen dargestellt. Zur Visualisierung der erkannten Struktur wird die LaTeX-Ausgabe des Erkenners zunächst kompiliert und über einen DVI-Viewer extern dargestellt. Die Korrekturfunktionen für die Formelerkennung folgen dem gleichen Prinzip. Dazu werden die manuellen Korrekturen mittels eines Stiftes auf dem Canvas vorgenommen. Nach erneuter Erkennung und nach Ausführung der Korrekturen wird das externe Ausgabefenster, sowie bei Bedarf das korrigierte Schriftbild aktualisiert.

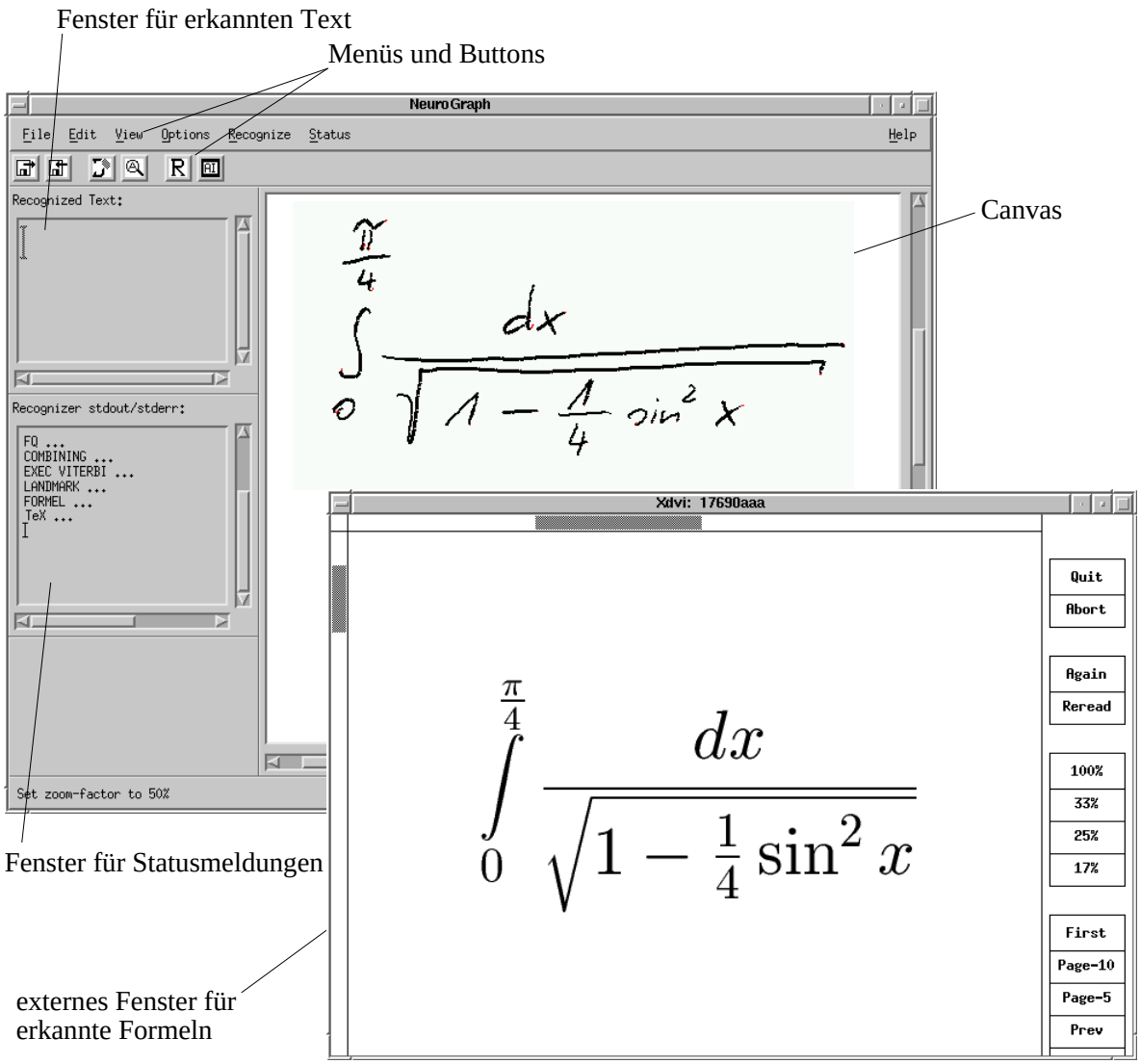


Abbildung 8.1: Oberfläche des Demonstrators

Anhang A

Zusammenfassung grundlegender informationstheoretischer Zusammenhänge

Die Entropie H ist der Erwartungswert des Informationsgehaltes $\mathcal{I}$. Dieser wiederum ist definiert als der negative Logarithmus der Auftrittswahrscheinlichkeit p des Zeichens x:

$$\mathcal{I}(x) = -\log p(x). \tag{A.1}$$

Daraus folgt:

$$H(X) = E\{\mathcal{I}(x)\} = E\{-\log p(x)\} = -\sum_y p(y) \cdot \log p(y), \tag{A.2}$$

bzw für die Verbundentropie

$$H(X,Y) = E\{-\log p(x,y)\} = -\sum_x \sum_y p(x,y) \cdot \log p(x,y). \tag{A.3}$$

Mittels des Satzes von Bayes ($p(x,y) = p(x) \cdot p(y|x) = p(y) \cdot p(x|y)$) läßt sich die Verbundentropie umformen zu

$$H(X,Y) = -\sum_x \sum_y p(x,y) \cdot \log\{p(y) \cdot p(x|y)\} = -\sum_x \sum_y p(x,y) \cdot \log\{p(x) \cdot p(y|x)\} \tag{A.4}$$

und

$$H(X,Y) = H(Y) \underbrace{-\sum_x \sum_y p(x,y) \cdot \log p(x|y)}_{H(X|Y)} = H(X) \underbrace{-\sum_x \sum_y p(x,y) \cdot \log p(y|x)}_{H(Y|X)}. \tag{A.5}$$

Die Gleichung $H(X,Y) = H(Y) + H(X|Y) = H(X) + H(Y|X)$ führt unmittelbar zu dem informationstheoretischen Kanalmodell, welches die Beziehung für die Transinformation I

liefert:

$$I(X,Y) = H(Y) - H(Y|X) = H(X) - H(X|Y) \tag{A.6}$$

Ersetzt man jeweils die Entropie mittels obiger Gleichungen, ergibt sich:

$$I(X,Y) = -\sum_x p(x) \cdot \log p(x) + \sum_x \sum_y p(x,y) \cdot \log p(x|y), \tag{A.7}$$

wobei die erste Summe folgendermaßen umgeformt werden kann:

$$I(X,Y) = -\sum_x \sum_y p(x,y) \cdot \log p(x) + \sum_x \sum_y p(x,y) \cdot \log p(x|y), \tag{A.8}$$

was sich wiederum zsammenfassen läßt zu

$$I(X,Y) = \sum_x \sum_y p(x,y) \cdot \log \frac{p(x|y)}{p(x)}. \tag{A.9}$$

Die bedingte Wahrscheinlichkeit läßt sich mit Hilfe des Satzes von Bayes ($p(x,y) = p(x) \cdot p(y|x) = p(y) \cdot p(x|y)$) in eine Verbundwahrscheinlichkeit überführen:

$$I(X,Y) = \sum_x \sum_y p(x,y) \cdot \log \frac{p(x,y)}{p(x) \cdot p(y)}. \tag{A.10}$$

Die Transinformation I läßt sich darüberhinaus allgemein für den Fall einer Z-dimensionalen Wahrscheinlichkeitsdichtefunktion angeben:

$$I(X^{(1)}, \cdots, X^{(Z)}) = \sum_{x^{(1)}} \cdots \sum_{x^{(Z)}} p(x^{(1)}, \cdots, x^{(Z)}) \cdot \log \frac{p(x^{(1)}, \cdots, x^{(Z)})}{\prod_{z=1}^{Z} p(x^{(z)})}. \tag{A.11}$$

Anhang B

Beschreibung der verwendeten Datenbasen

Sämtliche Daten wurden mit einer konstanten Abtastfrequenz von 200 Hz abgetastet. Dazu wurden ausschließlich Digitalisertabletts der Firma WACOM verwendet. Neben den Kartesischen Koordinaten des Stiftes, die mit einer Auflösung von 300 dpi aufgenommen wurden, wird zu jedem Abtastzeitpunkt der Stiftdruck mit einer maximalen Auflösung von 8 Bit gelesen. Üblicher Weise wird hiervon für die weitere Verarbeitung lediglich das Vorzeichenbit ausgewertet. Die verwendeten WACOM-Boards liefern bei positivem, wie bei negativem Stiftdruck die Information über die Stiftposition. Positiver Stiftdruck bedeutet aufgesetzten Stift, während negativer Stiftdruck bei abgehobener Stiftspitze gemessen wird. Auf diese Weise sind in den Rohdaten auch die Bewegungen des abgesetzten Stiftes (virtuelle Sequenzen) bis zu einer Höhe von ca. 2 cm über dem Board enthalten. Die Daten werden schließlich in kompakter Form in einem speziellen Binärformat abgelegt.
Unter diesen Randbedingungen wurden im wesentlichen drei Datenbasen zusammengestellt. Dies sind die schreiberabhängige Textdatenbasis, die schreiberunabhängige Textdatenbasis und die schreiberabhängige Formeldatenbasis, die nachfolgend näher beschrieben werden.

B.1 Schreiberabhängige Textdatenbasis

Die schreiberabhängige Datenbasis beinhaltet Trainings- und Testdaten von insgesamt drei Schreibern (’ank’, ’jmr’ und ’vdm’). Es wurden von jedem Schreiber Textpassagen zum Training und Einzelworte für den Test aufgenommen. Die Trainingsmenge umfaßt pro Schreiber 2000 Wörter, während die Testmenge aus je 200 Wörtern besteht. Dies entspricht ca. 500000 (50000) Abtastvektoren zum Training (Test). Des weiteren wurden für die Initialisierung der HMM segmentierte Einzelzeichen mit Hilfe eines Formblattes aufgenommen, sodass sich durch die Anordnung der Felder ein definierter geometrischer Abstand zwischen den einzel-

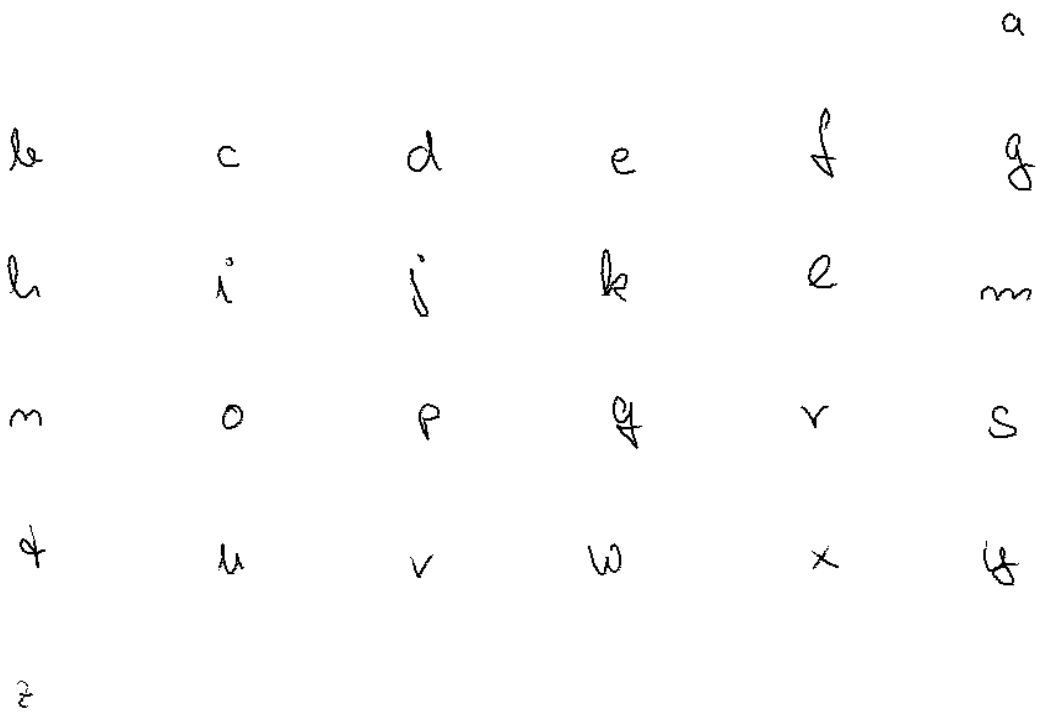


Abbildung B.1: Vorsegmentierte Einzelzeichen

nen Zeichen einstellt. Ein Beispiel zu den vorsegmentierten Einzelzeichen ist in Abb. B.1 gezeigt. In der Datenbasis sind neben Klein- und Großbuchstaben mit Umlauten und dem ’ß’ auch Ziffern, sowie sämtliche Satzzeichen enthalten.

B.2 Schreiberunabhängige Textdatenbasis

Die schreiberunabhängige Textdatenbasis läßt sich wie die schreiberabhängige Datenbasis in einen Initialisierungsteil und in eine Trainings- und Testmenge unterteilen. Der Zeichenvorrat ist identisch zu dem der schreiberabhängigen Datenbasis. Von einer relativ kleinen Anzahl von Schreibern wurden in der oben beschriebenen Weise die vorsegmentierten Daten gesammelt.

Die Datensätze der Trainingsdatenbasis bestehen wie im schreiberabhängigen Fall aus Textpassagen. Diese weisen unterschiedliche Längen von etwa 5-20 Wörtern auf. Der einzelne Datensatz reicht dabei durchaus über mehrere Zeilen (Abb. B.2). Über einen Zeitraum von zwei Jahren wurden zwei verschiedene Teilmengen der Trainingsdatenbasis akquiriert (*set0* und *set1*). *set0* umfasst dabei 8000 Beispielwörter von 50 Schreibern, während *set1* 24000 Beispielwörter von 100 Personen enthält. Beide Teilmengen fließen gleichgewichtet in das Training ein. Um eine Aussage über die Schreiberunabhängigkeit zu erhalten, wurde die Testmenge aus ’ungesehenen’ Testschreibern zusammengestellt. D. h., dass in den Daten zur Initialisierung, wie auch in den Daten für das Training keine Beispiele von Schreibern

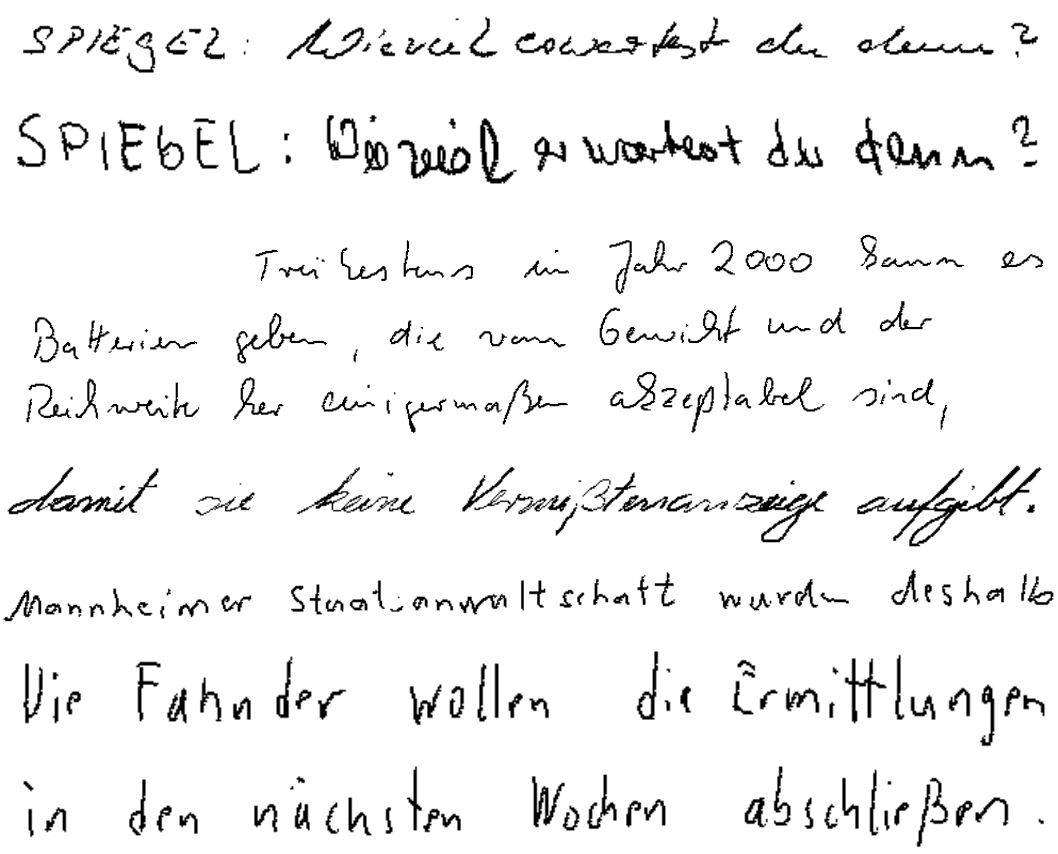


Abbildung B.2: Textpassagen für das Training

der Testmenge enthalten sind und umgekehrt. Die Testdaten bestehen wiederum aus Einzelworten, die von je 10 männlichen und 10 weiblichen Schreibern gesammelt wurden. Von jeder Testperson wurden jeweils ca. 200 Wörter aufgenommen, sodass sich insgesamt 4000 Wörter im Testset befinden. Diese Testmenge deckt ein Vokabular von ca. 2000 verschiedenen Wörtern ab.

B.3 Schreiberabhängige Formeldatenbasis

Wie bei der schreiberabhängigen Textdatenbasis (Abschnitt B.1) setzt sich die in diesem Abschnitt beschriebene Datenbasis aus einem Initialisierungsteil, einer Trainings- und einer Testmenge zusammen. Die Aufnahmen wurden von insgesamt drei Schreibern gesammelt (’ank’, ’bec’, ’sla’). Hierbei wurden für die Initialisierung von jedem Schreiber segmentierte Einzelzeichen aller vorkommenden Symbole aufgenommen. Neben dem Standard-Zeichenvorrat, wie er bei der Texterkennung verwendet wird, sind dies verschiedene Operatoren, Sonderzeichen, und griechische Buchstaben.

Der Hauptteil, d. h. die Trainings- und die Testmenge der Formeldatenbasis besteht hingegen aus je 130 unsegmentierten Gleichungen, wie sie beispielhaft in Abb. B.3 dargestellt sind.

Von diesen 130 Trainingsbeispielen werden pro Schreiber 100 Proben für das Training, und die jeweils restlichen 30 Proben für die Evaluierung verwendet.

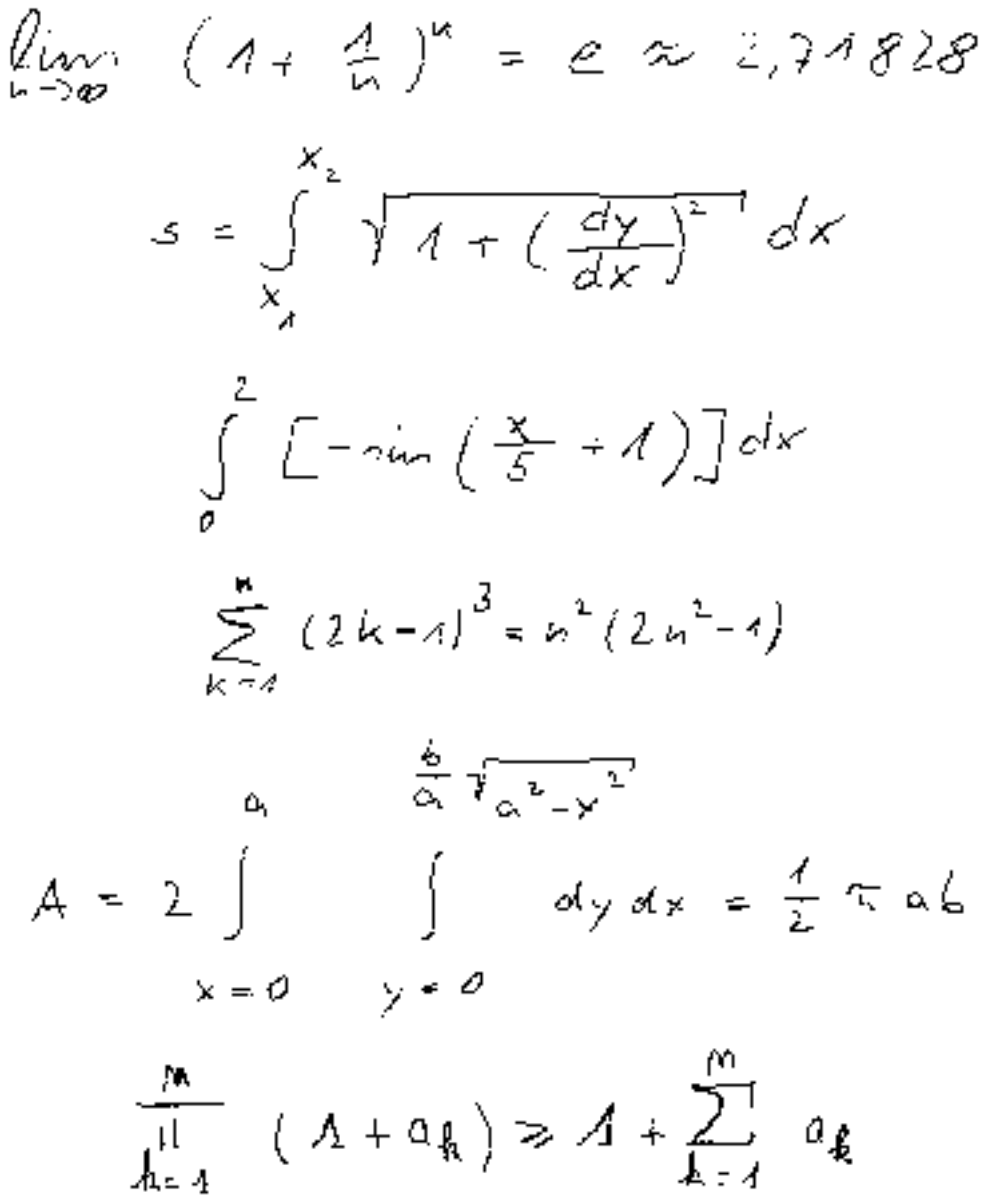


Abbildung B.3: Beispiele aus der Formeldatenbasis (Test und Training)

Literaturverzeichnis

[Abm94] WOLFGANG ABMAYR. *Einführung in die digitale Bildverarbeitung*. B. G. Teubner, Stuttgart, 1994.

[Bah81] L. R. BAHL, R. BAKIS, P. S. COHEN, A. G. COLE, F. JELINEK, B. L. LEWIS und R. L. MERCER. Continuous parameter acoustic processing for speech recognition of a natural speech corpus. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Seiten 1149–1155, Atlanta, Georgia, April 1981.

[Bau69] L. E. BAUM. An Inequality and Associated Maximization in Statistical Estimation for Probabilistic Functions of Markov Processes. In *Inequalities-III, Proc. of the third Symposium on Inequalities*, Seiten 1–8, Los Angeles, CA, September 1969.

[Bec97] DIRK BECKER. Entwicklung eines handschriftlichen formeleditors. Diplomarbeit, Mercator University Duisburg, 1997.

[Blo95] DOROTHEA BLOSTEIN. General Diagram-Recognition Methodologies. In *Proc. Int. Workshop on Graphics Recognition*, Seiten 200–212, University Park, Pennsylvania, 1995.

[Blo96] DOROTHEA BLOSTEIN, HODA FAHMY und ANN GRBAVEC. Issues in the Practical Use of Graph Rewriting. *Lecture Notes in Computer Science*, 1073:38–55, 1996.

[Blo97] DOROTHEA BLOSTEIN und ANN GRBAVEC. Recognition of Mathematical Notation. In H. BUNKE und P. S. P. WANG, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 21, Seiten 557–582. World Scientific Publishing, 1997.

[Bra99a] ANJA BRAKENSIEK, ANDREAS KOSMALA, DANIEL WILLETT und GERHARD RIGOLL. Vergleich verschiedener statistischer Modellierungsverfahren für die

On- und Off-Line Handschrifterkennung. In *21. DAGM-Symposium, Tagungsband Springer-Verlag*, Seiten 70–77, Bonn, Germany, September 1999.

[Bra99b] ANJA BRAKENSIEK, ANDREAS KOSMALA, DANIEL WILLETT, WENWEI WANG und GERHARD RIGOLL. Performance Evaluation of a New Hybrid Modeling Technique for Handwriting Recognition Using Identical On-Line and Off-Line Data. In *5th International Conference on Document Analysis and Recognition*, Seiten 446–449, Bangalore, India, September 1999.

[Bra00] ANJA BRAKENSIEK, JÖRG ROTTLAND, ANDREAS KOSMALA und GERHARD RIGOLL. Off-Line Handwriting Recognition Using Various Hybrid Modeling Techniques and Character N-Grams. In *7th International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, Amsterdam, Netherlands, September 2000.

[Bro87] P. F. BROWN. *The Acoustic-Modeling Problem in Automatic Speech Recognition*. Dissertation, Carnegie-Mellon University, Mai 1987. Also IBM Research Division Technical Report RC 12750.

[Cla97] P.R. CLARKSON und R. ROSENFELD. Statistical Language Modeling Using the CMU-Cambridge Toolkit. In *Proc. ESCA Eurospeech*, 1997.

[Cot97] M. COTÉ, M. CHERIET, E. LECOLINET und C. Y. SUEN. Automatic reading of cursive scripts using human knowledge. In *Proc. Int. Conference on Document Analysis and Recognition (ICDAR)*, Volume 1, Seiten 107–108, Ulm, 1997.

[Eic97a] STEFAN EICKELER, ANDREAS KOSMALA und GERHARD RIGOLL. A New Approach to Content-Based Video Indexing Using Hidden Markov Models. In *Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, Seiten 149–154, Louvain-la-Neuve, Belgium, Juni 1997.

[Eic97b] STEFAN EICKELER, ANDREAS KOSMALA und GERHARD RIGOLL. Echtzeitfähige Videosequenzerkennung mit statistischen Verfahren. In *19. DAGM-Symposium, Tagungsband Springer-Verlag*, Seiten 105–112, Braunschweig, Germany, September 1997.

[Eic98] STEFAN EICKELER, ANDREAS KOSMALA und GERHARD RIGOLL. Hidden Markov Model Based Continuous Online Gesture Recognition. In *Int. Conference on Pattern Recognition (ICPR)*, Seiten 1206–1208, Brisbane, August 1998.

[Eic00] STEFAN EICKELER, STEFAN MÜLLER und GERHARD RIGOLL. Recognition of JPEG Compressed Face Images Based on Statistical Methods. *Image and Vision*

Computing Journal, Special Issue on Facial Image Analysis, 18(4):279–287, März 2000.

[Far94] GERALD FARIN. *Kurven und Flächen im Computer Aided Geometric Design*. Vieweg, second Auflage, 1994.

[Gon92] RAFAEL C. GONZALES und RICHARDS E. WOODS. *Digital Image Processing*. Addison Wesley, 1992.

[Haf99] CARSTEN HAFERKAMP. Untersuchung zu Slant- und Größennormalisierung für die Online-Handschrifterkennung. Diplomarbeit, Mercator University Duisburg, 1999.

[Häm89] GÜNTHER HÄMMERLIN und KARL-HEINZ HOFFMANN. *Numerische Mathematik*. Springer-Verlag, first Auflage, 1989.

[Hü99] FRANK HÜLSKEN. Untersuchung von Merkmalsextraktionsverfahren für die Online-Handschrifterkennung. Diplomarbeit, Mercator University Duisburg, 1999.

[Kas95] ROBERT HOWARD KASSEL. *A Comparison of Approaches to On-Line Handwritten Character Recognition*. Dissertation, Massachusetts Institute of Technology, 1995.

[Kel95] ANDREAS KELDENICH. Anschluß und Erprobung eines Digitalisier-Tableaus zum Aufbau eines adaptiven Handschrifterkennungssystems. Diplomarbeit, Faculty of Electrical Engineering - Computer Science, Gerhard-Mercator-University Duisburg, Februar 1995. in German.

[Kop94] HELMUT KOPKA. *LaTeXeine Einführung*. Kontakt und Studium. Eddison-Wesley, 1994. 624 Seiten.

[Kos97a] ANDREAS KOSMALA, JÖRG ROTTLAND und GERHARD RIGOLL. Improved On-Line Handwriting Recognition Using Context Dependent Hidden Markov Models. In *Proc. Int. Conference on Document Analysis and Recognition (ICDAR)*, Volume 2, Seiten 641–644, Ulm, 1997.

[Kos97b] ANDREAS KOSMALA, JÖRG ROTTLAND und GERHARD RIGOLL. An Investigation of the Use of Trigraphs for Large Vocabulary Cursive Handwriting Recognition. In *Proc. IEEE Int. Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Seiten 3373–3376, Munich, 1997.

[Kos97c] ANDREAS KOSMALA, JÖRG ROTTLAND und GERHARD RIGOLL. Large Vocabulary On-Line Handwriting Recognition with Context Dependent Hidden Markov Models. In *Mustererkennung*, Seiten 254–261, Braunschweig, Germany, 1997.

[Kos98a] ANDREAS KOSMALA und GERHARD RIGOLL. A Hybrid NN/HMM approach for Large Vocabulary On-Line Handwriting Recognition. In *European Congress on Intelligent Techniques and Soft Computing (EUFIT)*, Aachen, Germany, 1998. Invited Session on Soft Computing in Communication Terminals.

[Kos98b] ANDREAS KOSMALA und GERHARD RIGOLL. On-Line Handwritten Formula Recognition Using Statistical Methods. In *International Conference on Pattern Recognition (ICPR)*, Brisbane, 1998.

[Kos98c] ANDREAS KOSMALA und GERHARD RIGOLL. Recognition of On-Line Handwritten Formulas. In *6th International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, Taejon, Korea, 1998.

[Kos98d] ANDREAS KOSMALA und GERHARD RIGOLL. Tree-Based State Clustering Using Self-Organizing Principles for Large Vocabulary On-Line Handwriting Recognition. In *International Conference on Pattern Recognition (ICPR)*, Brisbane, 1998.

[Kos99a] ANDREAS KOSMALA, STEPHANE LAVIROTTE, LOÏC POTTIER und GERHARD RIGOLL. On-Line Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars. In *5th International Conference on Document Analysis and Recognition (ICDAR '99)*, Bangalore, India, 1999.

[Kos99b] ANDREAS KOSMALA und GERHARD RIGOLL. On-line Handwritten Formula Recognition. In SEONG-WHAN LEE, editor, *Advances in Handwriting Recognition*, chapter 9, Seiten 539–548. World Scientific, 1999.

[Kos99c] ANDREAS KOSMALA und DANIEL WILLETT. Datenverarbeitungsverfahren und Datenverarbeitungsvorrichtung zum Erkennen einer zu erkennenden Zeichenfolge, sowie computerlesbares Speichermedium und Computerprogramm-Erzeugnis. DE-Patentanmeldung 199 61 476.8, 12 1999.

[Kos99d] ANDREAS KOSMALA, DANIEL WILLETT und GERHARD RIGOLL. Advanced State Clustering for Very Large Vocabulary HMM-based On-Line Handwriting Recognition. In *5th International Conference on Document Analysis and Recognition (ICDAR '99)*, Bangalore, India, 1999.

[Kos00] ANDREAS KOSMALA und GERHARD RIGOLL. On-line handwritten formula recognition with integrated correction recognition and execution. In *International Conference on Pattern Recognition (ICPR 2000)*, Barcelona, Spain, 2000.

[Lav97] STÉPHANE LAVIROTTE und LOÏC POTTIER. Optical formula recognition. In *Proc. Int. Conference on Document Analysis and Recognition (ICDAR)*, Volume 1, Seiten 357–361, Ulm, 1997.

[Lav98] STÉPHANE LAVIROTTE und LOÏC POTTIER. Mathematical Formula Recognition using Graph Grammar. In *Electronic Imaging*, Seiten 44–52, San Jose, USA, 1998.

[Lee89] KAI-FU LEE. *Automatic Speech Recognition: The Development of the SPHINX System*. Kluwer, Dordrecht, The Netherlands, 1989.

[Lee90] K. LEE. Context Dependent Phonetic Hidden Markov Models for Speaker-Independent Continuous Speech Recognition. *Trans. on Acoustics, Speech and Signal Processing*, 38(4):599–609, 1990.

[Mak94] JOHN MAKHOUL, THAD STARNER, RICHARD SCHWARTZ und GEORGE CHOU. On-Line Cursive Handwriting Recognition Using Hidden Markov Models and Statistical Grammars. In *Proc. of the Human Language Technology Workshop*, Seiten 432–436, Plainsboro, NJ, March 1994.

[Man94] S. MANKE, M. FINKE und A. WAIBEL. Combining Bitmaps with Dynamic Writing Information for On-Line Handwriting Recognition. In *Proc. Int. Conference on Pattern Recognition (ICPR)*, Seiten 596–598, Jerusalem, 1994.

[Mes99] ANDRE MESSERSCHMIDT. Integration von Korrekturfunktionen in einen handschriftlichen Formeleditor. Diplomarbeit, Mercator University Duisburg, 1999.

[Mor97] D. MORI und H. BUNKE. Automatic interpretation and execution of manual corrections on text documents. In H. BUNKE und P. S. P. WANG, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 26, Seiten 679–702. World Scientific Publishing, 1997.

[Mü98] STEFAN MÜLLER, GERHARD RIGOLL, ANDREAS KOSMALA und DENIS MAZURENOK. Recognition of Hand-Drawn Pictograms Using HMMs with Rotating Feature Extraction. In *6th International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, Taejon, Korea, 1998.

[Mü99] STEFAN MÜLLER, GERHARD RIGOLL, ANDREAS KOSMALA und DENIS MAZURENOK. Combining Shape Matrices and HMMs for Hand-Drawn Pictogram

Recognition. In SEONG-WHAN LEE, editor, *Advances in Handwriting Recognition*, chapter 9, Seiten 519–528. World Scientific, 1999.

[Nat95] KRISHNA S. NATHAN, HOMAYOON S. M. BEIGI und JAYASHREE SUBRAHMONIA. Real-Time On-Line Unconstrained Handwriting Recognition Using Statistical Methods. In *Proc. IEEE Int. Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Volume 4, Seiten 2619–2622, Detroit, Michigan, 1995.

[Neu99] CHRISTOPH NEUKIRCHEN. *Integration neuronaler Vektorquantisierer in ein Hidden-Markov-Modell-basiertes System zur automatischen Spracherkennung*. Dissertation, Faculty of Electrical Engineering, Gerhard-Mercator-University Duisburg, 1999.

[Pfa69] J. PFALTZ und A. ROSENBERG. Web Grammars. In *First Int. Joint Conference on Joint Artificial Intelligence*, Seiten 609–619, Washington, 1969.

[Pra91] WILLIAM K. PRATT. *Digital Image Processing*. John Wiley & Sons, 1991.

[Rab85] L. R. RABINER, B. H. JUANG, S. E. LEVINSON und M. M. SONDHI. Recognition of Isolated Digits Using HMMs with Continuous Mixture Densities. *AT&T Tech. J.*, 64(6):1211–1233, 1985.

[Rab89] LAWRENCE R. RABINER. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc. of the IEEE*, 77(2):257–285, 1989.

[Ram97] J. Y. RAMEL, N. VINCENT und J. M. BRUN. Bezier curves as a tool to describe kinetic drawing. In *Proc. Int. Conference on Document Analysis and Recognition (ICDAR)*, Volume 2, Seiten 780–783, Ulm, 1997.

[Ren95a] STEVE RENALS und MIKE HOCHBERG. Decoder Technology for Connectionist Large Vocabulary Speech Recognition. Technical report, Department of Computer Science, University of Sheffield, 1995.

[Ren95b] STEVE RENALS und MIKE HOCHBERG. Efficient Search Using Posterior Phone Probability Estimates. In *Proc. IEEE Int. Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Seiten 596–599, Detroit, USA, 1995.

[Rig94] GERHARD RIGOLL. Maximum Mutual Information Neural Networks for Hybrid Connectionist-HMM Speech Recognition Systems. *IEEE Transactions on Speech and Audio Processing, Special Issue on Neural Networks for Speech*, 2(1):175–184, Januar 1994.

[Rig96a] GERHARD RIGOLL, ANDREAS KOSMALA, JÖRG ROTTLAND und CHRISTOPH NEUKIRCHEN. A Comparison between Continuous and Discrete Density Hidden Markov Models for Cursive Handwriting Recognition. In *Proc. Int. Conference on Pattern Recognition (ICPR)*, Volume 2, Seiten 205–209, Vienna, 1996.

[Rig96b] GERHARD RIGOLL, ANDREAS KOSMALA und MIKE SCHUSTER. A New Approach to Video Sequence Recognition Based on Statistical Methods. In *IEEE Int. Conference on Image Processing (ICIP)*, Seiten 839–842, Lausanne, September 1996.

[Rig96c] GERHARD RIGOLL, ANDREAS KOSMALA und MIKE SCHUSTER. High Performance Gesture Recognition Using Probabilistic Neural Networks and Hidden Markov Models. In *5th Intern. Workshop on time-varying Image Processing and Moving Image Recognition*, Seiten 1687–1689, Florence, September 1996.

[Rig97a] GERHARD RIGOLL und ANDREAS KOSMALA. New Improved Feature Extraction Methods for Real-Time High Performance Image Sequence Recognition. In *IEEE Int. Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Seiten 2901–2904, Munich, April 1997.

[Rig97b] GERHARD RIGOLL, ANDREAS KOSMALA und STEFAN EICKELER. High Performance Real-Time Gesture Recognition Using Hidden Markov Models. In *Gesture Workshop*, Seiten 69–80, Bielefeld, Germany, September 1997.

[Rig98a] GERHARD RIGOLL, STEFAN EICKELER, ANDREAS KOSMALA und STEFAN MÜLLER. Echtzeitfähige Gestikerkennung mit stochastischen Mustererkennungsverfahren. In *28. Jahrestagung der Gesellschaft für Informatik*, Magdeburg, Germany, September 1998.

[Rig98b] GERHARD RIGOLL und ANDREAS KOSMALA. A Systematic Comparison Between On-Line and Off-Line Methods for Signature Verification with Hidden Markov Models. In *International Conference on Pattern Recognition (ICPR)*, Brisbane, 1998.

[Rig98c] GERHARD RIGOLL, ANDREAS KOSMALA und DANIEL WILLETT. An Investigation of Context-Dependent and Hybrid Modeling Techniques for Very Large Vocabulary On-Line Cursive Handwriting Recognition. In *6th International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, Taejon, Korea, 1998.

[Rig98d] GERHARD RIGOLL, ANDREAS KOSMALA und DANIEL WILLETT. A New Hybrid Approach to Large Vocabulary Cursive Handwriting Recognition. In *International Conference on Pattern Recognition (ICPR)*, Brisbane, 1998.

[Rig99] GERHARD RIGOLL, ANDREAS KOSMALA und DANIEL WILLETT. A Systematic Comparison of Advanced Modeling Techniques for Very Large Vocabulary On-Line Cursive Handwriting Recognition. In SEONG-WHAN LEE, editor, *Advances in Handwriting Recognition*, chapter 2, Seiten 69–78. World Scientific, 1999.

[Rig00] GERHARD RIGOLL. Kommunikationssystem, Vorrichtung und Verfahren zum Bearbeiten eines zu erkennenden Musters, Vorrichtung und Verfahren zur Erkennung eines Musters. DE-Patentanmeldung 100 06 421.3, 1 2000.

[Rot00] JÖRG ROTTLAND. *Ein hybrider Ansatz zur automatischen Spracherkennung und Sprecheradaption für große Wortschätze*. Dissertation, Faculty of Electrical Engineering, Gerhard-Mercator-University Duisburg, Februar 2000.

[Sch95] MARKUS E. SCHENKEL. *Handwriting Recognition using Neural Networks and Hidden Markov Models*. Hartung-Gorre Verlag, Konstanz, 1995.

[ST95] E. G. SCHUKAT-TALAMAZZINI. *Automatische Spracherkennung – Grundlagen, statistische Modelle und effiziente Algorithmen*. Künstliche Intelligenz. Vieweg, Braunschweig, 1995.

[Sun97] CHANGMING SUN und DEYI SI. Skew and slant correction for document images using gradient direction. In *Proc. Int. Conference on Document Analysis and Recognition (ICDAR)*, Volume 1, Seiten 142–146, Ulm, 1997.

[VNG87] P. MERMELSTEIN V. N. GUPTA, M. LENNING. Integration of Acoustic Information in a Large Vocabulary Word Recognizer. In *Proc. IEEE Int. Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Seiten 697–700, Dallas, 1987.

[Wey96] PATRICK WEYEN. Aufbau eines hybriden NN-HMM Systems zur schreiber- und schreibstilunabhänigen Handschrifterkennung. Diplomarbeit, Faculty of Electrical Engineering - Computer Science, Gerhard-Mercator-University Duisburg, Januar 1996. in German.

[Wil98] DANIEL WILLETT, CHRISTOPH NEUKIRCHEN und GERHARD RIGOLL. DUCODE — der Stackdecoder. Technical report, Fachbereich Elektrotechnik, Gerhard-Mercator-Universität - Duisburg, 1998.

[Yan95] LIPING YANG. *Processing and Recognition of Handwriting in Multimedia Environments*. Dissertation, Technische Universiteit Delft, 1995.

[You92] S. J. YOUNG. The General Use of Tying in Phoneme-Based HMM Speech Recognisers. In *Proc. IEEE Int. Conference on Acoustics, Speech and Signal Processing(ICASSP)*, Volume 1, Seiten 569–572, San Francisco, USA, 1992.

[You93] S. J. YOUNG und P. C. WOODLAND. The Use of State Tying in Continuous Speech Recognition. In *Proc. Eurospeech, European Conference on Speech Communication and Technology*, Seiten 2203–2206, Berlin, 1993.

[You94] S. J. YOUNG, J. J. ODELL und P. C. WOODLAND. Tree-Based State Tying for High Accuracy Acoustic Modelling. In *Proc. of the Human Language Technology Workshop*, Seiten 307–312, Plainsboro, NJ, USA, März 1994.

[Zel94] ANDREAS ZELL. *Simulation Neuronaler Netze*. Kontakt und Studium. Eddison-Wesley, 1994. 624 Seiten.