Institut für Logik,
Komplexität und Deduktionssysteme
Lehrstuhl: Prof. Waibel

Erkennung von handgeschriebenen, mathematischen Formeln

Diplomarbeit
Mai 2000

Ausarbeitung : Roald Wolff
Betreuer : Stefan Jäger

Hiermit erkläre ich, daß ich die vorliegende Diplomarbeit selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Die verwendeten Literaturquellen sind im Literaturverzeichnis vollständig aufgeführt.

Karlsruhe, den 14. Juni 2000 .....R. Wolff.....

Zusammenfasssung

Im Mittelpunkt dieser Arbeit stand die Entwicklung eines Erkenners von handgeschriebenen, mathematischen Formeln, der sowohl schreiberunabhängig sein sollte als auch keine Restriktionen bei der Eingabe erfordern sollte. Der Zeichensatz des Systems umfaßt die Zeichen: A-Z, a-z, 0-9, ∞, +, -, *, =, (,) und die mathematischen Strukturen Summen- und Integralzeichen, Wurzel und Brüche.
Um eine Eingabe zu segmentieren, wurde der Ansatz eines Symbol-Hypothesen-Netzes ausgewählt. Dieser Ansatz zeichnet sich dadurch aus, daß zusammen mit selbsterarbeiteten Merkmalen auch die Ergebnisse eines Einzelzeichenerkennners für die Segmentierung herangezogen werden. Zusätzlich wird nicht nur eine beste Segmentierung berücksichtigt, sondern es werden alternative Segmentierungen zugelassen. Um die Zeichen zu erkennen, konnte auf den Einzelzeichenerkenner des NPen++ Projekts zugegriffen werden. Die Strukturanalyse sollte es ermöglichen, auch geschachtelte Ausdrücke von Wurzelzeichen oder Brüchen bzw. von Exponenten und Indices zu erkennen. Das Problem der Anordnung der Zeichen bei Brüchen, Wurzel-, Summen- und Integralzeichen wurde mit einem regelbasierten Ansatz gelöst, während bei der Bestimmung von Exponenten und Indices ein neuronales Netz zum Einsatz kam.
Als Fehlermaß zur Bestimmung der Erkennungsleistung des gesamten Systems wurde die Editierdistanz der Testformeln in LaTeX-Schreibweise verwendet. Auf dieser Testmenge, die 950 Formeln mit 13.382 Zeichen von 28 Schreibern umfaßt, erzielt das System eine Fehlerrate von 26.6%.
In dieser Arbeit wurde zum ersten Mal der Einsatz eines n-Gramm Sprachmodells für die Formelerkennung ausgetestet. Nach dem Aufbau einer Datenbasis wurden verschiedene n-Gramm Modelle der Ordnung 3 mit unterschiedlichen Äquivalenzklassen ausprobiert. Alle Hypothesen aus der N-Besten Liste des Erkenners wurden zusätzlich noch mit diesem Sprachmodell bewertet, so daß sich die Fehlerrate um 4% auf 22.6% reduzieren ließ.

Inhaltsverzeichnis

1 Einleitung **6**
1.1 Überblick 7

2 Segmentierung **9**
2.1 Weitere Ansätze 10
2.2 Symbol-Hypothesen-Netz 12
2.2.1 Restriktionen der Eingabe 12
2.2.2 Grundlegende Idee 13
2.2.3 Merkmale für die Segmentierung 15
2.2.4 Das neuronale Netz 17
2.2.5 Fehleranalyse 19

3 Vorverarbeitung **21**
3.1 Normalisierung 21
3.1.1 Nicht-lineare Interpolation fehlender Datenpunkte 22
3.1.2 Glättung 23
3.1.3 Neuabtastung 24
3.1.4 Aufsetzungsfehler 25
3.2 Merkmalsextraktion 26
3.2.1 Stiftzustand 27
3.2.2 Schreibrichtung 27
3.2.3 Krümmung 27
3.2.4 Erscheinungsbild 28
3.2.5 Linearität 28
3.2.6 Gewelltheit 29
3.2.7 Neigung 29
3.2.8 Kontext-Bitmaps 29

4 Einzelzeichenerkenner **31**
4.1 Hidden Markov Modelle 32
4.1.1 Definitionen 32
4.2 Evaluierungsproblem 33
4.3 Dekodierungsproblem 33

4.4 Optimierungsproblem 34
4.5 Emissionswahrscheinlichkeiten 34
4.6 MS-TDNN 35
4.6.1 Architektur des MS-TDNN 36
4.6.2 Training des MS-TDNN 36
4.7 Ergebnisse 37

5 Strukturanalyse **41**
5.1 Mathematische Operatoren 41
5.2 Exponenten und Indices 43
5.2.1 Weitere Ansätze 43
5.2.2 Neuronales Netz 44
5.3 Geschachtelte Exponenten/Indices 47

6 Evaluierung **49**
6.1 Fehlermaß 49
6.2 Ergebnisse 50
6.3 Gewichtung Einzelzeichenerkenner/ Segmentierer 52
6.4 Korrektur beim Gleichheitszeichen 54
6.5 N-Besten Liste 54

7 Sprachmodell **58**
7.1 Grammatiken 58
7.2 n-Gramm Modell 59
7.2.1 Äquivalenzklassenbildung 60
7.2.2 Interpolations- und Glättungsverfahren 61
7.2.3 Perplexität 62
7.3 Experimente 62
7.3.1 Datensammlung 63
7.3.2 Äquivalenzklassen 64
7.3.3 Ergebnisse 65

8 Datenbasis **67**

9 Ausblick **70**

10 Literaturverzeichnis **72**

Kapitel 1

Einleitung

Das Problem der automatischen Handschrifterkennung steht schon seit längerer Zeit im Mittelpunkt der Forschung. Besonders die Verbreitung von Palmtop Computers und Pen-Interfaces haben dieses Forschungsgebiet populär gemacht und einige kommerzielle Systeme hervorgebracht.
Obwohl die Erkennung von Wörtern und Sätzen bereits funktioniert, fehlt es an erforschten Methoden und Algorithmen für die Erkennung von mathematischen Formeln. Aber gerade in wissenschaftlichen Texten kommen häufig mathematische Ausdrücke vor, deren Eingabe über einen Stift wesentlich komfortabler als mit gängigen Methoden ist. Denn eine Eingabe über Tastatur und Maus ist umständlich und das Erlernen von Formatierungssprachen wie LaTeX dauert lange. Neben dieser Anwendungsmöglichkeit kann die automatische Erkennung von Formeln auch zu einer bequemeren Interaktion mit Algebrasystemen und zur automatischen Transkription von Dokumenten oder Vorlesungen genutzt werden.
Die Formelerkennung erweist sich als wesentlich schwieriger als die Erkennung von Wörtern und Text. Denn zum einen sind in mathematischen Formeln deutlich mehr verschiedene Zeichen enthalten, die auch noch in unterschiedlichen Größen vorkommen können, zum anderen sind die Zeichen nicht wie bei Texten nebeneinander angeordnet, sondern sie können auch über-, unter- und ineinander stehen. Damit gibt es bei der Formelerkennung 2 komplexe Probleme zu lösen:

1. Die Eingabe muß in Zeichen segmentiert werden und die einzelnen Zeichen müssen erkannt werden.

2. Aus der Anordnung der Zeichen zueinander muß eine mathematische Struktur bestimmt werden.

Der im Rahmen dieser Arbeit entwickelte Formelerkenner orientiert sich an dieser Problemteilung.

1.1 Überblick

In der Abbildung 1.1 ist der grundlegende Aufbau des Formelerkenners abgebildet. Nach der Eingabe der Formel mit einem Stift liegen die On-Line Daten vor, so daß mit dem Erkennungsprozeß begonnen werden kann. Zunächst muß das Problem der Segmentierung und der Erkennung der Zeichen gelöst werden. Dazu wird das in Kapitel 2 beschriebene Symbol-Hypothesen-Netz verwendet. Es zeichnet sich dadurch aus, daß die Segmentierung nicht nur alleine mit Hilfe von räumlichen Merkmalen sondern auch unter Verwendung des Einzelzeichenerkenners durchgeführt wird. Damit bei Eingaben, die sich nicht eindeutig segmentieren lassen, keine Fehlentscheidung gemacht wird, werden verschiedene, mit Wahrscheinlichkeiten versehene Segmentierungen zugelassen. In späteren Erkennungsschritten wird aus diesen Alternativen die beste Hypothese ausgewählt.
Nach der Segmentierung sind nur die einzelnen Zeichen aus der Eingabe bekannt, aber noch nicht in welcher Anordnung die Zeichen zueinander stehen. So müssen z.B. Zähler und Nenner bei Brüchen, die Grenzen eines Integrals und Exponenten und Indices bestimmt werden. Außerdem ist die Schreibweise von mathematischen Ausdrücken nicht immer strikt von links nach rechts, sondern es werden oft Klammern oder Summengrenzen am Ende nachträglich in den Ausdruck eingefügt. Diese Probleme werden in der Strukturanalyse (Kapitel 5) behandelt.
Einerseits sollte ein Formelerkenner in der Lage sein, nur korrekte mathematische Formeln zu erkennen, andererseits sollte er eine Eingabe nicht abweisen, falls der Benutzer eine syntaktisch falsche Formel eingibt. Um diese beiden gegensätzlichen Forderungen zu erfüllen, wurde erstmals ein n-Gramm Sprachmodell für die Formelerkennung realisiert. Alle Hypothesen aus der N-Besten Liste werden durch das Sprachmodell neu bewertet, so daß sich die Reihenfolge innerhalb der Liste verändern kann. Die Experimente mit dem Sprachmodell sind in Kapitel 7 beschrieben.
Um die Leistungsfähigkeit des Systems zu demonstrieren, wurden ausführliche Versuche auf Testdaten durchgeführt. Die unterschiedlichen Auswirkungen der Einstellung verschiedener Parameter auf die Fehlerrate des Systems sind in Kapitel 6 dargestellt.
Am Ende dieser Arbeit ist eine Zusamenfassung der verwendeten Datenbasis und ein Ausblick auf weitere Verbesserungsmöglickeiten aufgeführt.

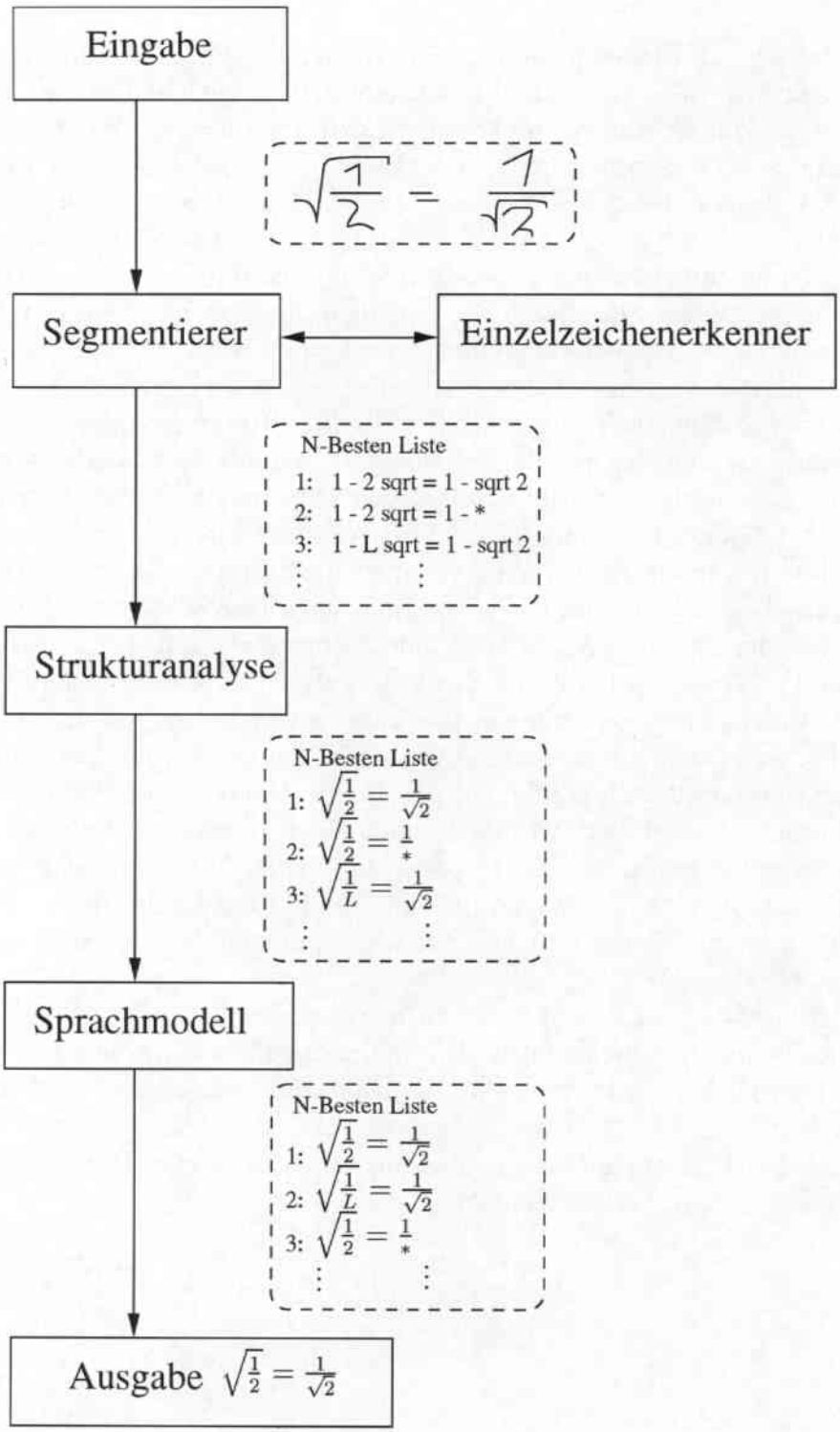

Abbildung 1.1: Gesamtüberblick über das System

Kapitel 2

Segmentierung

Bevor die Eingabe des Benutzers, die Folge der Datenpunkte (x(t), y(t), p(t)), dem Einzelzeichenerkenner zugeführt werden können, müssen noch einige Vorüberlegungen angestellt werden. Sobald sich in der Eingabe mehrere Zeichen befinden, reicht es nicht aus, nur mit dem Einzelzeichenerkenner zu arbeiten. Zwar ist dieser in der Lage, einzelne Zeichen richtig zu klassifizieren, aber er kann nicht entscheiden, ob in der Eingabe mehrere Zeichen enthalten sind und an welcher Stelle eine Trennung stattfindet. Deshalb ist es notwendig, den Einzelzeichenerkenner zu unterstützen, indem dafür gesorgt wird, daß ihm immer nur die Datenpunkte eines Zeichens zugeführt werden. Diese Forderung ist damit gleichbedeutend, daß eine Segmentierung der Eingabe in die einzelnen Zeichen gefunden wird. Dieses Problem ist nicht immer eindeutig zu lösen, wie Abbildung 2.1 verdeutlicht. Der erste Ausdruck kann sowohl als 'K4' als auch als $1 < 4$ interpretiert werden; der zweite als '13+4' als auch als 'B+4'.
Um diese Aufgabe der Segmentierung zu lösen, wird in dieser Arbeit ein sogenannter Soft-Decision Ansatz [29] verwendet. Dieser Ansatz zeichnet sich dadurch aus, daß mehrere mögliche Segmentierungen der Eingabe berücksichtigt werden; d.h., sobald sich die Eingabe nicht mehr eindeutig segmentieren läßt, werden alternative Segmentierungen zugelassen und jede Alternative mit einer bestimmten Wahrscheinlichkeit versehen. Damit liefert dieser Soft-Decision Ansatz als Ergebnis eine Liste der besten Segmentierungen der Eingabe. Bevor dieser Ansatz detailliert beschrieben wird, werden zunächst noch alternative Lösungen für das Segmentierungsproblem kurz vorgestellt.

Abbildung 2.1: Beispiele für zweideutige Segementierungsmöglichkeiten

2.1 Weitere Ansätze

Die Segmentierung einer Eingabe erfolgt häufig auf Basis der Strokes der Eingabe. Dabei ist ein Stroke definiert als der Schriftzug zwischen dem Herabsetzen des Stiftes auf die Schreibfläche und dem folgenden Abheben des Stiftes. Wie in Abbildung 2.2 deutlich wird, reduziert sich damit das Segmentierungsproblem auf die Bestimmung, welche Strokes zu einem gemeinsamen Zeichen gehören.
Die einfachste Lösung zum Auffinden einer Partitionierung der Eingabe besteht darin, Merkmale zwischen den Strokes zu berechnen und abhängig von dessen Ergebnissen regelbasiert zu bestimmen, welche Strokes zusammen ein Zeichen bilden. Ein typisches Merkmal, das an dieser Stelle eingesetzt wird, ist z.B. die minimale Entfernung zwischen 2 Strokes. Bei diesem Verfahren bestehen aber 2 Probleme: zum einen wird die Segmentierung unabänderlich festgelegt und ein Fehler in diesem Schritt beeinflußt alle folgenden Schritte und zum anderen wird ein regelbasierter Entscheider nicht alle möglichen Eingaben zufriedenstellend abdecken. Da es gerade bei der Formelerkennung sehr viele unterschiedliche Anordnungen der Strokes zueinander gibt, ist dieser Ansatz hier nicht sinnvoll.
In [19] wurde ein Segmentierungsvorschlag extra für die Formelerkennung vor-

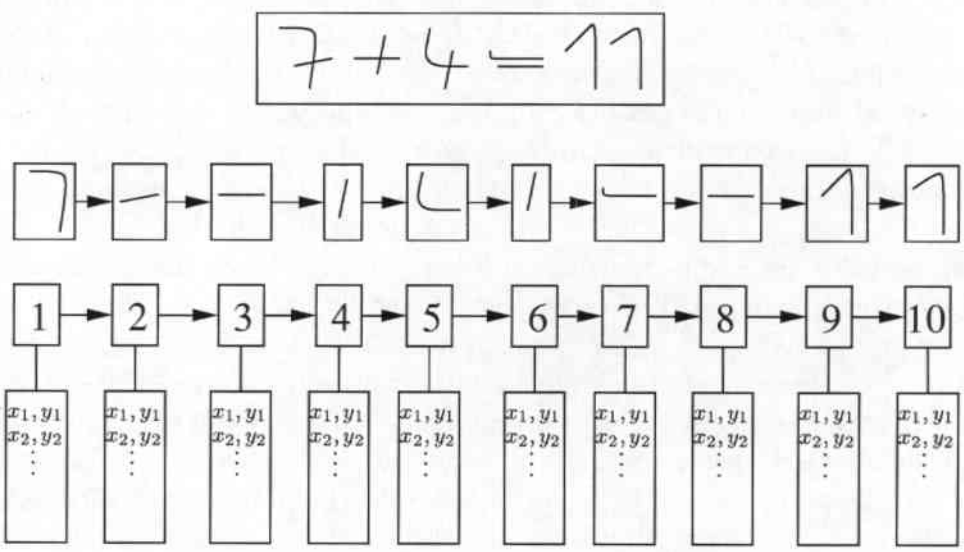

Abbildung 2.2: Schriftzug der Eingabe, Strokesequenz und zugehörige Datenpunkte

gestellt, der sich auch mit der Zusammenfassung von Strokes zu einem Zeichen beschäftigt. Die Grundlage dieses Ansatzes bildet ein minimal spannender Baum, der wie folgt auf der Strokemenge bestimmt wird. Durch die Berechnung der Entfernungen zwischen den Mittelpunkten der umgebenden Boxen jedes Strokes

wird die Eingabe in einen zusammenhängenden, ungerichteten, bewerteten Graphen umgewandelt, d.h. die Knoten im Graph entsprechen den Strokes und die Kanten den Entfernungen zwischen den Mittelpunkten (siehe Abbilung 2.3). Auf dem Graph kann mit Hilfe des Algorithmus von Kruskal ein minimal spannender Baum ermittelt werden. Die Anordnung in eine Baumstruktur sorgt dafür, daß Strokes, die zu einem Zeichen gehören, auch im Baum benachbarte Knoten darstellen. Unter der Vorraussetzung, daß ein Zeichen aus maximal vier aufeinanderfolgenden Strokes besteht, kann die Suche nach möglichen Zeichen innerhalb dieses Baumes auf Teilbäume beschränkt werden. Falls eine Verbindung zwischen 2 Strokes besteht, so wird geprüft, ob sie zusammen ein Zeichen bilden, und im positiven Fall wird eine weitere Untersuchung mit den Nachbarknoten angestoßen. Falls dagegen zwischen 2 Strokes keine Verbindung besteht und sie im minimal spannenden Baum eine größere Entfernung als vier haben, so werden die beiden Strokes entweder als einzelne Zeichen partitioniert oder mit anderen Strokes zusammengesetzt, aber niemals zu einem Zeichen zusammengesetzt. Der durch diesen Ansatz berechnete minimal spannende Baum repräsentiert die Eingabe zwar gut (Strokes eines Zeichens sind benachbarte Knoten im Baum), aber es entsteht ein Problem, wenn der Ansatz auf Wurzelzeichen erweitert werden soll. Denn bei einem Wurzelzeichen liegt der Mittelpunkt der umgebenden Box mitten im Wurzelterm. Im ungünstigen Fall kann es dann passieren, daß zwei zusammengehörende Strokes im minimal spannenden Baum nicht mehr durch eine Kante verbunden sind und somit das Verfahren versagt. Damit wird deutlich, daß ein einziges Merkmal nicht ausreichen kann, um eine korrekte Segmentierung in allen möglichen Fällen durchzuführen.

Die eleganteste Möglichkeit der Segmentierung liegt in der Verwendung von Hid-

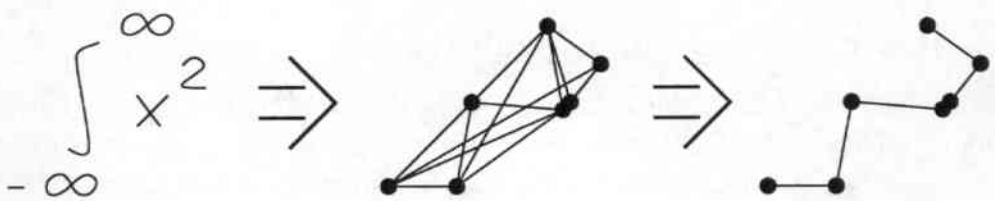

Abbildung 2.3: Segmentierung mit Hilfe von minimal spannenden Bäumen: links Eingabe, Mitte zugehöriger Graph, rechts minimal spannender Baum

den-Markov-Models (HMM), wie sie in Kapitel 4.1 beschrieben sind. HMMs werden in der Sprach- und Schrifterkennung erfolgreich zur Wort- und Satzerkennung eingesetzt, da sie die Fähigkeit besitzen, zeitliche, variante Abläufe zu erkennen und gleichzeitig zu segmentieren. In [15] wird ein Formelerkenner vorgestellt, der mit diesem Ansatz arbeitet.

2.2 Symbol-Hypothesen-Netz

Auch der in dieser Arbeit verwendete Ansatz zur Segmentierung versucht, eine Einteilung der Strokes in mögliche Zeichen zu finden. Ein wichtiger Unterscheidung zu den im vorangegangen Abschnitt vorgestellten Ansätzen ist, daß hier die Entscheidung nicht nur auf Grund von räumlichen Merkmalen zwischen den Strokes getroffen wird, sondern daß in die Entscheidung auch das Ergebnis des Einzelzeichenerkenners einfließt. Zusätzlich wird nicht nur die beste Segmentierung gesucht, sondern es werden auch alternative Segmentierungen zugelassen. Als Ergebnis der Segmentierung wird eine Gitterstruktur (Symbol-Hypothesen-Netz) erzeugt, aus dem die verschiedenen Hypothesen über die Segmentierung ermittelt werden können. Dabei wird jede Hypothese mit einer Wahrscheinlichkeit versehen, so daß man eine N-Besten Liste erhält. Alle Alternativen werden in den weiteren Schritten untersucht und dort entweder auf- oder abbewertet, so daß eine Fehlentscheidung bei der Segmentierung durch spätere Bewertungen ausgeglichen werden kann.
Dieser Ansatz zur Segmentierung mit Hilfe von Symbol-Hypothesen-Netze wurde in [27, 28] vorgestellt. Um aber eine bessere Leistungsfähigkeit zu erhalten, wurden einige Änderungen vorgenommen: um die Zusammengehörigkeit zweier Strokes zu prüfen, wurden eigene Merkmale ermittelt (siehe Abschnitt 2.2.3). Diese Merkmale werden aber im Gegensatz zu [27, 28] nicht mit einer festen Bewertung ausgewertet, sondern es wird ein neuronales Netz benutzt. Durch dessen Generalisierungfähigkeit soll gewährleistet werden, daß alle möglichen Anordnungen in der Eingabe korrekt segmentiert werden. Außerdem wird auf eine vorgestufte Klassifizierung der Strokes in die drei Klassen (einfach, mittel, komplex), in Abhängigkeit davon wie gut sich der Stroke als Linie beschreiben läßt, verzichtet.

2.2.1 Restriktionen der Eingabe

Die Suche nach einer Segmentierung ohne jegliche Beschränkungen ist eine schwierige Aufgabe. Denn jeder Stroke aus der Eingabe kann mit jedem anderen Stroke möglicherweise ein Zeichen bilden, so daß, falls die Eingabe aus N Strokes besteht, $2^N - 1$ verschiedene Zeichen gebildet werden können. Aus dieser großen Anzahl von Möglichkeiten muß dann die Lösung gefunden werden. Dieses exponentielle Wachsen des Suchraums zwingt dazu, gewisse Beschränkungen einzuführen, jedoch ohne daß der Benutzer auf eine natürliche Schreibweise von mathematischen Formeln verzichten muß. Deshalb setzen alle Formelerkenner die Restriktion voraus, daß ein Zeichen zunächst zu Ende geschrieben werden muß, bevor ein neues Zeichen begonnen wird. Obwohl diese Einschränkung sinnvoll ist, wird sie bei bestimmten Schreibweisen des Wurzelzeichens verletzt. Denn viele Benutzer beenden erst das Wurzelzeichen, nachdem sie den Term in der Wurzel geschrieben haben und dann wissen, wie lang das Wurzelzeichen zu sein hat. Trotzdem wird auf diese Restriktion nicht verzichtet, auch wenn solche Eingaben damit nicht

mehr möglich sind. Die zweite wichtige Restriktion des Suchraums beschränkt die maximale Anzahl der Strokes pro Zeichen. Untersuchungen auf der Datenbasis haben gezeigt, daß die Zeichen maximal aus 4 Strokes aufgebaut sind. Das bedeutet, daß für das Auffinden einer Segmentierung immer nur vier aufeinanderfolgende Strokes untersucht werden müssen. Die Größe des Suchraums wächst durch das Zufügen der beiden Restriktionen nun linear mit N und erlaubt eine effektive Suche.

2.2.2 Grundlegende Idee

Durch das Einführen der Restriktionen sind bei einer Eingabe von N Strokes insgesamt $4 * N - 6$ verschiedene Strokegruppen möglich ($N \geq 3$). Dabei soll der Begriff Strokegruppe als Zusammenfassung von Strokes zu einem möglichen Zeichen verstanden werden. Diese Anzahl der Möglichkeiten soll durch den Einsatz eines Neuronalen Netzes noch weiter verringert werden, indem z.B. Strokes, die weit voneinander entfernt sind, nicht als mögliche Strokegruppe untersucht werden. In einem ersten Schritt werden zwischen dem Stroke m und dem Stroke m+g (g=1,2,3) Merkmale berechnet und als Eingabe für das neuronale Netz benutzt. Diese Merkmale, die im einzelnen im nächsten Abschnitt vorgestellt werden, sollen natürlich möglichst charakteristisch dafür sein, ob zwei Strokes zu einem Zeichen oder zu unterschiedlichen Zeichen gehören. Die Ausgabe des neuronalen Netzes wird interpretiert als Maß für die Wahrscheinlichkeit für die beiden Zustände: die beiden Strokes gehören zu einem Zeichen zusammen oder nicht. Als Ergebnis liefert dieser Schritt somit eine Bewertung $S_{m,g}$ (g=0,1,2,3) von jeder der $4 * N - 6$ Strokegruppen, die in eine Gitterstruktur eingetragen werden können (siehe Abbildung 2.4). Aus dieser Gitterstruktur kann nun eine Hypothese über die Segmentierung der gesamten Eingabe bestimmt werden, indem ein Pfad innerhalb dieses Gitters vom Anfang- bis zum Endknoten verfolgt wird. Die Bewertung einer solchen Hypothese ergibt sich durch Aufsummieren der logarithmierten Bewertungen der einzelnen Strokegruppen des Pfads.
Beim Betrachten der Abbildung 2.4 fällt auf, daß sehr viele Hypothesen erzeugt werden, deren Bewertung sehr gering sein sollte. Um die Anzahl der Hypothesen zu verringern, werden 2 Schwellwerte S_o, S_u eingesetzt, die dafür sorgen, daß keine sehr unwahrscheinlichen Strokegruppen in eine Hypothese aufgenommen werden. Falls für die Bewertung einer Strokegruppe $S_{m,g} > S_o$ gilt, dann wird für den Stroke m nur diese Strokegruppe als Möglichkeit betrachtet. Falls auf der anderen Seite $S_{m,g} < S_u$ gilt, so wird diese Strokegruppe als unmöglich angesehen und wird in keiner Hypothese erscheinen. In der Praxis hat es sich bewährt, die Werte auf $S_o = 0.99$ und auf $S_u = 0.01$ zu setzen.
Auf diese Weise wird die Anzahl der Hypothesen über die Segmentierung auf ein handliche Menge reduziert. Auf diese reduzierte Menge von Strokegruppen wird nun ein Einzelzeichenerkenner (siehe Kapitel 4) angesetzt. Dieser liefert mit einer bestimmten Wahrscheinlichkeit versehen die besten Erkennungsergebnisse

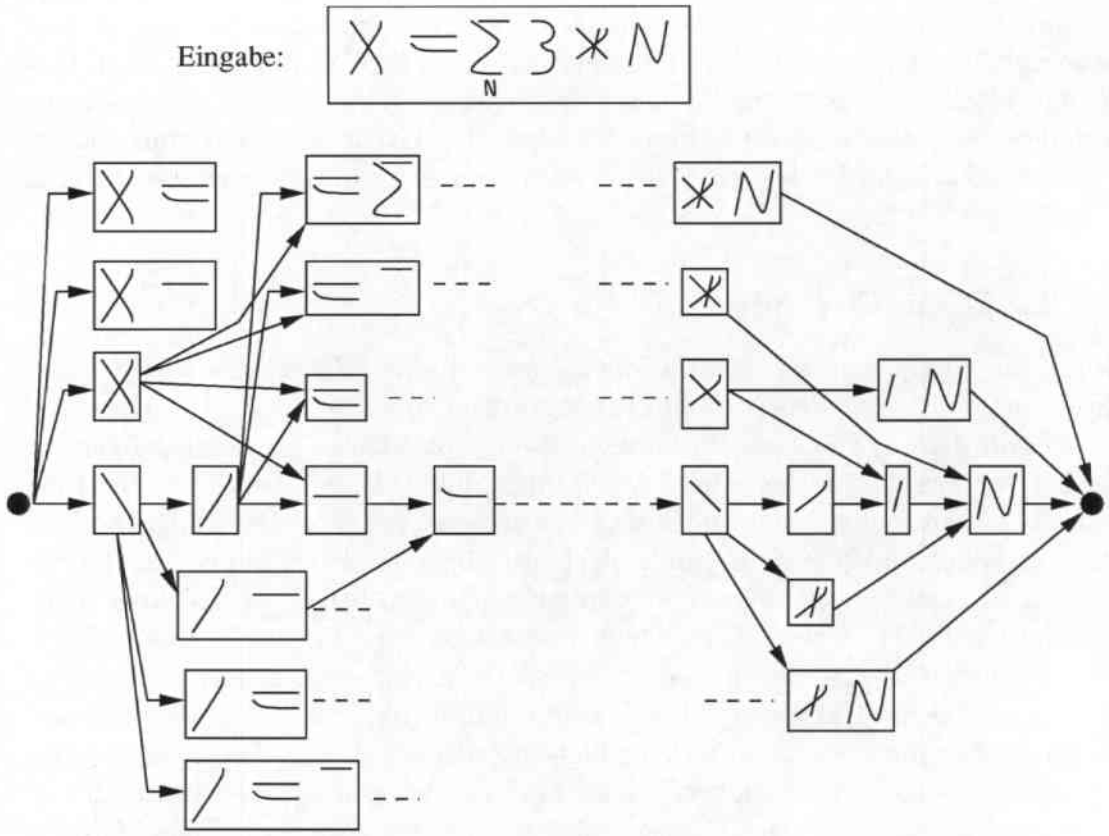

Abbildung 2.4: Anordnung der bewerteten Strokegruppen in eine Gitterstruktur

zurück. Falls mehrere Strokes fälschlicher Weise zu einer Strokegruppe zusammengefaßt worden sind, so wird der Einzelzeichenerkenner dies mit schlechten Erkennungsergebnissen quittieren. Dagegen werden für korrekte Strokegruppen auch hohe Erkennungsergebnisse zurückgeliefert. Damit läßt sich nun die Bewertung für jede Strokegruppe wie folgt berechnen:

$$P(m,g) = \delta * E(m,g) + (1-\delta) * S_{m,g}, \tag{2.1}$$

wobei die Strokegruppe aus den Strokes m bis m+g (g=0,1,2,3) besteht, E(m,g) das Ergebnis des Einzelzeichenerkenners ist, $S_{m,g}$ die Bewertung der Zusammengehörigkeit der Strokegruppe ist und $\delta \in [0,1]$ ein Gewichtsfaktor zwischen diesen Werten ist. Die Auswirkung des Gewichtsfaktors wird bei den Ergebnissen des Systems in Kapitel 6 beschrieben.
Wenn die Strokegruppen wieder in eine Gitterstruktur eingetragen werden, dann erhält man ein Symbol-Hypothesen-Netz, wie es in Abbildung 2.5 dargestellt ist. Dabei sind neben jeder Strokegruppe die Ergebnisse des Einzelzeichenerkenners abgebildet, wobei von oben nach unten die Wahrscheinlichkeit des Erkennungser-

gebnisses abnimmt. Die resultierenden Hypothesen über die Eingabe lassen sich wiederum durch Verfolgen eines Pfads vom Anfangs- zum Endknoten im Symbol-Hypothesen-Netz ermitteln. Auch hier ergibt sich die Bewertung einer Hypothese durch Aufsummieren der einzelnen logarithmierten Bewertungen $P(m, g)$ der Strokegruppen.

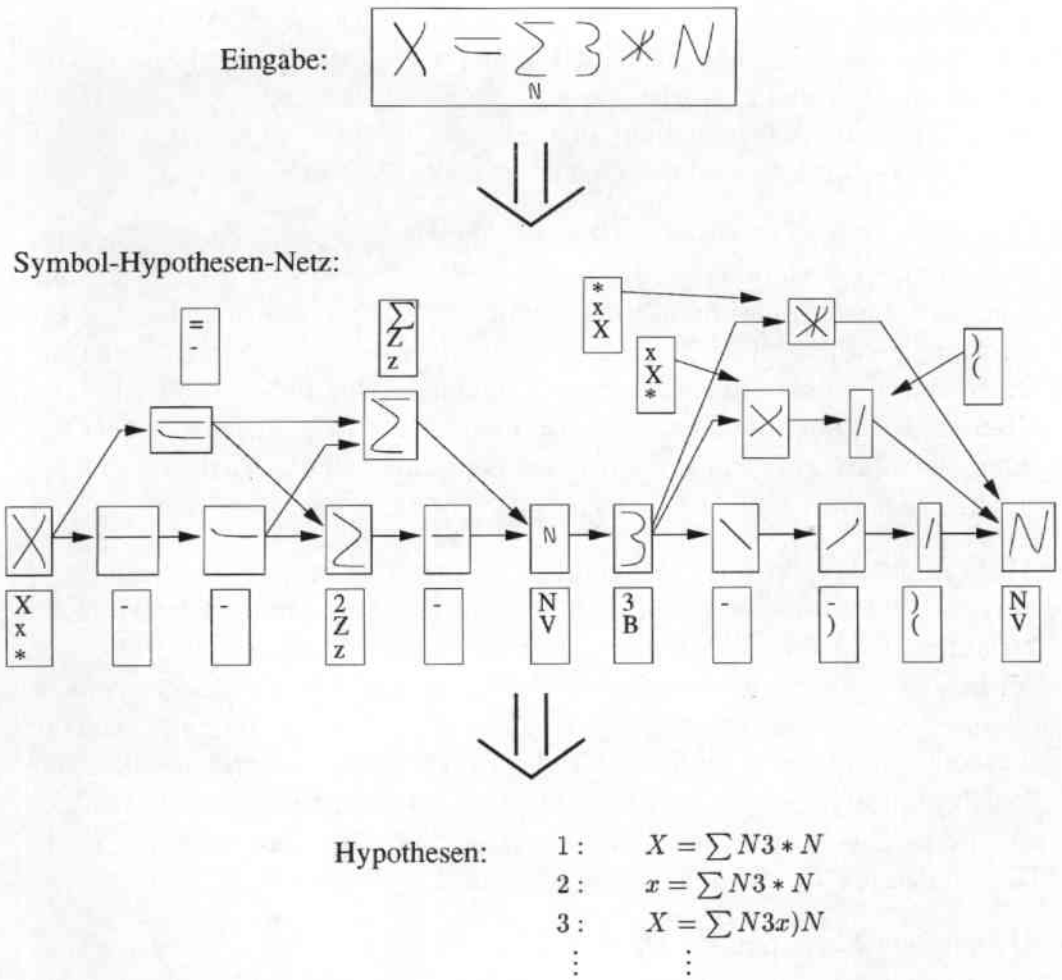

Abbildung 2.5: Segmentierung einer Eingabe durch ein Symbol-Hypothesen-Netz

2.2.3 Merkmale für die Segmentierung

In diesem Abschnitt werden die Merkmale vorgestellt, die zwischen den verschiedenen Strokes berechnet werden und als Eingabe für das neuronale Netz dienen. Es ist klar, daß ein einziges Merkmal nicht ausreichen kann, um die verschiedenen Anordnungsmöglichkeiten innerhalb einer Formel abzudecken. Stattdessen

werden mehrere Merkmale verwendet, wobei jedes Merkmal in bestimmten Situationen charakterisierend ist.

- Überlappung in X-Richtung: Dieses Merkmal soll Aufschluß darüber geben, inwieweit sich die beiden umgebenden Boxen der Zeichen in X-Richtung überschneiden. Eine Normierung wird mit der Summe der Breiten der umgebenden Boxen durchgeführt, so daß der Wert dieses Parameters immer im Interval [0,1] liegt. Der Parameter hat z.B. einen hohen Wert für Strokes, die zu einem Zeichen gehören (beispielsweise beim Zeichen 7). Falls die Strokes nicht zu dem selben Zeichen zusammengehören, so ist der Wert 0 für Zeichen, die nebeneinander in Reihe geschrieben sind, und 1 bei den mathematischen Konstruktionen wie Wurzel und Bruch.

- Überlappung in Y-Richtung: Genau wie beim vorhergehenden Merkmal wird hier die Überlappung der umgebenden Boxen in Y-Richtung bestimmt. Hier wird die Normierung mit der Summe der Höhen der umgebenden Boxen durchgeführt. Diese Merkmal soll die Segmentierung von Zeichen unterstützen, die übereinander angeordnet sind wie bei Brüchen oder bei den Grenzen von Integralen. In solchen Fällen ist der Wert des Merkmals fast immer 0, während Zeichen, die nebeneinander in Reihe geschrieben werden, einen hohen Wert nahe 1 erzeugen.

- Abstand der Trajektorien: Eine weitere wichtige Eigenschaft zur Segmentierung ist, ob sich die Trajektorien berühren bzw. ob sie nahe nebeneinander verlaufen. Deshalb wird die minimale Entfernung zwischen den Punkten der beiden Trajektorien berechnet. Für Strokes innerhalb eines Zeichens ist dieser Abstand meistens wesentlich geringer als für Strokes unterschiedlicher Zeichen. Da die berechnete Größe des Abstands aber von der aktuellen Schriftgröße abhängt, ist es notwendig, eine Normierung durchzuführen. Dazu wird der berechnete, minimale Abstand durch das Maximum der beiden Diagonalen der umgebenden Boxen dividiert.

- Abstand der Mittelpunkte: Da sich bei enger Schreibweise Strokes verschiedener Zeichen berühren können, reicht das obige Merkmal nicht aus. Deshalb wird ein zweites Abstandsmerkmal, der Abstand zwischen den Mittelpunkten der umgebenden Boxen, bestimmt. Auch hier wird eine Normierung mit dem Maximum der Diagonalen vorgenommen.

- XY-Verhältnis: Dieses Merkmal soll Auskunft über die Eigenschaften der umgebenden Box geben und wird für beide Strokes separat berechnet. Es wird berechnet, wie sich die Länge in X-Richtung der umgebenden Box zu der Länge in Y-Richtung der umgebenden Box verhält. Für einfache Strokes wie z.B. horizontale und vertikale Striche wird dieses Merkmal entweder sehr groß oder sehr klein. Dagegen werden komplexere Strokes, die sich nicht

durch eine Linie beschrieben lassen, sondern mehrere Liniensegmente enthalten, eher einen Wert um 1 erhalten. Damit im ungünstigen Fall die Werte des Merkmals nicht zu groß werden, wird ein maximaler Wert festgelegt. Für die Segmentierung ist die Beobachtung wichtig, daß ein Zeichen nie aus zwei komplexen Strokes besteht sondern eher aus mehreren einfachen Strokes aufgebaut ist.

- Diagonalverhältnis: Dieses Merkmal beschreibt das Verhältnis zwischen den beiden Diagonalen der umgebenden Boxen. Damit soll verhindert werden, daß Strokes verschiedener Größe zusammengefaßt werden. Genauso wie bei dem vorherigen Merkmal wird dieser Wert auch nach oben begrenzt.

- Parallelität: Da häufig Fehler bei der Segmentierung des Gleichheitszeichens und des H erfolgen, wurde dieses Merkmal extra eingeführt, um deren korrekte Segmentierung zu unterstützen. Das Problem besteht darin, daß die beiden Strokes z.B. beim Gleichheitszeichen in einem großen räumlichen Abstand zueinander stehen, und somit das neuronale Netz deren Zusammengehörigkeit niedrig bewertet. Mit Hilfe einer Heuristik wird nun geprüft, ob es sich um einfache Strokes handelt, d.h. beschreibbar durch eine Gerade, und ob sie parallel zueinander verlaufen.

2.2.4 Das neuronale Netz

Nachdem die Merkmale zwischen zwei Strokes berechnet worden sind, werden sie als Eingabe einem neuronalen Netz zugeführt. Die Aufgabe des neuronalen Netzes besteht darin, aus diesen Merkmalen ein Maß für die Wahrscheinlichkeit der beiden Klassen - Strokes gehören zusammen/ nicht zusammen - zu bestimmen. Diese Bewertung wurde in der Formel 2.1 als $S_{m,g} \in [0,1]$ bezeichnet. Optimalerweise sollte das Netz bei unklaren Situationen beide Ereignisse gleich hoch bewerten, so daß sich für $S_{m,g}$ ein Wert nahe bei 0.5 ergibt. Damit wird keine der beiden Möglichkeit ausgeschlossen bzw. überbewertet und die letztendliche Entscheidung dem Einzelzeichenerkenner überlassen.
Um das neuronale Netz zu trainieren, wurden 1655 Beispiele aus den gesammelten Daten extrahiert. Diese Beispiele umfassen alle möglichen Anordnungen in einer Formel und sind im Detail in der Tabelle 2.1 dargestellt.

Mit den 1655 Beispielen aus der Trainingsmenge wird das neuronale Netz mit dem Backtracking-Algorithmus trainiert, um anschließend dessen Generalisierungsfähigkeit auf der Testmenge zu evaluieren. Das Netz ist in 3 Schichten (Eingabeschicht, verborgene Schicht und Ausgabeschicht) aufgebaut, wobei die verborgene Schicht aus 20 Neuronen besteht. Die Anzahl der Fehler nach 10.000 Trainingsiterationen ist in der Tabelle 2.2 dargestellt. Dabei wird eine Klassifizierung als Fehler gewertet, wenn die Bewertung für die falsche Klasse (z.B. Strokes gehören zusammen) höher ist als für die richtige Klasse (Strokes gehören

	Anzahl	Trainingsmenge	Testmenge
Strokes eines Zeichens: (A..Z,a..z,0-9,+,-,*,$\sum$)	603	422	181
Strokes verschiedener Zeichen:			
-Brüche	290	160	130
-Wurzeln	212	140	72
-Grenzen bei Summen- und Integralzeichen	121	40	81
-Zeichen in einer Linie	383	274	109
-Exponenten und Indices	46	20	26
Summe:	1655	1056	599

Tabelle 2.1: Aufteilung der Trainings- und Testmenge

zu unterschiedlichen Zeichen). Auf diese Weise werden auch Klassifizierungen als Fehler in die Tabelle aufgenommen, wenn die Bewertungen für die richtige Klasse zwar niedriger als die falsche Klasse aber dennoch hoch ist. In so einem Fall muß, wie voran erläutert, nicht unbedingt eine Fehlsegmentierung entstehen, da der Einzelzeichenerkenner das Gesamtergebnis noch korrigieren kann. Wie aus der

	Trainingsmenge	Testmenge
Strokes eines Zeichens: (A..Z,a..z,0-9,+,-,*,$\sum$)	9	8
Strokes verschiedener Zeichen:		
-Brüche	0	0
-Wurzeln	0	1
-Grenzen bei Summen- und Integralzeichen	0	1
-Zeichen in einer Linie	0	0
-Exponenten und Indices	0	1
Summe:	9	11

Tabelle 2.2: Anzahl der Fehler bei der Segmentierung auf der Trainings- und Testmenge

Tabelle ersichtlich wird, werden auf der Trainingsmenge 9 und auf der Testmenge 11 Fehler gemacht. Dies entpricht einer Fehlerrate von 0.8% bzw. 1.8%, so daß deutlich wird, daß der gewählte Ansatz zur Segmentierung der Eingabe gut funktioniert. Dies gilt insbesondere wenn man die produzierten Fehler genauer

analysiert, wie es im nächsten Abschnitt geschehen soll.

2.2.5 Fehleranalyse

Beim Untersuchen der Fehler des neuronalen Netzes fällt auf, daß ein paar typische Fehlerarten dominieren. So werden die meisten Fehler dabei gemacht, daß zwei Strokes eines Zeichens irrtümlicherweise nicht zusammengefaßt werden. Besonders schwierig ist die Zusammenfaßung zweier Strokes, wenn sie weit voneinander entfernt geschrieben werden, wie es z.B. beim Gleichheitszeichen oder beim 'H' der Fall ist. Von den 9 Fehlern auf der Trainingsmenge entfallen 5 Fehler auf solche Fälle; bei der Testmenge sind es 2 Fehler.
Weiterhin werden in der Tabelle, wie oben erläutert wurde, auch Fehler aufgeführt, die sich später evtl. durch den Einzelzeichenerkenner korrigieren lassen. Zwar läßt sich nicht exakt bestimmen, für welche Fehler dies der Fall ist, da dies auch von dem Parameter δ in der Formel 2.1 abhängt, aber wenn die Bewertungen beider Ereignisse ungefähr gleich hoch sind, dann stehen die Chancen für eine Korrektur durch den Einzelzeichenerkenner gut. Eine solche Situation kommt bei den Fehlern der Trainingsmenge zweimal und bei den Fehlern der Testmenge sogar fünfmal vor. Wenn man davon ausgeht, daß der Einzelzeichenerkenner in diesen Fällen für eine korrekte Segmentierung sorgt, dann werden auf der Trainingsmenge nur noch 7 und auf der Testmenge 6 Segmentierungsfehler produziert. Das entspricht einer Fehlerrate von 0.6% auf der Trainingsmenge bzw. von 1.0% auf der Testmenge.

Um die hohe Anzahl der Fehlsegmentierungen bei den Gleichheitszeichen zu minimieren, wird zusätzlich ein Korrekturschritt am Ende der Hypothesenbildung durchgeführt. Falls die beiden Strokes eines Gleichheitszeichens getrennt werden, so erscheinen sie hintereinander als zwei Minuszeichen in der Hypothese. Im Korrekturschritt werden die Hypothesen nach solchen Anordnungen der Minuszeichen durchsucht. Wenn eine solche Anordnung auftritt, wird zusätzlich geprüft, ob die räumliche Beziehung der Strokes einem Gleichheitszeichen entspricht und ob nicht noch andere Zeichen involviert sind. Falls keine dieser Bedingungen verletzt ist, werden die beiden Minuszeichen in der Hypothese durch ein Gleichheitszeichen ersetzt.
Insgesamt läßt sich damit zusammenfassend sagen, daß dieser Weiche Entscheidungsansatz durchaus erfolgreich ist und bei dem Problem der Formelerkennung berechtigterweise eingesetzt wird. Wie die Beispiele in der Abbildung 2.6 zeigen, werden sogar Zeichen richtig segmentiert, die sich räumlich überlappen und deren korrekte Segmentierung somit schwer ist.

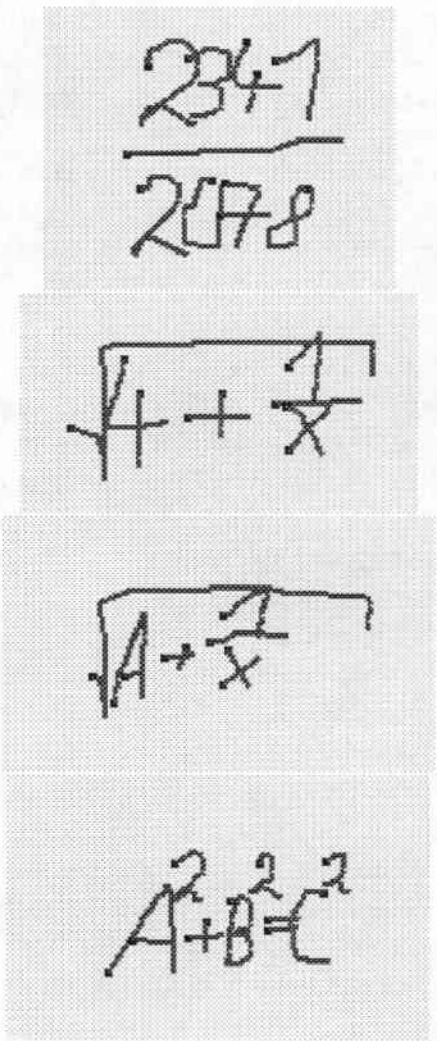

Abbildung 2.6: Beispiel für korrekte Segmentierungen

Kapitel 3

Vorverarbeitung

Die Vorverarbeitung ist in gängigen Texterkennungssystemen die erste Komponente im Erkennungsprozeß und läßt sich in zwei Schritte unterteilen: Normalisierung und Merkmalsextraktion. Die Stifttrajektorie (repräsentiert durch die zeitlich geordnete Folge der Datenpunkte $(x(0), y(0), p(0)), \ldots, (x(N), y(N), p(N))$) ist, in der Form wie sie von der Aufnahmehardware geliefert wird, noch geprägt von schreiberabhängigen Eigenschaften. So schreiben verschiedene Benutzer die Zeichen in unterschiedlicher Größe und Geschwindigkeit. Zusätzlich sorgen Aufsetzungsfehler des Stiftes auf die Oberfläche und Zittern in der Handschrift für eine zu große Varianz in dem Eingabemuster.
Durch die Normalisierung soll das Eingabesignal so transformiert werden, daß diese schreiberabhängigen Eigenschaften aus den Datenpunkten entfernt werden und die schreiberabhängige Varianz in den Eingabemustern möglichst gering ist. Bei der Merkmalsextraktion werden aus den Datenpunkten geeignete Merkmale berechnet, die im anschliessenden Schritt, der Einzelzeichenerkennung, das Zeichen repräsentieren werden. Eine schlechte Wahl der Merkmale führt somit unweigerlich zu schlechten Erkennungsergebnissen. Zwar kann das normalisierte Eingabemuster in ein statisches Grauwertbild umgewandelt werden, aber dabei bleibt die zeitliche Entstehung des Zeichens unberücksichtigt. Deshalb werden Merkmale benutzt, die diese zeitliche Information auswerten.
Für die Einzelzeichenerkennung haben sich bestimmte Methoden der Normalisierung und Merkmalsextraktion, wie sie in vielen Anwendungen benutzt werden [9, 4], bewährt und wurden in das NPen-System integriert und verbessert [16, 18].

3.1 Normalisierung

Größennormalisierung

Die Aufgabe der Größennormalisierung besteht darin, die Varianz in der Schriftgröße, die von Schreiber zu Schreiber variiert, auszugleichen. Da sich auch inner-

halb einer eingegebenen Formel die Größe der Zeichen stark ändern kann; z.B. bei Exponenten oder Indices, ist es wichtig, alle Zeichen auf eine Einheitsgröße zu projezieren. So ist das System in der Lage, unabhängig von der aktuellen Schreibgröße, Zeichen richtig zu erkennen. Dazu wird die Höhe jedes Zeichens auf eine Einheitsgröße skaliert, so daß sich die Y-Koordinaten der Eingabedaten im Wertebereich [0,1] befinden.

$$y_{neu}(t) = \frac{y_{alt}(t) - y_{min}}{y_{max} - y_{min}}, \qquad x_{neu}(t) = \frac{x_{alt}(t) - x_{min}}{y_{max} - y_{min}}$$

Mit den X-Koordinaten der Eingabedaten darf nicht genauso verfahren werden. Denn falls die X-Koordinaten mit der Breite des Zeichens normiert werden, dann bleibt das Längen- zu Breitenverhältnis des Eingabemusters nicht erhalten. Trotzdem ist es notwendig, das Eingabemuster auch in den X-Koordinaten invariant von der aktuellen Schreibposition und Schreibgröße zu halten. Deshalb wird der Dynamikbereich der X-Koordinaten eingegrenzt, indem auch sie mit der Höhe normiert werden.
Durch diese Vorgehensweise entsteht aber das Problem, daß die Größennormalisierung eines Zeichen unäbhangig von dessen Kontext verläuft. So werden Buchstaben, die sich in ihrem Schriftbild zwischen Klein- und Großschreibung hauptsächlich in der Größe unterscheiden wie z.B. [v,V;x,X;c,C;w,W], auf die gleiche Größe abgebildet und dadurch ihrem wichtigsten Unterscheidungsmerkmal beraubt. Dieser Verlust an Information kann durch spätere Vorverarbeitungsschritte nicht kompensiert werden und führt dazu, daß häufig Verwechselungen zwischen solchen Groß- und Kleinbuchstaben auftauchen. Bei der Erkennung handgeschriebener Wörter wird das Problem dadurch gelöst, daß die Größennormalisierung eines Zeichens in Abhängigkeit von dessen Kontext erfolgt, wofür die Berechnung einer Basis-, Ober- und Unterlinie notwendig ist. Da mit dieser Methode bei mathematischen Formeln wie z.B. Brüchen oder Exponenten keine sinnvollen Linien ermittelt werden können, kann dieser Ansatz nicht direkt für die Formelerkennung übernommen werden.

3.1.1 Nicht-lineare Interpolation fehlender Datenpunkte

In diesem Normalisierungsschritt sollen Aussetzer der Hardware während der Eingabephase der Formel ausgeglichen werden, um die Trajektorie zu vervollständigen bzw. besser darzustellen. Ursprünglich besteht das Eingabesignal aus einer Folge (x(t), y(t), p(t)) von Koordinaten, die die Hardware mit einer konstanten Abtastrate liefert. Dadurch ist die Anzahl der Datenpunkte in der Eingabe durch die Schreibgeschwindigkeit und die Abtastrate der Hardware bestimmt. Falls neben einer schnellen Schrift zusätzlich noch Abtastfehler der Hardware hinzukommen, ist es möglich, daß für ein Zeichen nur sehr wenige Datenpunkte vorhanden sind. Die Trajektorie solcher Datenpunkten stellt den ursprünglichen Schriftzug nur noch schlecht dar. Dieses Problem kann gelöst werden, indem zwischen zwei

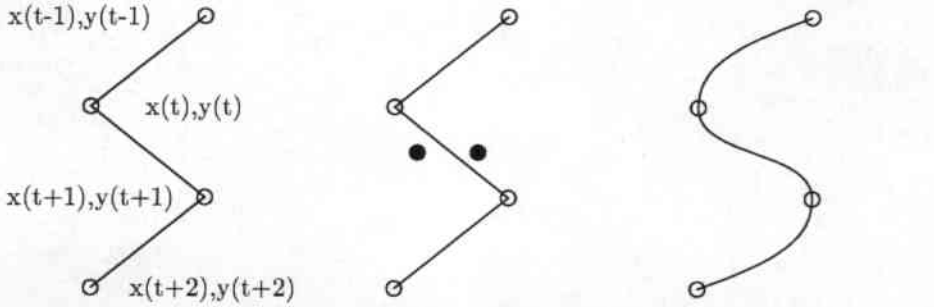

Abbildung 3.1: Interpolation fehlender Datenpunkte

benachbarten Datenpunkten mit einem großen Abstand neue Datenpunkte linear eingefügt werden. Dies wird bei einem weiteren Normalisierungsschritt, der Neuabtastung (s.u.), gemacht. Die Linearität bei der Neuabtastung sorgt aber nur dafür, daß unterschiedliche Schreibgeschwindigkeiten ausgeglichen werden. Sie sorgt aber nicht dafür, daß eine durch Aussetzungsfehler der Hardware unvollständige Trajektorie die Eingabe besser darstellt.
Dieses Problem läßt sich wie folgt lösen [18]: Falls der Abstand zweier benachbarter Datenpunkte (x(t), y(t), p(t)) und (x(t+1), y(t+1), p(t+1)) größer ist als ein vorgegebener Schwellwert, so wird durch die beiden Datenpunkte und zusätzlich berechneten Kontrollpunkte eine Bezierkurve gelegt. Diese Bezierkurve ist zusätzlich abhängig vom Verlauf der Trajektorie in den Datenpunkt (x(t), y(t), p(t)) hinein und aus dem Datenpunkt (x(t+1), y(t+1), p(t+1)) hinaus. Die neu hinzukommenden Datenpunkte werden auf der Bezierkurve gewählt und sorgen dafür, daß der Verlauf des Schriftzuges wesentlich natürlicher als bei einer reinen linearen Interpolation ist, siehe Abbildung 3.2.

3.1.2 Glättung

In diesem Normalisierungsschritt wird versucht, das Schriftbild der Trajektorie zu glätten; d.h. Effekte durch Ausreißer in den Eingabedaten, die durch Zittern der Handschrift oder Abtastfehler der Hardware zustande kommen, sollen vermindert werden. Dazu wird jeder Datenpunkt $(x_{alt}(t), y_{alt}(t))$ nach folgender Vorschrift in einen neuen Datenpunkt $(x_{neu}(t), y_{neu}(t))$ transformiert.

$$x_{neu}(t) = \frac{x_{alt}(t-N) + \cdots + x_{alt}(t-1) + \alpha x_{alt}(t) + x_{alt}(t+1) + \cdots + x_{alt}(t+N)}{2N + \alpha}$$

$$y_{neu}(t) = \frac{y_{alt}(t-N) + \cdots + y_{alt}(t-1) + \alpha y_{alt}(t) + y_{alt}(t+1) + \cdots + y_{alt}(t+N)}{2N + \alpha}$$

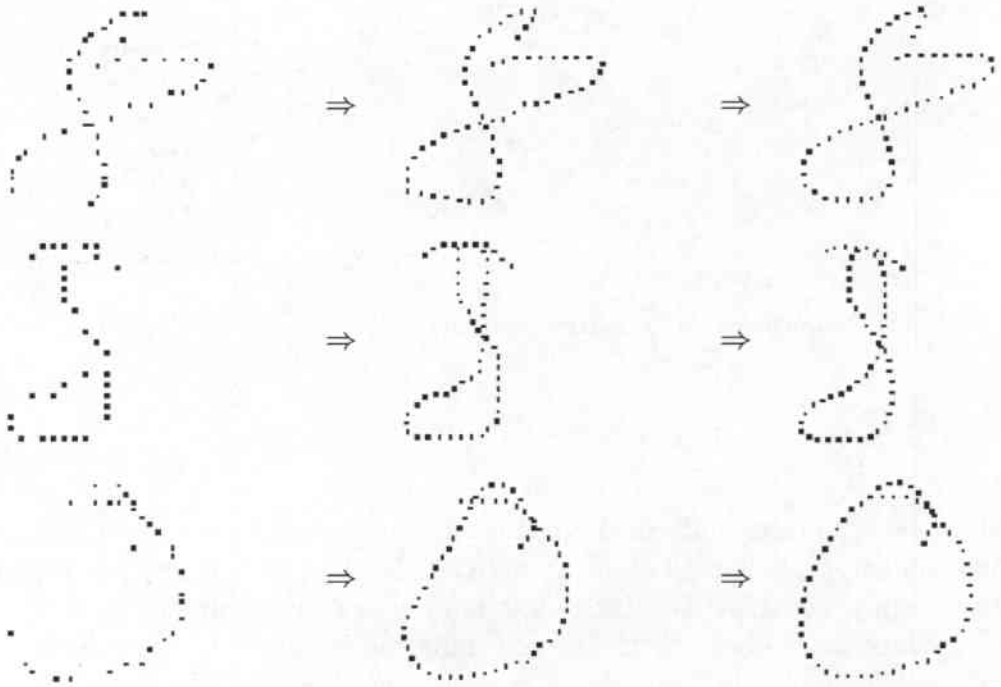

Abbildung 3.2: Hinzufügen neuer Datenpunkte mit Hilfe von Bezierkurven: links Original, mitte lineare Interpolation, rechts Interpolation mit Bezierkurven

In die Berechnung der geglätteten Koordinaten eines Datenpunktes fließen die Koordinaten der N vorausgehenden und nachfolgenden Datenpunkte sowie die, mit einem Gewicht α multiplizierten, originalen Koordinaten des Datenpunktes ein. Der Gewichtsfaktor ist dafür zuständig, daß bei geraden Segmenten die Liniencharakteristik verstärkt wird (α klein), und daß bei Spitzen und Kurven in der Trajektorie keine Abrundung vollzogen wird (α groß). Der Wert von α wird heuristisch für jeden Datenpunkt neu in Abhängigkeit vom Winkel der Trajektorie berechnet und bestimmt somit, in welchem Maße die Glättung durchgeführt wird.

3.1.3 Neuabtastung

Die Neuabtastung dient dazu, die Varianz der Schreibgeschwindigkeit zwischen verschiedenen Schreibern bzw. innerhalb der Eingabe auszugleichen. Da die Schreibgeschwindigkeit das Bild der Trajektorie extrem beeinflußt, ist dieser Normalisierungsschritt sehr wichtig. Die Neuabtastung transformiert die Folge der Datenpunkte von einer zeitlich äquidistanten Folge in eine räumlich äquidistante Folge, d.h. nach der Neuabtastung ist der räumliche Abstand zwischen 2 Datenpunkten (x(t), y(t)) und (x(t+1), y(t+1)) gleich. Die Wahl der Größe dieses Abstandes

ist entscheidend. Wird er sehr klein gewählt, dann wird eine hohe Anzahl von Datenpunkten erzeugt, die so dicht beieinander liegen, daß sie keine neuen Informationen liefern und damit den Berechnungsaufwand unnötig erhöhen. Wird dagegen der Abstand zu hoch gewählt, dann besteht die Gefahr, daß die Trajektorie nicht mehr gut genug repräsentiert wird und Information verloren geht.
Bei der Einzelzeichenerkennung wird die Neuabtastung so durchgeführt, daß jedes Zeichen nach der Neuabtastung durch eine fest vorgegebene Anzahl von Punkten dargestellt wird. Diese feste Abtastrate wird für jedes Zeichen gleich angewendet, egal ob es sich um ein komplexes Zeichen oder einen einfachen Strich bzw. Punkt handelt.

3.1.4 Aufsetzungsfehler

Beim Betrachten der Daten fällt auf, daß bei vielen Zeichen am Anfang eines Strokes ein kleiner Haken vorhanden ist (siehe Abbildung 3.3). Dieser Haken gehört nicht zum eigentlichen Zeichen, sondern führt eher zu Verwechselungen wie z.B. bei dem Pluszeichen. Die Ursache für das Erscheinen der Haken liegt in der Aufnahmehardware. Da die Eingabe mit dem Stift direkt auf der Bildschirmoberfläche erfolgt, aber das Bild erst hinter der Oberfläche erscheint, wird die Auge-Hand-Koordination bei dem Aufsetzen des Stiftes auf dem Bildschirm getäuscht. Der Benutzer erwartet das Aufsetzen des Stiftes erst zu einem kurz späteren Zeitpunkt, führt aber dann, sobald er den Kontakt spürt, den Schriftzug zu Ende, so daß unbeabsichtigt ein kleiner Haken am Anfang entsteht.
Das Ziel in diesem Normalisierungsschritt besteht darin, diesen Haken aus dem Schriftzug mit Hilfe einer Heuristik zu entfernen. Da die Haken dadurch charakterisiert sind, daß sie nur am Anfang der Trajektorie vorkommen und durch eine plötzliche Richtungsänderung enden, läßt sich mit folgendem Ansatz das Problem lösen: nur die vordersten Datenpunkte des Schriftzuges werden durchlaufen, und es wird geprüft, ob der Winkel zwischen dem aktuellen Datenpunkt und seinem Vorgänger und seinem Nachfolger spitz ist. Falls dies der Fall ist, so werden alle Datenpunkte vor dem spitzen Winkel entfernt.
In der Abbildung 3.3 wird ersichtlich, daß durch diese Vorgehensweise die Aufsetzungsfehler aus dem Schriftzug entfernt werden können. Die Notwendigkeit dieses Normalisierungsschritts wird besonders bei dem Pluszeichen deutlich. Denn ohne die Korrektur ist es leicht mit einer 4 zu verwechseln. Andererseits wird auch deutlich, daß nicht alle Aufsetzungsfehler mit dem Verfahren erkannt werden. So werden bei dem dritten Beispiel, dem X, nicht beide Fehler korrigiert. Denn die Richtungsänderung bei dem einem Stroke ist nicht abrupt genug, als daß sie das Kriterium vom spitzen Winkel erfüllt und damit als Aufsetzungsfehler erkannt wird.

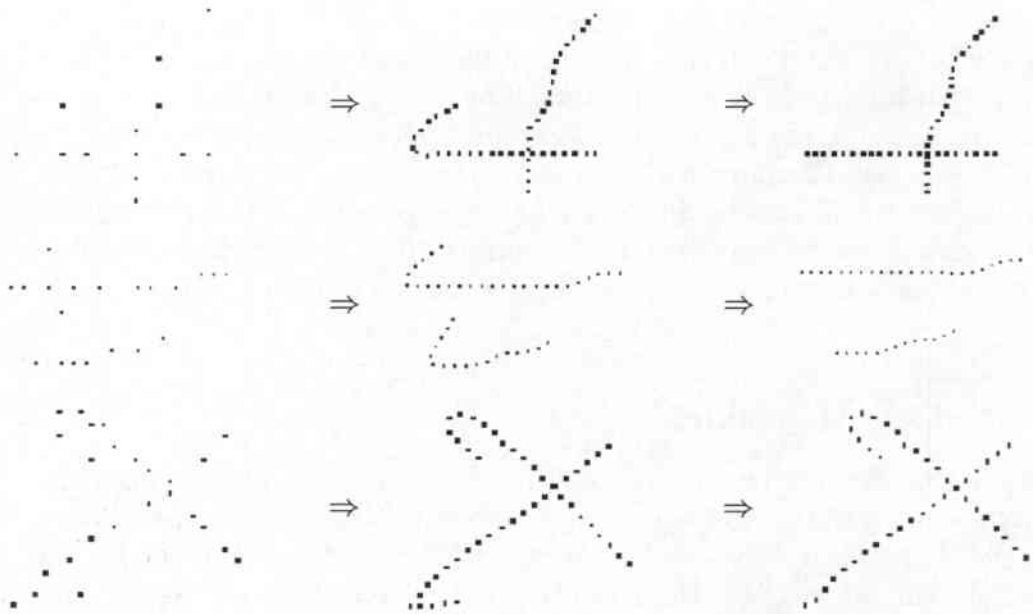

Abbildung 3.3: Korrektur von Aufsetzfehlern: links Original, Mitte ohne Korrektur, rechts mit Korrektur

3.2 Merkmalsextraktion

Nach der Normalisierung ist es notwendig, das Eingabesignal in eine geeignete Repräsentation für den Erkennungsprozeß zu transformieren. Eine Möglichkeit besteht darin, nur das Schriftbild des Zeichens zu betrachten und die zeitliche Information über die Entstehung des Zeichens zu vernachlässigen. In Experimenten [25] konnte aber gezeigt werden, daß sich bei dieser Vorgehensweise die Erkennungsergebnisse verschlechtern, so daß die zeitliche Information bei der Merkmalsextraktion unbedingt berücksichtigt werden sollte. Dazu wird jeder Datenpunkt (x(t), y(t)) der normalisierten Trajektorie in einen N-dimensionalen Merkmalsvektor $x_t = (f_1(t), \cdots, f_N(t))$ transformiert, d.h. die Merkmalsextraktion liefert dem Erkennungsprozeß eine Folge von Merkmalsvektoren $(x_1, \cdots, x_{T'})$. Wichtig bei der Online-Erkennung ist aber die sorgfältige Auswahl der Merkmale. Schließlich repräsentieren sie das Eingabesignal und beeinflussen direkt die Erkennungsleistung; deshalb sollen die Merkmale möglichst viel über die charakteristischen Eigenschaften der Trajektorie aussagen, während unwichtige Eigenschaften vernachlässigt werden können. Ziel der Merkmalsextraktion ist es, daß die Merkmalsvektoren eines Zeichens im Merkmalsraum ein möglichst kompaktes Gebiet einnehmen, während sich die Gebiete der Merkmalsvektoren verschiedener Zeichen möglichst scharf voneinander abtrennen. Der erste Teil dieses Postulats der Mustererkennung [21] wird durch die Normalisierung erreicht und der zweite

Teil durch eine geeignete Merkmalsgewinnung.

3.2.1 Stiftzustand

Bei diesem Merkmal soll die Information, ob der Stift zum Zeitpunkt t auf der Schreiboberfläche aufsetzt oder abgehoben ist, weitergegeben werden. Durch die Neuabtastung werden neue Datenpunkte zwischen dem Abheben des Stiftes und dem erneuten Aufsetzen auf die Schreiboberfläche linear eingefügt. Damit diese neuen Datenpunkte nicht fälschlicherweise zu dem originalen Schriftzug gehörend eingeteilt werden, muß in den Merkmalsvektoren die Information enthalten sein, ob sich der Stift zum Zeitpunkt t in dem sichtbaren oder dem unsichtbaren Bereich der Trajektorie befindet. Deshalb wird dem Merkmalsvektor ein Merkmal für den Stiftzustand p(t) zugefügt. Dieses Merkmal wird auf 1 gesetzt, falls der Stift auf der Schreiboberfläche aufsitzt und auf 0, falls der Stift abgehoben ist.

3.2.2 Schreibrichtung

Die aktuelle Schreibrichtung wird durch zwei Werte repräsentiert, indem der Kosinus und der Sinus des Winkels zwischen der horizontalen Linie und der Verbindungslinie zwischen den beiden Nachbarpunkten (x(t-1), y(t-1)) und (x(t+1), y(t+1)) berechnet wird (siehe Abbildung 3.4).

$$\cos\alpha(t) = \frac{\Delta x(t)}{\Delta s(t)}$$
$$\sin\alpha(t) = \frac{\Delta y(t)}{\Delta s(t)}$$

Dabei ist die Länge der Verbindungslinie als $\Delta s(t) = \sqrt{\Delta x^2(t) + \Delta y^2(t)}$ definiert, mit einer Höhe von $\Delta x(t) = x(t-1) - x(t+1)$ und einer Länge von $\Delta y(t) = y(t-1) - y(t+1)$.

3.2.3 Krümmung

Die Krümmung in einem Datenpunkt des Schriftzuges wird auch durch zwei Werte repräsentiert, indem der Kosinus und der Sinus des Winkels zwischen der Verbindungslinie zwischen den Datenpunkten (x(t-2),y(t-2)) und (x(t),y(t)) und der Verbindungslinie zwischen den Datenpunkten (x(t),y(t)) und (x(t+2),y(t+2)) berechnet wird (siehe Abbildung 3.4). Durch Verwendung trigonometrischer Funktionen lassen sich diese Werte aus den Merkmalen der Schreibrichtung ableiten.

$$\cos\beta(t) = \cos\alpha(t-1) * \cos\alpha(t+1) + \sin\alpha(t-1) * \sin\alpha(t+1)$$
$$\sin\beta(t) = \cos\alpha(t-1) * \sin\alpha(t+1) - \sin\alpha(t-1) * \cos\alpha(t+1)$$

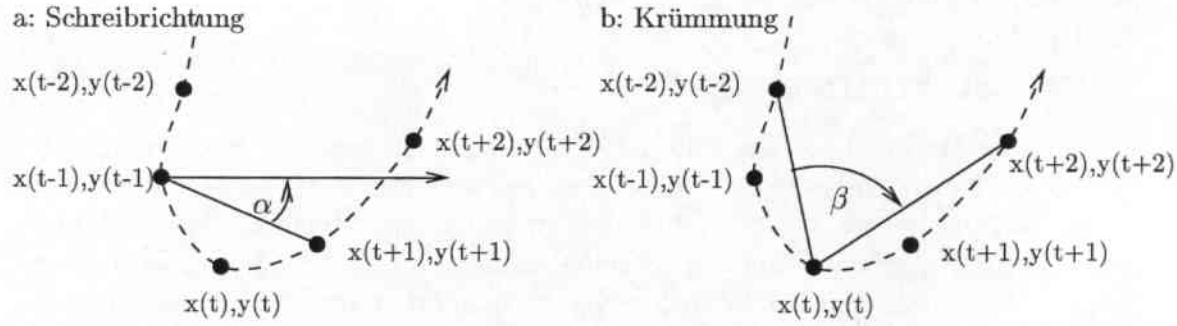

Abbildung 3.4: Definitionen für die Berechnung weiterer Merkmale

3.2.4 Erscheinungsbild

Für die Berechnung dieses und der folgenden Merkmale wird nicht nur der aktuelle Datenpunkt betrachtet, sondern es wird ein Fenster der Breite 5 über die Datenpunkte geschoben. Dadurch beruhen die Berechnungen nicht nur auf dem aktuellen Datenpunkt sondern auch auf dessen Kontext. Die Abbildung 3.5 zeigt die für die Berechnung der nächsten Merkmale notwendigen Größen.
Das Erscheinungsbildmerkmal ist definiert als das Verhältnis zwischen Höhe und Breite des betrachteten Fensters:

$$A(n) = \frac{2 * \Delta y(n)}{\Delta x(n) + \Delta y(n)} - 1,$$

wobei $\Delta x(n)$ die Höhe und $\Delta y(n)$ die Breite des Fensters darstellt. Der Wertebereich diese Merkmals liegt somit immer im Interval [-1,1].

3.2.5 Linearität

Dieses Merkmal gibt Auskunft darüber, wie linear die Trajektorie um den aktuellen Datenpunkt verläuft. Für die Berechnung des Linearitätsmerkmals wird immer eine Umgebung von N Datenpunkten betrachtet, und der quadratische Abstand eines jeden Datenpunktes zu der Verbindungslinie zwischen dem ersten und letzten Datenpunkt der Umgebung aufsummiert. Zusätzlich wird durch die Anzahl der Datenpunkte normiert.

$$L(t) = \frac{1}{N} * \sum_i d_i^2,$$

wobei d_i der Abstand von Punkt i von der Verbindungslinie ist. Der Wertebereich liegt im Interval [0,R], mit $R < (\Delta x^2 + \Delta y^2)$.

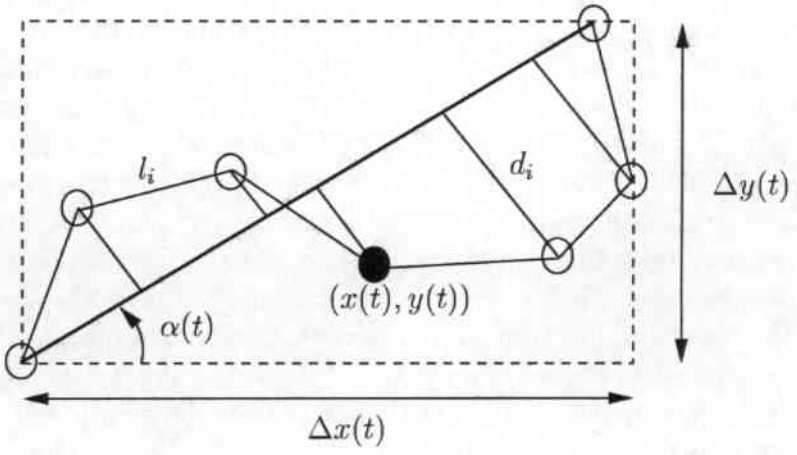

Abbildung 3.5: Definitionen für die Berechnung weiterer Merkmale

3.2.6 Gewelltheit

Auch dieses Merkmal beschreibt die Linearität im aktuellen Kurvensegment. Es ist definiert als das Verhältnis zwischen der Bogenlänge des Liniensegments und dem Maximum von der Höhe und Breite der umgebenden Box:

$$C(t) = \frac{(N-1)l_i}{max(\Delta x, \Delta y)} - 2,$$

dabei gibt l_i die Länge des Segments an.

3.2.7 Neigung

Dieses Merkmal ist ein Maß dafür, in welche Richtung sich das Liniensegment gegenüber der horizontalen Linie bewegt und ist definiert als der Kosinus von $\alpha(t)$, des Winkels zwischen der direkten Verbindungslinie des Segments und der horizontalen Linie.

3.2.8 Kontext-Bitmaps

Alle bisherigen Merkmale zeichnen sich dadurch aus, daß sie nur zeitlich eng benachbarte Datenpunkte in ihre Berechnung einbeziehen. Diese rein lokale Sicht bei der Online-Erkennung führt aber zu dem Problem, daß Beziehungen zwischen zwei zeitlich entfernten Datenpunkten nicht berücksichtigt werden. Eine solche Beziehung zwischen 2 Datenpunkten besteht z.B. wenn sich die Trajektorie überschneidet oder räumlich eng nebeinander verläuft. Bei der Offline-Erkennung liegt dagegen meistens eine globale Sicht auf die Daten vor. Hier sind die räumlichen

Beziehungen zwischen den Datenpunkten das entscheidende Kriterium. Damit bei der Online-Erkennung nicht auf diese wichtigen Merkmale verzichtet werden muß, wurde in [16] ein Ansatz vorgestellt, wie Online- mit Offlinemerkmalen kombiniert werden können.
Der grundlegende Gedanke dabei ist, daß dem Merkmalsvektor zum Zeitpunkt t Wissen über die räumliche Umgebung des Datenpunktes (x(t), y(t)) hinzugefügt wird. Dafür wird ein Grauwertbild (Kontext-Bitmap) von einem 9x9 Fenster mit dem aktuellen Datenpunkt als Mittelpunkt erstellt. Zwar wäre es möglich, diese Grauwertinformation direkt als Merkmal zu verwenden, aber dies würde zu einem Ungleichgewicht zwischen den globalen und lokalen Merkmalen führen. Denn die 11 lokalen Merkmale würden gegenüber den 81 globalen Merkmalen kaum noch ins Gewicht fallen. Ziel muß es deshalb sein, die Grauwertinformationen aus dem Bildausschnitt zu reduzieren. Dies wird erreicht, indem der Bildauschnitt auf ein 3x3 Fenster verkleinert wird, wobei sich die neuen Grauwerte aus einer Interpolation der alten Grauwerte ergeben. Diese 9 Grauwerte werden an den Merkmalsvektor angehängt und sorgen dafür, daß nun zusätzlich eine globale Sicht vorhanden ist.

Kapitel 4

Einzelzeichenerkenner

Durch die in den vorhergehenden Abschnitten beschriebenen Schritte wurde zuerst eine mögliche Segmentierung der Eingabe ermittelt und dann durch die Vorverarbeitung eine geeignete Repräsentation des Eingabesignals in Merkmalsvektoren gefunden. Es schließt sich nun das Klassifizierungsproblem an, bei dem die einzelnen Zeichen erkannt werden müssen. Für diese Aufgabe konnte auf das NPen++-System zurückgegriffen [17] werden, das für die Erkennung von kursiv geschriebenen Wörtern und Sätzen entwickelt worden ist, aber auch einen Einzelzeichenerkenner enthält. Dessen Arbeitsweise und dessen Methoden werden in diesem Kapitel erläutert.
Allgemein wird bei dem Erkennungsprozeß die Eingabe, die als zeitliche Folge von Merkmalsvektoren vorliegt, dem Zeichen zugeordnet, für das die Wahrscheinlichkeit gemäß des erkennungsinternen Modells des Zeichens am größten ist. Dies drückt die Formel von Bayes wie folgt aus:

$$P(Z_l|X) = \frac{P(X|Z_l) * P(Z_l)}{P(X)}$$

Die Entscheidung des Einzelzeichenproblems fällt zugunsten desjenigen Zeichens Z_{l^*} aus, das die maximale a posteriori Wahrscheinlichkeit $P(Z_l|X)$ besitzt. Der Dichtewert im Nenner ist bei der Maximumbildung unabhängig von dem Zeichen Z_l und kann deshalb vernachlässigt werden. Die a priori Wahrscheinlichkeit $P(Z_l)$ dient der bevorzugten Auswahl häufig auftretender Zeichen. So ist bekannt, daß z.B. bei der Zeichenerkennung innerhalb von Wörtern die Vokale a,e wesentlich häufiger auftreten als die Konsonanten x,y. Die konkreten Werte der Wahrscheinlichkeit lassen sich auf einer Textprobe schätzen. In dem Fall der Formelerkennung wurden bisher keine Untersuchungen durchgeführt, so daß eine gleichförmige Verteilung $P(Z_l) = 1/L$ zugrunde gelegt wird. Insgesamt fällt die Entscheidung damit nur noch auf Basis der bedingten Dichte $P(X|Z_l)$, der Wahrscheinlichkeit, daß die Eingabe X vorliegt, wenn das Zeichen Z_l geschrieben wurde.
Da die Dichtewerte $P(X|Z_l)$ unbekannt sind, wird die Dichtefunktion für jedes

Zeichen Z_l mit Hilfe eines Paramtersatzes λ_l durch parametrisierte Funktionen angenähert. Für diesen Zweck hat sich der Einsatz von Hidden Markov Modellen (HMMs) durchgesetzt, da sie mit dem Baum-Welch-Algorithmus eine automatische Schätzung des Parametersatzes λ_l ermöglichen. Der grundlegende Aufbau des Einzelzeichenerkenners ist in Abbildung 4.1 zusammengefaßt.

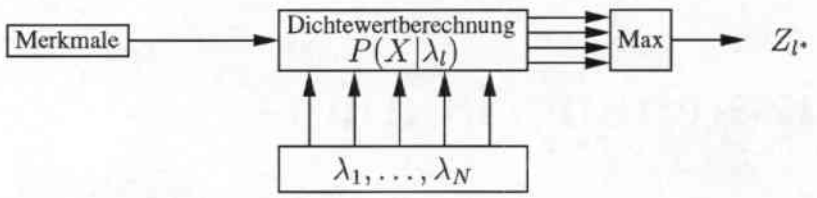

Abbildung 4.1: Einzelzeichenerkennung mit Hidden Markov Modellen

4.1 Hidden Markov Modelle

Hidden Markov Modelle bilden bei der Spracherkennung die erfolgreiche Basis für den Erkennungsprozeß. Da das Problem der Schrifterkennung ähnlich geartet ist wie das Problem der Spracherkennung, in beiden Fällen besteht die Eingabe aus einer Folge von Daten mit zeitlicher Variabilität, werden die HMMs auch bei der Schrifterkennung eingesetzt. Vorteilhaft ist dabei, daß auf die gewonnenen Erkenntnisse bei der Verwendung der HMMs bei der Spracherkennung zurückgegriffen werden kann [22].

4.1.1 Definitionen

Ein HMM λ erster Ordnung ist ein doppelt stochastischer Prozeß und formal definiert als ein Fünftupel $\lambda = (A, B, \pi, N, M)$. Dabei sind die Komponenten wie folgt definiert:

- N : Die Anzahl der HMM-Zustände, $S = \{s_1, \ldots, s_N\}$. Zu jedem Zeitpunkt t wird genau einer dieser Zustände $q_t = s_i$ angenommen.
- M : Die Anzahl der verschiedenen Symbole im Beobachtungsalphabet, $V = \{v_1, \ldots, v_M\}$. Zu jedem Zeitpunkt t wird genau eine dieser Beobachtungen $O_t = v_i$ gemacht, wobei eine Beobachtung dem Merkmalsvektor zum Zeitpunkt t entspricht.
- A : Die Matrix der Übergangswahrscheinlichkeiten gibt an, mit welcher Wahrscheinlichkeit ein Zustand s_j dem gegebenen Zustand s_i folgt.

$$a_{ij} = P(q_t = s_j | q_{t-1} = s_i) \qquad für \quad i, j = 1, \ldots, N$$

Die Einträge der Matrix a_{ij} genügen den Stochastizitätsbedingungen, d.h. es gilt $a_{ij} \geq 0$ und $\sum_j a_{ij} = 1$. In den meisten Fällen wird aber nur ein lineares Modell benutzt.

- π : In dem N-dimensionalen Vektor π sind die Wahrscheinlichkeiten für die Einnahme des Anfangszustands enthalten, $\pi_i = P(q_1 = s_i)$.
- B : Die Menge der Emissionswahrscheinlichkeiten $B = \{b_1, \ldots, b_N\}$. Dabei beschreibt $b_j(x)$ die Wahrscheinlichkeit im Zustand s_j zu sein und die Eingabe x zu beobachten, $b_j(x) = P(O_t = x | q_t = s_j)$.

Mit Hilfe eines HMMs läßt sich nun die gewünschte Dichtefunktion $P(X|Z_l)$ berechnen. Für einen Beobachter ist dabei nur die Folge der Beobachtungen (Merkmalsvektoren) $O_1, \ldots, O_T$ sichtbar, aber nicht die Folge der eingenommenen Zustände $q_1, \ldots, q_T$. Bei der Verwendung von HMMs ergeben sich drei Probleme, die in den nächsten Abschnitten aufgezeigt und deren Lösung vorgestellt werden soll.

4.2 Evaluierungsproblem

Eine Aufgabe bei HMMs besteht darin, zu einer Beobachtungsfolge $O = o_1, \ldots, o_T$ die Wahrscheinlichkeit $P(O|\lambda)$ zu bestimmen, mit der das HMM λ die Beobachtung O erzeugt. Diese Wahrscheinlichkeit wird während der Erkennung für jedes Modell berechnet und ihr maximaler Wert entscheidet darüber, welches Zeichen der Beobachtungsfolge zugeordnet wird. Deshalb ist es wichtig, daß mit dem 'Vorwärts'-Algorithmus ein effizientes Berechnungsverfahren für $P(O|\lambda)$ vorhanden ist. Dieser Algorithmus beruht auf dem Prinzip der dynamischen Programmierung und besitzt einen Aufwand von $O(N^2T)$. Zur Berechnung wird die Vorwärtswahrscheinlichkeit $\alpha_t(j) = P(o_1 \ldots o_t, q_t = s_j|\lambda)$, zum Zeitpunkt t im Zustand s_j zu sein und zuvor die Beobachtungen $o_1, \ldots, o_t$ zu machen, definiert. Sie läßt sich nach folgender Vorschrift rekursiv berechnen:

$$\alpha_t(j) = \begin{cases} \pi_j b_j(o_1) & \text{für} \quad t = 1 \\ \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) & \text{für} \quad t > 1 \end{cases}$$

Demnach stellt $\alpha_T(j)$ die Wahrscheinlichkeit dar, die gesamte Beobachtung O zu machen und im Zustand s_j zu terminieren, so daß sich die gesuchte Wahrscheinlichkeit von $P(O|\lambda)$ als $P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$ ergibt.

4.3 Dekodierungsproblem

Bei dem Dekodierungsproblem wird nach der Zustandsfolge Q gesucht, die die Ausgabewahrscheinlichkeit für eine gegebene Beobachtung $o_1, \ldots, o_T$ maximiert:

$$\arg \max_{i_1 \ldots i_T} p(q_{i_1} \ldots q_{i_T} | o_1 \ldots o_T, \lambda)$$

Eine Lösung läßt sich mit Hilfe des Viterbi-Algorithmus finden, der auch auf dem Prinzip der dynamischen Programmierung beruht. Statt der $\alpha_t(j)$ werden jetzt die maximal erzielbaren Wahrscheinlichkeiten

$$\vartheta_t(j) = \max_{q_1 \ldots q_t} P(q_1 \ldots q_t = s_j, o_1 \ldots o_t | \lambda)$$

für die Erzeugung der Teilausgabe $o_1 \ldots o_t$ unter der Nebenbedingung berechnet, daß der Produktionsvorgang im Zustand s_j endet. Gleichzeitig wird eine Rückverzeigerungsmatrix $[\psi_t(j)]$ zur abschließenden Extraktion der gesuchten Folge aufgebaut. Nach der Initialiserung der Werte $\vartheta_1(j) = \pi_j b_j(O_1)$ und $\psi_1(j) = 0$ für alle $j = 1, \ldots, N$ erfolgt eine rekursive Berechnung:

$$\vartheta_t(j) = \max_i(\vartheta_{t-1}(i)a_{ij})b_j(O_t) \tag{4.1}$$

$$\psi_t(j) = \arg\max_i \vartheta_{t-1}(i)a_{ij} \tag{4.2}$$

Aus den berechneten Werten kann die gewünschte Zustandsfolge Q^* nach folgender Vorschrift ermittelt werden: $q_t^* = \psi_{t+1}(q_{t+1}^*)$, wobei $q_T^* = \arg\max_i \psi_T(j)$ gesetzt wird.

4.4 Optimierungsproblem

Das Optimierungsproblem ist definiert als das Auffinden eines Parametersatzes $\hat{\lambda}$, für den gilt $\hat{\lambda} = \arg\max_\lambda P(o_1 \ldots o_T|\lambda)$. Eine analytische Lösung für dieses Problem existiert nicht. Stattdessen wird mit einer iterativen Methode zu einem gegebenen λ ein $\overline{\lambda}$ gefunden, so daß die Wahrscheinlichkeit eine Merkmalsfolge $o_1 \ldots o_T$ zu beobachten, mit den Parametern von $\overline{\lambda}$ größer ist als mit dem Parametern von λ, also $P(o_1 \ldots o_T|\overline{\lambda}) > P(o_1 \ldots o_T|\lambda)$. Diese iterative Methode ist als der Baum-Welch-Algorithmus bekannt, der durch Aufsummieren der durchlaufenden Zustände und der vorgekommenden Zustandsübergänge neue Werte für $\overline{A}, \overline{B}$ und $\overline{\pi}$ schätzt. Zwar kann durch dieses Verfahren nicht garantiert werden, daß eine optimale Lösung gefunden wird, aber es konnte gezeigt werden, daß das Verfahren zumindestens in einem lokalen Maximum terminiert.

4.5 Emissionswahrscheinlichkeiten

Der Erfolg beim Einsatz von HMMs hängt entscheidend davon ab, wie gut die Emissionswahrscheinlichkeiten $b_j(o_t)$ eine Beobachtung o_t einordnen. Bei der einfachsten Form der HMMs werden nur diskrete Zeichen als Beobachtung zugelassen. Dies erfordert aber den Einsatz eines Vektorquantisierers, der die Merkmalsvektorfolge in eine Folge von diskreten Zeichen transformiert. Mit dieser Operation handelt man sich aber unweigerlich einen Informationsverlust ein, der von keinem Verarbeitungsschritt kompensiert werden kann. Der Sinn und Zweck

liegt darin, den Berechnungsaufwand klein zu halten, was heutzutage nicht mehr notwendig ist.
Stattdessen werden HMMs mit kontinuierlichen Emissionswahrscheinlichkeiten $b_j(x), x \in [-\infty, \infty]^D$ verwendet, so daß der Merkmalsvektor x direkt zur Berechnung der Ausgabewahrscheinlichkeit im Zustand s_j benutzt wird. Damit beliebige Dichtefunktionen angenommen werden können, wird die Ausgabeverteilung durch gewichtete Überlagerung einer hinreichenden Anzahl von Normalverteilungen approximiert. Für jeden Zustand müssen nun die Mittelvektoren und die Kovarianzmatrizen dieser Normalverteilungen zusammen mit den Mixturgewichten geschätzt werden. Durch die hohe Anzahl der Parameter entsteht aber hierbei ein zu großer Aufwand, so daß die Anzahl der Normalverteilungen reduziert werden muß. Deshalb besitzt jeder Zustand nicht mehr seine eigenen Normalverteilungen, sondern greift auf einen Pool von Normalverteilungen zu, der von allen Zuständen gemeinsam genutzt wird. Da jeder Zustand eigene Mixturgewichtverteilung besitzt, können wiederum beliebige Ausgabeverteilungen erzeugt werden. Diese häufig eingesetzte Form des HMMs ist als semikontinuierliches Markovmodell bekannt.
Eine Weiterentwicklung der HMMs wurde versucht, indem die Emissionswahrscheinlichkeiten mit neuronalen Netzen ermittelt wurden. Da sie bessere diskriminative Eigenschaften besitzen als die Mixturgewichtverteilungen, sind sie für diesen Einsatzzweck geeignet.

4.6 MS-TDNN

Das Multi-State Time Delay Neuronal Network (MS-TDNN) [7, 8, 26] ist ein mehrschichtiges, vorwärtgerichtetes Netz, dessen Leistungsfähigkeit im Verarbeiten von sequentiellen Datenströmen besteht. Ursprünglich wurde es bei der Spracherkennung eingesetzt, aber auch bei der Zeichenerkennung, kann es seine Fähigkeit, zeitlich veränderliche Muster unterschiedlicher Länge zu erkennen, ausspielen. Im Gegensatz zu seinem Vorgänger, dem TDNN, wird bei dem MS-TDNN ein Zeichen nicht als Ganzes modelliert sondern als eine Folge von Zuständen. Diese Unterscheidung entspricht mehr den natürlichen Gegebenheiten und unterstützt so den Erkennungsprozeß. Denn die Trajektorie eines Zeichens läßt sich auch in verschiedene Bereiche unterteilen, in denen bestimmte Eigenschaften, wie z.B. Krümmung oder Schreibrichtung, vorherrschen. Diese Unterteilung wird bei der Modellierung übernommen, indem jedes Zeichen durch eine bestimmte Anzahl von Zuständen (Bereichen) beschrieben wird. In der Abbildung 4.2 ist dargestellt, wie ein solches Modell für ein Zeichen aussieht. In dieser Arbeit wurde für jedes Zeichen ein lineares Modell mit 4 Zuständen gewählt.

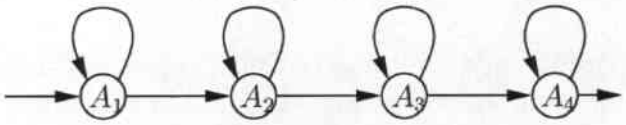

Abbildung 4.2: Modellierung eines Zeichens mit Hilfe eines Links-Rechts-HMM, das aus 4 Zuständen besteht.

4.6.1 Architektur des MS-TDNN

Der grundlegende Aufbau eines MS-TDNN ist in der Abbildung 4.3 dargestellt. Dabei wird eine wichtige Eigenschaft des MS-TDNN deutlich und zwar, daß nicht nur der aktuelle Merkmalsvektor alleine betrachtet wird, sondern daß er immer im Kontext mit den vorhergehenden und nachfolgenden Merkmalsvektoren verwendet wird. Dieses Fenster, das über die Merkmalsvektoren geschoben wird, ist dafür verantwortlich, daß ein MS-TDNN zeitliche Abhängigkeiten zwischen den Merkmalsvektoren erlernen kann. Zu jeden Zeitpunkt werden für die Merkmalsvektoren des Fensters die Wahrscheinlichkeiten berechnet, welcher HMM-Zustand gerade eingenommen wird.
Sowohl die Werte in der verborgenen Schicht als auch in der Zustandsschicht ergeben sich aus den Werten der Vektoren innerhalb des Fensters zusammen mit einer Gewichtsmatrix. Die Aktivierungen in der Zustandsschicht werden interpretiert als die Wahrscheinlichkeit, daß zum Zeitpunkt t der Zustand s_i eingenommen wird. Als Resultat werden somit die Aktivierungen der einzelnen Zustände über die gesamte Zeitachse berechnet. Damit das MS-TDNN die Eingabe richtig klassifizieren kann, ist abschließend zu prüfen, mit welchen Wahrscheinlichkeiten die einzelnen Zeichen angenommen werden. Dazu wird versucht, durch die Aktivierungen der Zustände eines Zeichens einen Viterbipfad zu finden. Damit wird gewährleistet, daß das HMM-Modell des Zeichens vom Anfangs- zum Endzustand durchlaufen wird. Die Werte für die besten Viterbipfade eines jeden Zeichens werden als Aktivierungen in die Ausgabeschicht übergetragen.

4.6.2 Training des MS-TDNN

Prinzipiell werden die Gewichte des MS-TDNN genauso wie die des TDNN mit einem Backtrackingverfahren trainiert. Bei der Verwendung von MS-TDNNs ergibt sich aber das Problem, daß für ein gegebenes Zeichen die Sollwerte für die Zustandschicht unbekannt sind. Die sind aber notwendig, um den Fehler in der Zustandsschicht zu berechnen und davon ausgehend die Gewichte in den einzelnen Schichten zu adjustieren. Deshalb wird der Viterbipfad π des Zeichens berechnet, so daß jedem Merkmalsvektor der Eingabe ein Zustand $\pi(t)$ zugeordnet werden kann. Auf diese Weise erhält man zu jedem Zeitpunkt der Eingabe den

gewünschten Sollwert in der Zustandschicht, so daß das Backtrackingverfahren starten kann.

4.7 Ergebnisse

Mit den in den vorherigen Abschnitten beschriebenen Verfahren wurde ein schreiberunabhängiger Einzelzeichenerkenner für die Formelerkennung trainiert. Er enthält neben den Ziffern, den Groß- und Kleinbuchstaben auch 10 mathematische Operatoren: $+, -, *, =, (,), \sqrt{}, \sum, \int, \infty$. Mit diesen insgesamt 72 Zeichen lassen sich die häufigsten mathematischen Formeln darstellen.
Um den Einzelzeichenerkenner zu trainieren, wurden die geschriebenen Einzelzeichen aus der Datenbasis (siehe Kapitel 8) verwendet. Die Trainingsmenge enthält Daten von 24 Schreibern; die Testmenge von 4 Schreibern. Damit die Testmenge nicht zu klein ausfällt und somit nicht aussagekräftig wird, wurden ihr zusätzliche Beispiele zugefügt. Diese Beispiele wurden aus segmentierten Formeln genommen, die von 18 verschiedenen Schreibern kamen. Die Aufteilung in Trainings- und Testmenge kann aus der Tabelle 4.1 entnommen werden.
Die Erkennungsleistung auf der Testmenge kann aus der Tabelle 4.2 entnommen

	Trainingsmenge	Testmenge
Ziffern:	227	398
Buchstaben:	1185	203
Math. Operatoren:	219	129
Summe:	1631	730

Tabelle 4.1: Aufteilung der Trainings- und Testmenge

werden. Da in der Testmenge die relative Häufigkeit der Ziffern wesentlich größer ist als die relative Häufigkeit der Buchstaben oder der mathematischen Operatoren wird die Erkennungsleistung zusätzlich auch getrennt nach diesen Bereichen angegeben. Im Folgenden sollen die Fehler in den einzelnen Bereichen analysiert

	Fehler(%)	Fehler(abs.)
Ziffern:	2.51	10
Buchstaben:	30.54	62
Math. Operatoren:	16.27	21
Insgesamt:	12.73	93

Tabelle 4.2: Prozentuale und absolute Fehler auf der Testmenge

werden. Besonders fällt die schlechte Erkennungsleistung bei den Buchstaben auf. Dies liegt vor allen daran, daß für bestimmte Buchstaben die Unterscheidung zwischen Groß- und Kleinbuchstaben sehr schwierig ist. Falls solche Fehler unberücksichtigt bleiben, z.B. Verwechselungen zwischen v und V, so reduziert sich die Fehlerrate auf 18.72%. Dann dominieren Verwechselungen zwischen Zeichen, die diese Erkennungsaufgabe grundsätzlich schwierig machen, wie z.B. wie z.B. g und y, e und l, n und r, R und P, A und H.
Bei den mathematischen Operatoren finden hauptsächlich Fehlklassifikationen bei dem Malzeichen statt. So sind von 21 Fehlern alleine 8 darauf zurückzuführen, daß ein Malzeichen mit einem x verwechselt wurde. Der Grund für diese häufige Verwechselung liegt zuerst an der ähnlichen Schreibweise der beiden Zeichen, aber auch daran, daß viele Schreiber anstatt eines Malzeichens in einer Formel tatsächlich ein x schreiben. Dieser Fehler hat seinen Ursprung in der Datensammlung, bei der das Malzeichen anstatt mit $*$ mit $\times$ angegeben war. Ein weiterer häufiger Fehler ist die Verwechselung des Pluszeichens mit einer 4. Dieser Fehler tauchte in der Testmenge insgesamt fünfmal auf, wobei die Ursache auch auf der ähnlichen Schreibweise der Zeichen beruht.
Die Fehlerrate bei den Ziffern ist mit 2.51% erfreulich gering. Typische Fehlerarten treten hier nicht auf.
Auch für den Menschen ist es oft nicht möglich, eindeutig zu entscheiden, welches Zeichen geschrieben wurde. Gerade bei typischen Verwechselungen, wie z.B. Malzeichen mit x, kann keine korrekte Entscheidung nur aus der Berücksichtigung des Zeichens vollzogen werden. Stattdessen wird der Kontext des Zeichens betrachtet, um eine Lösung zu ermitteln. Deshalb wird z.B. ein Mensch die Lösung '$4 * 4 = 16$' der Lösung '$4x4 = 16$' vorziehen. Solche Kontextbetrachtungen werden im Kapitel 7 untersucht, in dem nicht nur das beste Ergebnis des Einzelzeichenerkenners berücksichtigt wird sondern auch die weniger wahrscheinlichen Ergebnisse. Die Frage ist, ob sich die Erkennungsleistung verbessert, wenn mehrere Ergebnisse berücksichtigt werden? Wenn man die Tabelle 4.4 betrachtet, so läßt sich diese Frage eindeutig mit ja beantworten. Dort ist dargestellt, wie sich die Erkennungsleistung auf der Testmenge verhält, wenn die bis zu fünf besten Erkennungsergebnisse berücksichtigt werden. Schon durch Einbeziehen des zweitbesten Ergebnisses erhöht sich die Erkennungsrate auf 94.38%; beim Berücksichtigten der drei besten Ergebnisse erhöht sie sich auf 96.57%. Falls die N-Besten Liste weiter vergrößert wird, dann steigt die Erkennungsleistung weiter an, wobei die Vergrößerung aber nicht mehr so drastisch ausfällt. Somit ließe sich eine hohe Anzahl von Fehlern vermeiden, wenn durch die Kontextbetrachtung das korrekte Ergebnis aus der N-Besten Liste des Einzelzeichenerkenners ermittelt werden kann.

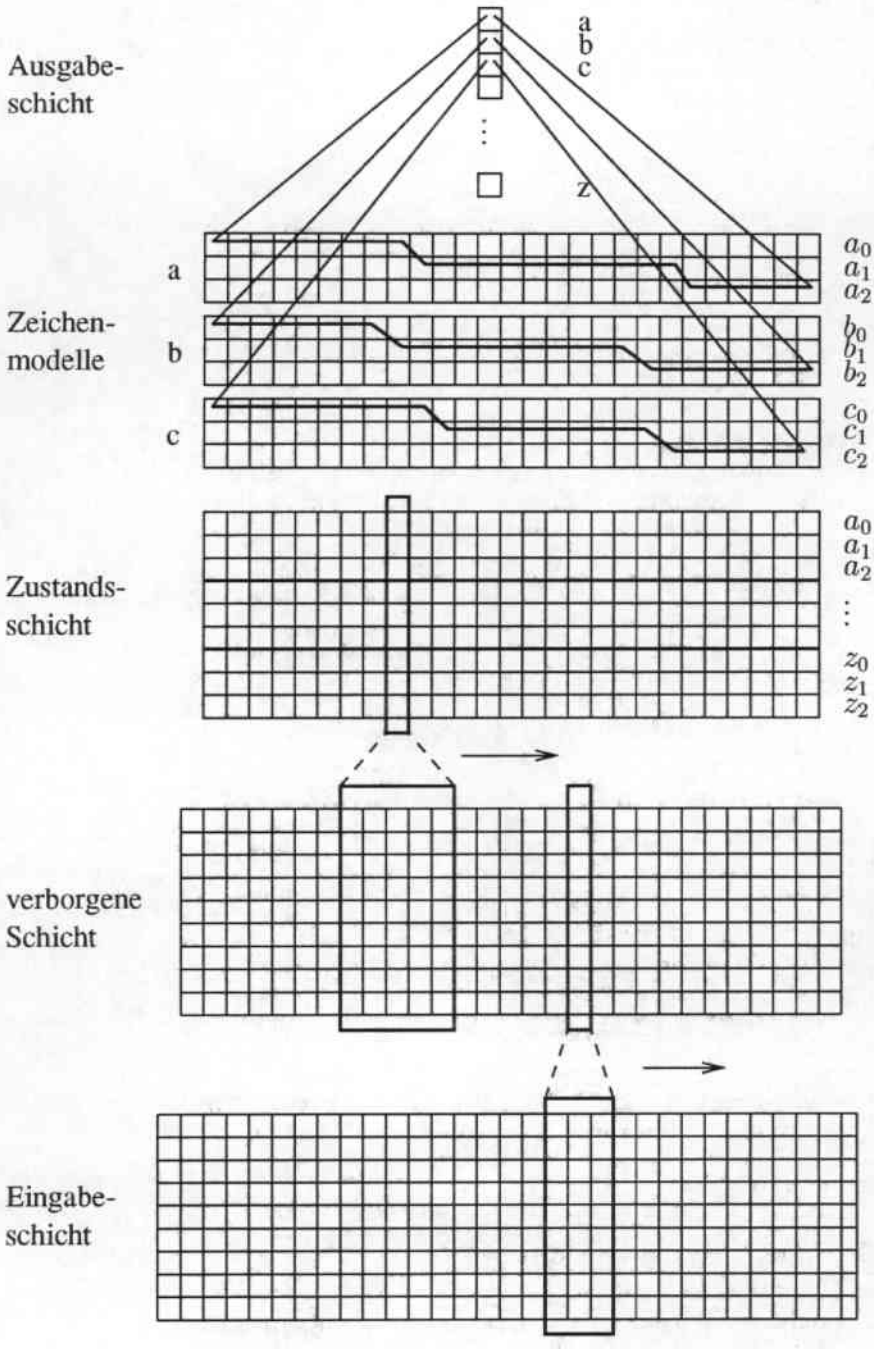

Abbildung 4.3: Allgemeiner Aufbau eines MS-TDNNs

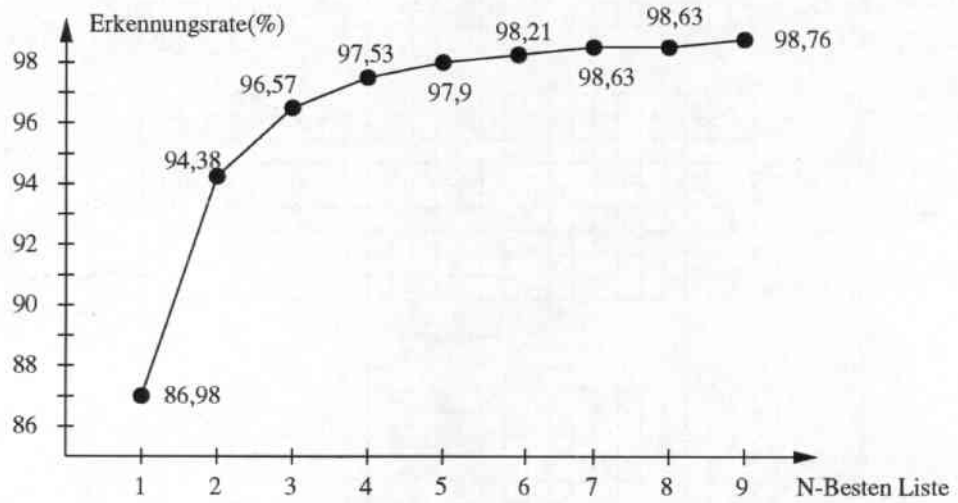

Abbildung 4.4: Erkennungsrate für die N-Besten Liste der Testmenge

Kapitel 5

Strukturanalyse

Alle bisherigen Untersuchungen haben sich mit der Frage beschäftigt, welche Zeichen in der Eingabe vorkommen. Durch den Einsatz des Segmentierers und des Einzelzeichenerkenners wird eine Hypothese erstellt, die diese Frage beantwortet. Hier schließt sich nun die Aufgabe an festzustellen, welche Struktur die Formel besitzt. So müssen z.B. Exponenten und Indices erkannt werden, Zähler und Nenner bei Brüchen bestimmt werden und die obere und untere Grenze bei Summen-/ Integralzeichen gefunden werden. Der Lösungsansatz für dieses Problem ist zweigeteilt: das Bestimmen der Strukur bei mathematischen Operatoren (Bruch, Wurzel, Summe, Integral) erfolgt rein regelbasiert, während das Auffinden von Exponenten/Indices mit Hilfe eines neuronalen Netzes vollzogen wird.

5.1 Mathematische Operatoren

Die Aufgabe der Strukturanalyse bei mathematischen Operatoren besteht darin, mehrere Zeichen zu mathematischen Einheiten zusammenzufassen; z.B. müssen alle Zeichen oberhalb eines Bruchstrichs zur Einheit Zähler zusammengefaßt werden. Bei dieser Arbeit wurden nur die mathematischen Operatoren Bruch, Summe, Integral und Wurzel untersucht. Aber der gewählte Ansatz läßt sich genauso für die Erkennung von Grenzwerten und Produktzeichen verwenden.
Prinzipiell beruhen alle Ansätze der Strukturanalyse [15, 29] auf räumliche Merkmale der Zeichen. Durch zusätzliche Restriktionen bei der Eingabe ist es aber möglich, die Suche nach der korrekten Anordnung zu vereinfachen. So wird in [15] die Eingabereihenfolge fest vorgeschrieben. Dort ist es notwendig, eine Formel von links nach rechts und von oben nach unten zu schreiben. Zwar ist diese Forderung bei den meisten Eingaben erfüllt - bei Brüchen wird meistens der Zähler vor dem Nenner geschrieben -, aber Klammern und die Grenzen bei Summen- und Integralzeichen werden oft am Ende einer Formel geschrieben. Damit solche häufig vorkommenden Schreibweisen nicht ausgeschlossen werden, wird bei die-

sem System keine Restriktion bei der Eingabereihenfolge gemacht.
Um die räumliche Struktur bei mathematischen Operatoren zu bestimmen, wird die Hypothese nach den Operatoren $(-, \sum, \int, \sqrt{})$ durchsucht. Sobald so ein Zeichen gefunden ist, bildet es den Ausgangspunkt für eine genauere Untersuchung.

- Brüche: Da der Einzelzeichenerkenner das Bruch- und das Minuszeichen nur sehr schlecht voneinander unterscheiden kann, werden die beiden Zeichen im Erkenner nur von einem Symbol repräsentiert. Die Entscheidung, welches Zeichen gemeint ist, wird erst in der Strukturanalyse gefällt. Falls sich sowohl Zeichen ober- als auch unterhalb des Striches befinden, dann wird der Strich als Bruchstrich interpretiert, ansonsten als Minuszeichen. Sobald ein Bruchstrich erkannt worden ist, wird bestimmt, in welcher Beziehung die übrigen Zeichen dazu stehen. Dabei wird jedes Zeichen, dessen Mittelpunkt der umgebenden Box oberhalb bzw. unterhalb des Bruchstrichs steht, dem Zähler bzw. Nenner zugeordnet. In diesen neu entstandenen Termen wird dann nach anderen Brüchen weitergesucht, so daß auch die Erkennung von geschachtelten Brüchen möglich ist.

- Wurzeln: Falls ein Wurzelzeichen in der Hypothese vorkommt, ist es notwendig alle Zeichen innerhalb der Wurzel zu bestimmen. Dazu wird untersucht, ob sich der Mittelpunkt der umgebenden Box eines Zeichens innerhalb der umgebenden Box des Wurzelzeichens befindet. Ist dies der Fall, so wird das Zeichen dem Inhalt der Wurzel zugeordnet. Innerhalb des Wurzelterms wird nach weiteren Wurzelzeichen gesucht, so daß es möglich ist, geschachtelte Wurzelausdrücke zu erkennen.

- Summen- und Integralzeichen: Bei Summen- und Integralzeichen müssen die oberen und unteren Grenzen gefunden werden. So wird z.B. ein Zeichen der oberen Grenze zugeordnet, falls sich dessen Mittelpunkt der umgebenden Box oberhalb des Summenzeichens befindet. Entsprechendes gilt für die untere Grenze. Bei diesem Schritt ist es wichtig, daß die zu untersuchenden Zeichen durch die bisher bestimmte Struktur der Formel eingeschränkt sind. Befindet sich z.B. ein Summenzeichen im Nenner eines Bruches, so werden auch nur die Zeichen des Nenners für diese Untersuchungen herangezogen.

- Übrige Terme: In allen anderen Termen, die keine mathematischen Operatoren mehr enthalten, sind die Zeichen bisher nach ihrer Schreibreihenfolge angeordnet. Diese Eigenschaft ist aber unerwünscht, falls z.B. Klammern am Ende der Eingabe der Formel hinzugefügt werden. Deshalb werden die Zeichen innerhalb dieser Terme nach ihrer räumlichen Anordnung von links nach rechts umgeordnet, wobei das Unterscheidungsmerkmal der Mittelpunkt der Zeichens ist. Insgesamt kann damit der Benutzer die Schreibreihenfolge der Zeichen völlig frei wählen und ist nicht wie bei bisherigen Systemen durch Beschränkungen eingeengt.

5.2 Exponenten und Indices

Die Suche nach Exponenten und Indices ist der kompliziertere Schritt der Strukturanalyse, da die Beziehung zwischen allen Zeichen und allen mathematischen Konstrukten untersucht werden muß, wobei es nicht wie im ersten Schritt einen Hinweis zum Start einer Suche gibt. Zwar kann ausgeschlossen werden, daß auf bestimmte Zeichen (+,-,*,=) ein Exponent/Index beginnt, dies grenzt aber den Einsatzbereich eines Formelerkenners ein. Deshalb wird von einer solchen Restriktion kein Gebrauch gemacht. Außerdem soll das System in der Lage sein, geschachtelte Expontenten wie z.B. x^{y^z} zu erkennen.
Die Grundlage des Bestimmens, ob ein Exponent/Index beginnt, ist die Beziehung der umgebenden Boxen zweier Zeichen zueinander. Dieser einfache Ansatz kann aber zu Verwechselungen führen, wie die Abbildung 5.1 zeigt. In beiden Fällen ist die Beziehung der umgebenden Boxen gleich. Aber ein Betrachter neigt dazu, die Zeichen im linken Bild als nebeneinanderstehend zu sehen und im rechten Bild eine Exponentenbeziehung zwischen den Zeichen zu erkennen. Ein Erkenner

Abbildung 5.1: Zweideutige Interpretation bei der Anordnung der umgebenden Boxen

darf also nicht nur aufgrund der Lage der umgebenden Boxen seine Entscheidung treffen, sondern muß zudem den Kontext betrachten.
Um einen Exponenten oder Index zu erkennen, wird in dieser Arbeit ein neuronales Netz verwendet, das auf Merkmalen trainiert wird, die aus den Zeichen gewonnen werden. Bevor diese Lösung detailiert beschrieben wird, werden im nächsten Abschnitt alternative Lösungen vorgestellt.

5.2.1 Weitere Ansätze

Der in [29] vorgestellte Ansatz zeichnet sich besonders dadurch aus, daß bei zweideutigen Anordnungen der Zeichen mehrere Alternativen in die Lösungsmenge aufgenommen werden. Jede alternative Anordnung wird mit einer Wahrscheinlichkeit bewertet, so daß sich eine Rangfolge von der wahrscheinlichsten bis zu der unwahrscheinlichsten Anordnung ergibt. Nach der Strukturanalyse werden alle Alternativen mit Hilfe einer Grammatik auf Plausibilität geprüft und die wahrscheinlichste als Ergebnis genommen. Diese Betrachtung von alternativen Anordnungen wird nicht nur bei der Bestimmung von Exponenten und Indices

verwendet sondern auch bei der Gruppierung von mathematischen Operatoren. Um die Anordnung zwischen 2 Zeichen zu bestimmen, werden die Wahrscheinlichkeiten der drei Möglichkeiten berechnet: liegt ein Exponent oder Index vor oder stehen die Zeichen nebeneinander in Reihe. Diese Berechnung beruht auf der Analyse der räumlichen Anordnung der umgebenden Boxen der beiden Zeichen. Falls eine Wahrscheinlichkeit unter eine gegebene Schwelle fällt, so wird dieser Fall nicht weiter berücksichtigt.
In [24] wird versucht, das Anordnungsproblem durch den Einsatz von Fuzzy-Mengen zu lösen. Dazu werden aus den zu untersuchenden Zeichen 4 Merkmale bestimmt: das Höhenverhältnis, der horizontale Abstand, der vertikale Abstand zwischen den Mittelpunkten der umgebenden Boxen und der vertikale Abstand zwischen den unteren Linien der umgebenden Boxen. Für jede Anordnungsmöglichkeit wird mit Hilfe der Methoden der Fuzzy-Logik aus den Merkmalen eine Wahrscheinlichkeit berechnet und die Anordnung mit dem größten Wert ausgewählt. Problematisch stellt sich bei diesem Ansatz die Suche nach den optimalen Parametern der Fuzzy-Mengen dar, da für dieses Problem kein Lernalgorithmus wie z.B. bei den neuronalen Netzen existiert.

5.2.2 Neuronales Netz

Zwar haben sich die im vorangegangenen Abschnitt beschriebenen Ansätze als erfolgreich erwiesen, trotzdem wird bei dieser Arbeit ein neuronales Netz eingesetzt. Denn es wird erhofft, daß dessen Generalisierungsfähigkeit für weniger Fehlklassifizierungen sorgt. Zusätzlich ist es nicht mehr notwendig, explizite Regeln zu finden. In dieser Arbeit wurde die Anordnung von Exponenten und Indices untersucht, aber dieser Ansatz ist ebenso auf Anordnungen von Zeichen, die über- oder untereinander stehen, erweiterbar.
Für die drei möglichen Anordnungsmöglichkeiten (nebeneinander in Reihe, Exponent, Index) zwischen 2 Zeichen wurden aus den Datenbeständen Beispiele

	Anzahl	Trainingsmenge	Testmenge
Exponenten:	91	65	26
Indices:	82	65	17
Nebeneinander:	955	135	820
Summe:	1128	265	863

Tabelle 5.1: Aufteilung der Trainings- und Testmenge

extrahiert und in eine Trainings- und Testmenge eingeteilt (siehe Tabelle 5.1). Für jedes Beispiel werden Merkmale bestimmt und dem neuronalen Netz als Eingabe zugeführt. Insgesamt werden 5 Merkmale benutzt, die alle charakteristisch für eine Anordnung sind:

- Richtung: Das typischste Merkmal für einen Exponenten oder Index ist, daß er oberhalb bzw. unterhalb seiner Basis steht. Diese Eigenschaft wird als Merkmal übernommen, indem die Richtung des Vektors zwischen den beiden Mittelpunkten der umgebenden Boxen beider Zeichen bestimmt wird.

$$\begin{pmatrix} d_x \\ d_y \end{pmatrix} = \frac{1}{(mx1 - mx2)^2 + (my1 - my2)^2} \begin{pmatrix} mx1 - mx2 \\ my1 - my2 \end{pmatrix}$$

  Falls zwei Zeichen in einer Linie stehen, so wird meistens $d_y = 0$ gelten. Bei Exponenten wird dagegen der Winkel zwischen der Gerade durch die beiden Mittelpunkten und der horizontalen Linie deutlich über 30° liegen.

- Wahrscheinlichkeit für Exponent/Index: Zur Berechnung dieser beiden Merkmale, werden die definierten Größen aus der Abbildung 5.2 benötigt. Um die Wahrscheinlichkeit zu ermitteln, daß ein Exponent vorliegt, wird zunächst die Größe x_1, als der Abstand der unteren Grenze der umgebenden Box des ersten Zeichens zur unteren Grenze der umgebenden Box des zweiten Zeichens, berechnet. Falls dieser Wert negativ ist, wird er auf 0 gesetzt bzw. falls er größer als die Höhe h ist, so wird er auf h zurückgesetzt. Anschließend wird x_1 durch h diviert, so daß gilt $x_1 \in [0, 1]$. Es ist ersichtlich, daß x_1 umso größer wird, je weiter die umgebende Box des zweiten Zeichens nach oben verschoben wird und es damit wahrscheinlicher ist, daß ein Exponent vorliegt.

  Zur Berechnung der Wahrscheinlichkeit, daß eine Indexbeziehung vorliegt,

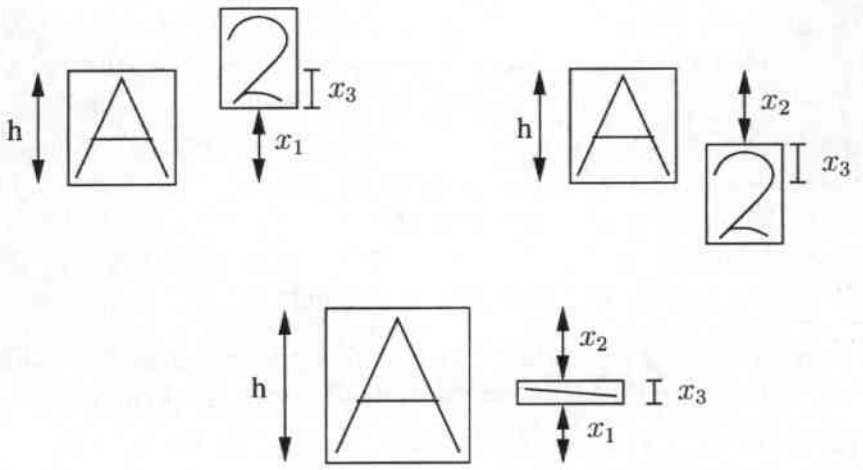

Abbildung 5.2: Verwendete Größen zum Berechnen der Wahrscheinlichkeiten, ob ein Exponent oder index vorliegt.

wird analog vorgegangen. Es wird die Größe x_2 berechnet, auf gültige Werte

überprüft und anschließend normiert. Falls zwei Zeichen in Reihe nebeneinander stehen, so sollten x_1 und x_2 kleine Werte besitzen. Ein besonderer Fall entsteht, wenn ein Zeichen nur eine geringe Höhe besitzt, wie es im unteren Beispiel der Abbildung 5.2 angedeutet ist. Dann erhält sowohl x_1 als auch x_2 einen größeren Wert, die sich aber ungefähr in ihrer Größe entsprechen.

- Überlappung der umgebenden Boxen in Y-Richtung: Für die Berechnung dieses Merkmals wird die in Abbildung 5.2 definierte Größe x_3 benötigt. Sie soll Auskunft darüber geben, inwieweit die beiden umgebenden Boxen der Zeichen nebeneinander stehen. Die Rechenvorschrift für dieses Merkmal lautet $\frac{2*x_3}{h_1+h_2}$, wobei h_1 und h_2 die Höhen der beiden Zeichen sind. Damit ist der Wertebereich dieses Merkmals auf das Interval [0,1] festgelegt. Wenn zwei Zeichen nebeneinander in Reihe stehen, dann tendiert der Wert des Merkmals zu 1. Falls dagegen eine Exponentenbeziehung zwischen den Zeichen vorliegt, dann liegt der Wert des Merkmals näher bei 0.

Auf der Trainingsmenge wurde ein neuronales Netz [5] mit sechs Hidden Units trainiert, dessen Ergebnisse während der Trainingsphase in der Abbildung 5.3 dargestellt sind. Es wird ersichtlich, daß der gewählte Ansatz sehr gut funktioniert. Nach 300 Trainingsiterationen wird auf der Trainingsmenge keine einzige Fehlklassifizierung gemacht. Die Kurve für die Anzahl der Fehler auf der Testmenge ist zwar sprunghaft, aber trotzdem ist die Fehlerquote bei z.B. 11 Fehlern mit 1.2% sehr gering.

Dennoch kann dieses Verfahren noch weiter verbessert werden: denn wie am Anfang des Abschnitts gezeigt wurde, ist eine eindeutige Bestimmung der Anordnung ohne Kontextwissen nicht möglich. Dies wird durch die Tatsache verdeutlicht, daß von den 11 Fehlern nach 500 Trainingsiterationen 10 Fehler bei nebeneinandergeschriebenen Zeichen mit einer unterschiedlichen Schreibhöhe entstehen, z.B. wenn die beiden Zeichen '(' und 'a' miteinander verglichen werden. Um dieses Problem zu vermeiden wird die Merkmalsberechnung nicht nur abhängig von der umgebenden Box der Zeichen gemacht sondern auch von dem Zeichen selbst. Ausgehend von den in der Abbildung 5.4 definierten Linien wird jedes Zeichen genau einer der folgenden Klassen zugeordnet:

- Klasse 1: Klasse der Zeichen, deren Schriftzug sich zwischen Basislinie und Oberlinie befindet. In dieser Klasse sind die meisten Zeichen enthalten wie z.B. 0,...,9, A,...,Z, b, d, h, k, l, t, (,), =, +, *.

- Klasse 2: Klasse der Zeichen, deren Schriftzug sich zwischen Kernlinie und Basislinie befindet. In diese Klasse gehören Zeichen wie z.B. a, c, e, m, n, o, r, s, u, v, w, x, z, -, ∞.

- Klasse 3: Der letzten Klasse werden alle Zeichen zugeordnet, deren Schriftzüge sich zwischen der Kernlinie und der Unterline erstrecken; z.B. g, j, p, q, y.

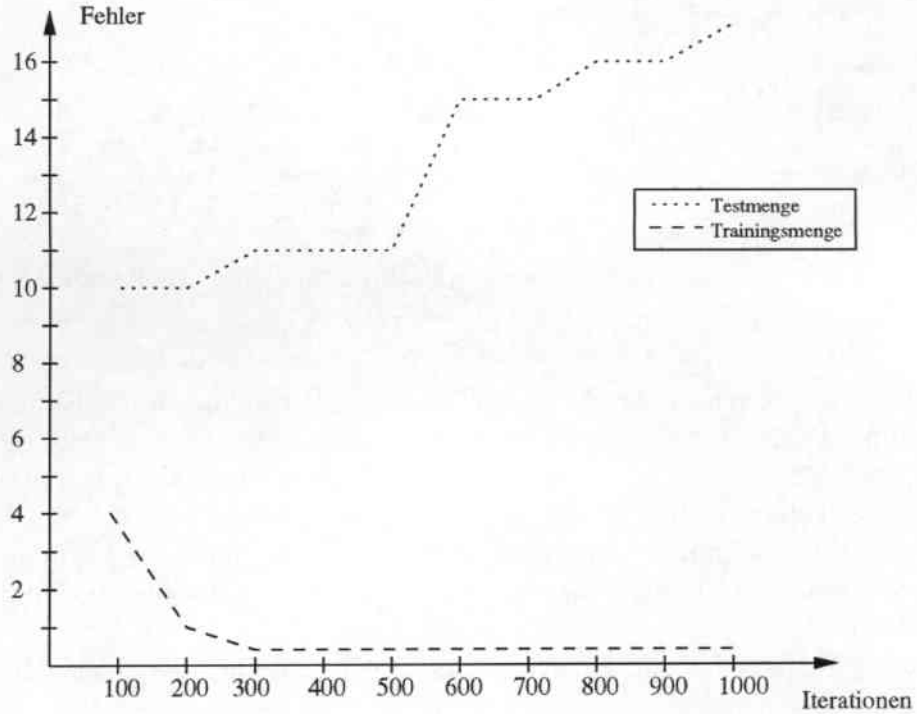

Abbildung 5.3: Fehleranzahl auf der Trainings- und Testmenge in Abhängigkeit von der Anzahl der Iterationen

Falls sich zwei Zeichen, zwischen denen die Anordnung zu bestimmen ist, in der gleichen Klasse befinden, so ist keine weitere Anpassung notwendig. Wenn sie sich dagegen in zwei unterschiedlichen Klassen befinden, so wird versucht, die unterschiedliche Schreibhöhe der verschiedenen Zeichen auszugleichen. Dabei wird die umgebende Box eines Zeichens für die Merkmalsberechnung verkleinert. So fällt bei einem Zeichen der Klasse 1 das obere Viertel der umgebenden Box weg und bei einem Zeichen der Klasse 3 entsprechend das untere Viertel. Auf diese einfache Weise wird die Schreibhöhe verschiedener Zeichen ausgeglichen, und das neuronale Netz kann zum Einsatz kommen. Von den oben genannten ursprünglichen 10 Fehlern lassen sich nach der Korrektur der Schreibhöhe 4 Fehler vermeiden.

5.3 Geschachtelte Exponenten/Indices

Um auch die geschachtelte Anordnung von Exponenten oder Indices zu erkennen, muß jedes Zeichen, egal ob es selbst schon im Exponent oder Index steht, wieder Ausgangspunkt einer neuen Suche nach Exponenten/Indices sein. Für diesen Zweck braucht jedes Zeichen das Kontextwissen, in welcher Stufe des Terms es

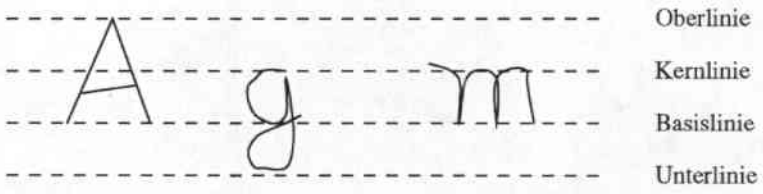

Abbildung 5.4: Definition verschiedener Linien im Schriftbild zur Klassifizierung der Zeichen

steht, so daß ein neu gefundener Exponent auch alle Bedingungen des Kontextes erfüllt. Wenn z.B. der Term a_{b^2} erkannt werden soll, so wird zunächst die Indexbeziehung zwischen a und b ermittelt. Das folgende Zeichen 2 steht mit b in einer Exponentenbeziehung. Damit es aber in den richtigen Kontext eingeordnet wird, muß auch noch die Indexbeziehung mit dem a erfüllt sein. Durch die Verwendung des Kontextwissens lassen sich beliebige Anordnungen von Exponenten und Indices ermitteln.
Die Leistungsfähigkeit bei der Strukturanalyse wird mit den Beispielen in der Abbildung 5.5 demonstriert.

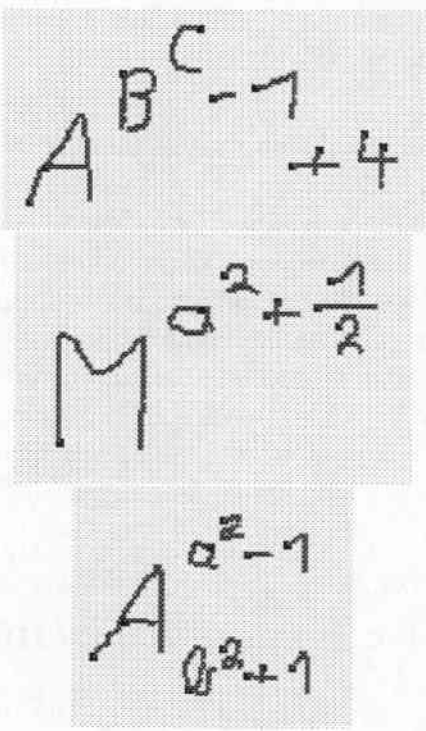

Abbildung 5.5: Beispiel für korrekte Bestimmung der Anordnung

Kapitel 6

Evaluierung

In den bisherigen Kapiteln wurden die einzelnen Kompenten und deren Leistungsfähigkeit vorgestellt. Nun soll die Erkennungsleistung des gesamten Systems geprüft werden. Dazu werden dem Formelerkenner vollständige Formeln zugeführt, so daß die einzelnen Komponenten (Segmentierer, Einzelzeichenerkenner und Strukturanalyse) zusammenarbeiten müssen, um die Formel korrekt zu erkennen.

6.1 Fehlermaß

Damit eine Aussage über die Qualität der Erkennungsleistung des Formelerkenners gemacht werden kann, muß ein geeignetes Fehlermaß verwendet werden. Zwar ist es möglich, nur die korrekt erkannten Formeln als eine erfolgreiche Erkennung zu werten, aber eine solche Berechnung der Fehlerquote ist zu ungenau. Denn wenn z.B. die Formel $\sum_{n=0}^{N} n^2$ als $\sum_{n=0}^{V} n^2$ erkannt wird, dann wurde der größte Teil der Formel bis auf eine Verwechselung beim Einzelzeichenerkenner korrekt erkannt. Falls ein solches Erkennungsergebnis einfach als falsch gewertet wird, so gibt es nicht die tatsächliche Leistungsfähigkeit des Systems wieder. Stattdessen wird die Erkennungsleistung des Systems mit Hilfe der Editierdistanz gemessen. Sie berücksichtigt bei ihrer Ermittlung, daß bei der Erkennung von Formeln neben der Verwechselung von Zeichen auch durch Segmentierungsfehler Einfügungen und Auslassungen von Zeichen auftreten können. Die Fehlerrate ergibt sich damit, als

$$Fehlerrate = \frac{\#Einfügungen + \#Auslassungen + \#Verwechselungen}{\#Gesamt},$$

wobei #Gesamt angibt, wieviele Zeichen insgesamt in der Referenzformel vorkommen. Anschaulich kann die Editierdistanz als die Anzahl der Operationen Einfügen, Löschen und Vertauschen angesehen werden, die notwendig sind, um

die Hypothese in die Referenzformel zu überführen.
Sowohl die Hypothese als auch die Referenzformel werden mit Hilfe des Textformatierungssystems LaTeX [14] dargestellt. So entspricht z.B. die Formel $u_i + v_i = w_i$ der Beschreibung u _ {i} + v _ {i} = w _ {i} in LaTeX. Falls nun durch den Erkennungsprozeß die Hypothese u i + v _ {i} = w _ {i} erzeugt wird, so liefert die Editierdistanz eine Fehlerrate von 3/17. Prinzipiell sind sogar Fehlerraten von über 100% möglich, falls z.B. mehr Zeichen in die Hypothese eingefügt wurden als in der Referenzformel vorhanden sind.

6.2 Ergebnisse

Für die Evaluierung des Systems wurden von jedem der 28 Schreiber aus der Datenbasis 39 Formeln verwendet. Folglich wird der Test mit fast 1100 Formeln durchgeführt, die etwa 13.500 Zeichen umfassen. Sie lassen sich in fünf Kategorien einteilen: in der ersten Kategorie sind alle Formeln enthalten, in denen keine mathematischen Sonderzeichen wie Brüche, Wurzeln oder Summenzeichen vorkommen. Auch Exponenten und Indices sind darin enthalten. Die zweite Kategorie entspricht den Formeln mit Brüchen und die dritte Kategorie den Formeln mit Wurzeln. Formeln, die Summenzeichen enthalten, sind in der Kategorie vier zusammengefaßt und Formeln mit Integralen in der letzten Kategorie. Dabei gilt, daß der Umfang der zu erkennenden mathematischen Anordnungen zusammen mit der Kategorienummer steigt; d.h., daß in der Kategorie 4 auch Wurzelterme, Brüche und Exponenten vorkommen.
Die Unterteilung in die einzelnen Kategorien kann bei der Evaluierung benutzt werden, damit die Leistungsfähigkeit des Formelerkenners in den einzelnen Aufgabenbereichen dargestellt werden kann. So ist in der Tabelle 6.1 für jede Kategorie die Fehlerrate und die Anzahl der zuerkennenden Zeichen abgebildet. Insbesondere fällt die schlechte Erkennungsleistung bei den Formeln mit Sum-

		#Zeichen	Fehlerrate
Kategorie 1	$(+, -, *, a_i \ldots)$	4328	26.20%
Kategorie 2	(a/b)	2794	22.58%
Kategorie 3	$(sqrt a)$	1829	21.97%
Kategorie 4	$(\sum_{i=1}^{N})$	2587	36.21%
Kategorie 5	$(\int_a^b)$	1844	24.89%
Insgesamt		13.382	26.62%

Tabelle 6.1: Fehlerraten in den einzelnen Kategorien (siehe Text)

menzeichen (Kategorie 4) auf. Ein Grund dafür liegt darin, daß im Allgemeinen die unteren Summengrenzen sehr klein und undeutlich geschrieben werden

und damit diese Zeichen schwer zu erkennen sind. Neben diesem grundsätzlichen Problem treffen hier mehrere Probleme aufeinander, die im folgenden erläutert werden sollen:

- Ein Problem entsteht bei der korrekten Segmentierung des 'i-Zeichens', das gerade bei den Summenausdrücken häufig vorkommt. Da der 'i-Punkt' meistens weit entfernt vom 'i-Strich' geschrieben wird, werden die beiden Strokes oft als getrennte Zeichen segmentiert. Da zudem der 'i-Punkt' oberhalb des 'i-Strichs' steht, wird eine Exponentenbeziehung zwischen den Zeichen ermittelt, so daß der Erkenner die Sequenz $'i - Strich' \ \hat{} \ \{ \ 'i - Punkt' \ \}$ liefert. Damit können bei einem einzigen 'i-Zeichen' bis zu 5 Fehler entstehen. In der Kategorie 4 tritt dieser Fall bei insgesamt 34.3% der 'i-Zeichen' auf. Das bedeutet, daß 19% der Fehler in dieser Kategorie auf Segmentierungsfehler beim 'i-Zeichen' zurückzuführen sind.
  Diese hohe Fehleranzahl beim 'i-Zeichen' ist eindeutig zu groß, so daß es sinnvoll ist, an dieser Stelle das System zu verbessern. Eine Möglichkeit, dieses Problem zu lösen, wird in [29] beschrieben. Dort wird in das System vor den Segmentierer eine Prerecognition Stage eingefügt. Sie wird benutzt, um 'i-Zeichen' zu erkennen und dessen Strokes unabhängig von dem Ergebnis des Segmentierers als zusammengehörig zu bewerten.

- Ein weiterer typischer Fehler entsteht häufig dann, wenn das $\sum$-Zeichen mit 2 Strokes geschrieben wird: zuerst der obere horizontale Strich und danach in einem Zug der untere Rest. Eine solche Schreibweise wird vom Segmentierer meistens als 2 getrennte Zeichen interpretiert. Dadurch entstehen aber einige Folgefehler: der horizontale Strich wird als Bruchstrich erkannt, in dessen Nenner der untere Teil des $\sum$-Zeichens steht, der wiederum selber im Index die urprüngliche untere Summengrenze enthält. Damit bekommt die Ausgabe des Erkenners eine völlig andere Struktur als die Referenzformel, woraus eine hohe Fehlerrate resultiert.

Im folgenden werden nun typische Fehler vorgestellt, die in allen Kategorien häufig auftreten. So fällt insbesondere auf, daß insgesamt über 33% der Fehler auf Verwechselungen des Einzelzeichenerkenners zurückzuführen sind: 44.4% (Kategorie 1), 40.25% (Kategorie 2), 34.8% (Kategorie 3), 15.68% (Kategorie 4) und 30.0% (Kategorie 5). An dieser Stelle ließe sich die Erkennungsleistung des Systems durch einen besser trainierten Einzelzeichenerkenner deutlich verbessern. Dies ist z.B. möglich, indem eine größere Datenbasis aufgebaut wird, so daß der Einzelzeichenerkenner mehr Trainingsmaterial bekommen kann.
Ein anderes häufig auftretendes Problem besteht in der korrekten Segmentierung des Gleichheitszeichen. Zwar wird durch einen Korrekturschritt (siehe Seite 19) versucht, den Fehler auszugleichen, aber das Verfahren greift nur, wenn beide

Strokes des Gleichheitszeichen als Minuszeichen erkannt werden. Sobald z.B. der obere Stroke als Wurzelzeichen erkannt wird, liefert das System eine Ausgabe der Form: $-$ ^ { $\surd$ }. Auf diese Weise werden durch die Editierdistanz fünf Fehler erzeugt. Der prozentuale Anteil dieser Art von Fehler an der Gesamtfehlerzahl in den einzelnen Kategorien beträgt: 6.1% (Kategorie 1), 2.3% (Kategorie 2), 7.4% (Kategorie 3), 14.4% (Kategorie 4) und 0.0% (Kategorie 5). Gerade bei den Formeln mit Summenausdrücken (Kategorie 4) sind viele Gleichheitszeichen enthalten, so daß der Fehleranteil mit 14.4% besonders hoch ist. Insgesamt läßt sich sagen, daß es notwendig ist, die Segmentierung bei dem Gleichheitszeichen zu optimieren, um eine bessere Erkennungsleistung zu bekommen. Eine Möglichkeit besteht darin, ein geeignetes Merkmal für das neuronale Netz des Segmentierers zu ermitteln, welches typisch für Gleichheitszeichen ist; oder es wird versucht, das neuronale Netz besser zu trainieren, indem mehr Trainingsbeispiele des Gleichheitszeichens verwendet werden.

6.3 Gewichtung Einzelzeichenerkenner/ Segmentierer

Beim Segmentieren einer Eingabe wird ein Symbol-Hypothesen-Netz erstellt, in dem die möglichen Zusammenfassungen der Strokes (Strokegruppe) zu einem Zeichen bewertet werden. Jede Bewertung einer Strokegruppe setzt sich aus zwei Einzelbewertungen zusammen: die des Einzelzeichenerkenners und die des Segmentierers. Die Gewichtung dieser beiden Werte wird durch den Parameter δ gesteuert. Damit ergibt sich die Bewertung einer Strokegruppe als $P(m,g) = \delta * E(m,g) + (1-\delta) * S_{m,g}$ (siehe Seite 14).
An dieser Stelle soll die Frage geklärt werden, welcher der optimale Wert für den Parameter δ ist. In der Abbildung 6.1 sind die Fehlerraten aller Kategorien zusammen, der Kategorie 1,2,3,5 zusammen und der Kategorie 4 alleine für verschiedene Einstellungen des Parameters δ dargestellt. Eine gesonderte Betrachtung der Kategorie 4 ist notwendig, da sich der Verlauf der Fehlerrate zu der der anderen Kategorien unterscheidet.
Zunächst wird ersichtlich, daß sich die Hinzunahme des Einzelzeichenerkenners in den Segmentierungsprozeß lohnt. Falls auf eine Bewertung des Einzelzeichenerkenners verzichtet wird, so erzielt das System eine Fehlerrate von 30.58%. Wenn er dagegen in den Segmentierungsprozeß einbezogen wird, so reduziert sich die Fehlerrate auf 26.62% ($\delta = 0.9$). Wird der Parameter δ auf 1.0 gestellt, so daß die Segmentierung alleine durch den Einzelzeichenerkenner erfolgt, so verschlechtert sich die Fehlerrate auf 28.63%.
Im Gegensatz zu den anderen Kategorien verbessert sich die Fehlerrate in der Kategorie 4 umso mehr, je höher der Parameter δ gestellt wird. Dies hängt damit

zusammen, daß hier einige Zeichen z.B. i,=,$\sum$ vorkommen, mit denen der Segmentierer Schwierigkeiten hat und somit dessen Bewertung schlecht ist. Dagegen liefert der Einzelzeichenerkenner in diesen Fällen eine hohe Bewertung für diese Strokegruppen. Deshalb fällt die Fehlerrate immer mehr, je weniger die Bewertung des Segmentierers in die Gesamtbewertung der Strokegruppe einfließt. Die beste Fehlerrate in der Kategorie 4 wird dann erreicht, wenn auf die Bewertung des Segmentierers verzichtet wird.

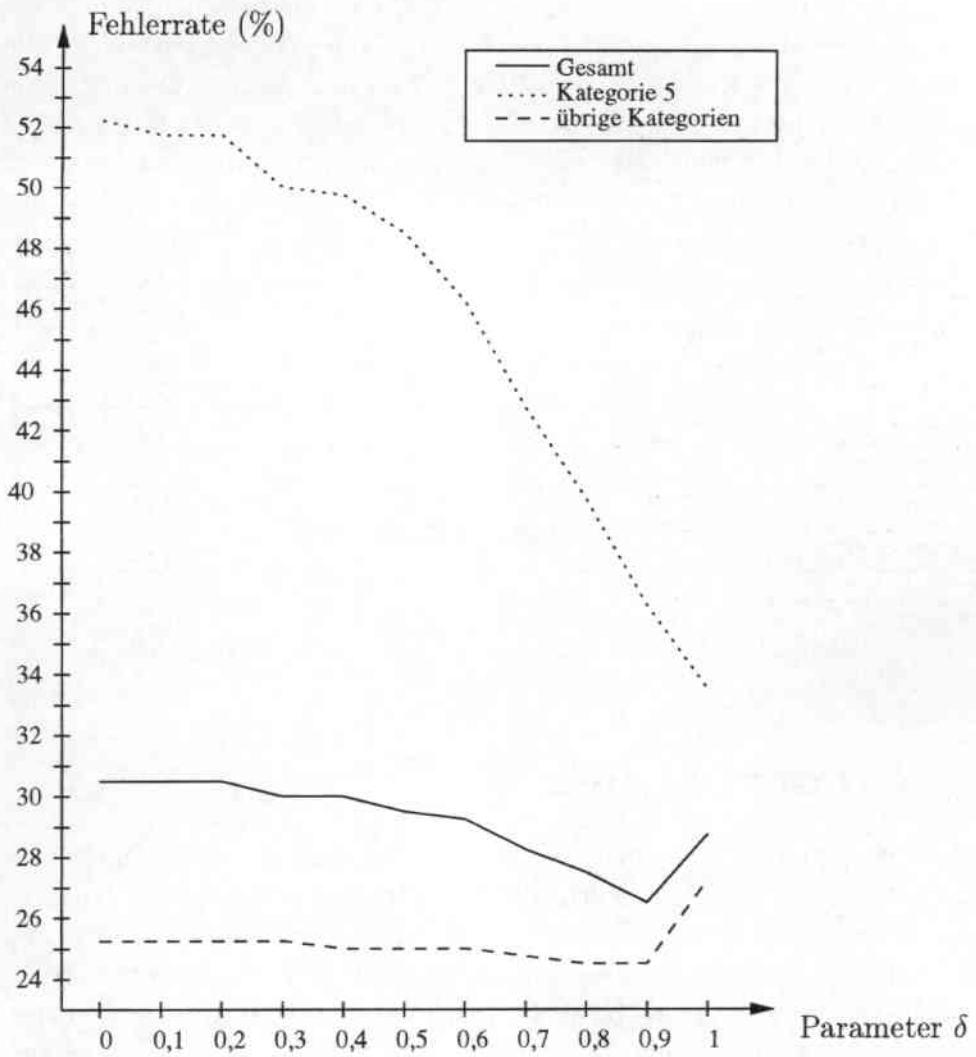

Abbildung 6.1: Verschiedene Einstellungen des Parameters δ zur Gewichtung des Einzelzeichenerkenners und des Segmentierers.

6.4 Korrektur beim Gleichheitszeichen

Schon bei der Analyse der Fehler des Segmentierers fiel auf, daß die korrekte Segmentierung von Gleichheitszeichen ein schwieriges Problem darstellt. Deshalb wird, wie auf Seite 19 beschrieben, in einem Korrekturschritt die Hypothese nach zwei hintereinanderstehenden Minuszeichen durchsucht. Falls sie bestimmte Bedingungen erfüllen, werden sie durch ein Gleichheitszeichen ersetzt.
Wie aus der Tabelle 6.2 ersichtlich wird, führt dieser Korrekturschritt zu einer deutlich verbesserten Erkennungsleistung. Besonders in der Kategorie 4, in der viele Gleichheitszeichen auftreten, kann die Fehlerrate von 46.42% auf 36.21% reduziert werden. In allen anderen Kategorien kommen nicht so viele Gleichheitszeichen vor, trotzdem profitieren auch sie von dem Korrekturschritt, wie die geringeren Fehlerraten zeigen. Insgesamt reduziert sich die Fehlerrate von 30.72% auf 26.62%, wodurch gezeigt ist, daß der Korrekturschritt hilft, eine hohe Anzahl von Fehlern zu vermeiden, und er somit berechtigterweise zum Einsatz kommt.

	ohne Korrektur	mit Korrektur
Kategorie 1	29.89%	26.20%
Kategorie 2	23.94%	22.58%
Kategorie 3	25.69%	21.97%
Kategorie 4	46.42%	36.21%
Kategorie 5	25.92%	24.89%
Insgesamt	30.72%	26.62%

Tabelle 6.2: Fehlerraten mit und ohne Korrekturschritt

6.5 N-Besten Liste

Sowohl der Segmentierer als auch der Einzelzeichenerkenner liefern nicht nur das beste Ergebnis zurück sondern auch mit Bewertungen versehene, alternative Ergebnisse. Diese alternativen Ergebnisse werden in dem Symbol-Hypothesen-Netz berücksichtigt, so daß das System als Ergebnis eine bewerte Liste der N besten Erkennungsergebnisse erzeugt. Diese Liste wird von einer Komponente benötigt, die am Ende des Erkennungsprozesses eingesetzt wird: das Language Model. Es prüft die Hypothesen z.B. mit Hilfe einer Grammatik, ob sie plausibel sind bzw. ob deren Struktur einer mathematischen Formel entspricht. Jede Hypothese wird durch das Language Model bewertet, so daß sich anschließend die Reihenfolge in der N-Besten Liste verändern kann. An dieser Stelle wird nun untersucht, welche Verbesserung in der Erkennungsleistung diese Komponente maximal erreichen kann.

In der Abbildung 6.2 sind die Fehlerraten in den einzelnen Kategorien dargestellt, wenn die 10 besten Hypothesen des Segmentierers berücksichtigt werden. Alternative Vorschläge des Einzelzeichenerkenners sind bei dieser Untersuchung noch nicht eingeflossen. Aus der Abbildung wird deutlich, daß die Fehlerraten in allen Kategorien fallen, wenn die beste Hypothese aus der N-Besten Liste herausgenommen werden kann. Dabei gilt, daß die Fehlerrate umso mehr fällt, je mehr Hypothesen berücksichtigt werden. Insbesondere verbessert sich die Fehlerrate bei der Kategorie 4, da hier ursprünglich besonders viele falsche Segmentierungen vorlagen. Insgesamt könnte das Language Model, wenn es nur die verschiedenen Segmentierungen untersucht, eine Verbesserung der Erkennungsleistung von 26.6% auf 20.38% erzielen.

Eine weitere Untersuchung soll prüfen, welche Verbesserung in der Erkennungs-

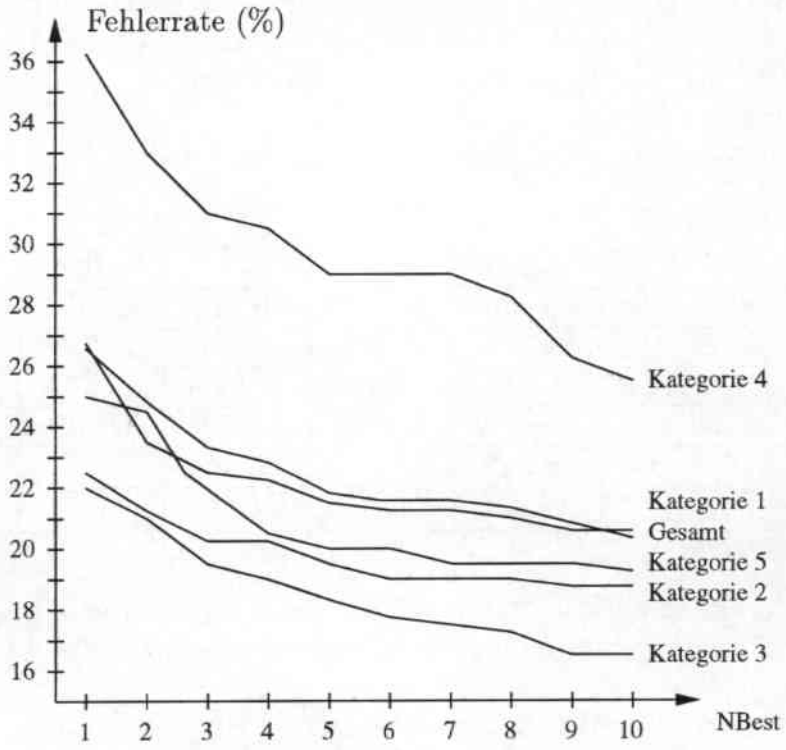

Abbildung 6.2: Die Fehlerraten für die N-Besten Liste des Segmentierers.

leistung erreicht werden kann, falls alternative Ergebnisse des Einzelzeichenerkenners zugelassen werden. Da bei den am besten bewerteten Hypothesen des Erkenners 33% der Fehler auf Fehlklassifizierungen des Einzelzeichenerkenners zurückzuführen sind, ist hier ein deutlicher Abfall bei der Fehlerrate zu erwarten. Bei diesem Versuch wird für jede Eingabe immer nur die höchst bewertete Segmentierung verwendet. Dafür werden bei der Hypothesenbildung die 3 besten

Ergebnisse des Einzelzeichenerkenners berücksichtigt; d.h. vom Start- zum Endpunkt des Symbol-Hypothesen-Netzes (siehe Seite 15) existiert nur ein einziger Pfad. Alle Hypothesen die entlang dieses Pfades erzeugt werden unterscheiden sich also nur darin, daß sie alternative Ergebnisse des Einzelzeichenerkenners beinhalten. Insgesamt entsteht auf diese Weise für jede Eingabe eine Liste der N besten Hypothesen.
In der Tabelle 6.3 sind die Fehlerraten dieser Hypothesen dargestellt. Es wird deutlich, daß die Fehlerraten in allen Kategorien sinken, je mehr Hypothesen berücksichtigt werden. Sie reduzieren sich auf 29.5% (Kategorie 1), 20.6% (Kategorie 2), 18.9% (Kategorie 3), 11.8% (Kategorie 4) und 13.3% (Kategorie 5). Somit läßt sich schlußfolgern, daß das System davon profitiert, wenn die alternativen Ergebnisse des Einzelzeichenerkenners einbezogen werden. Trotzdem reicht diese Vorgehensweise noch nicht aus, um alle Fehler auszugleichen, die durch Verwechselungen des Einzelzeichenerkenners entstehen. Denn bei einem fehlerfreien Einzelzeichenerkenner müßten die Fehlerraten viel stärken fallen: um 44.4% (Kategorie 1), um 40.25% (Kategorie 2), um 34.8% (Kategorie 3), um 15.68% (Kategorie 4) und um 30.0% (Kategorie 5).
In einem letzten Versuch wird die Erkennungsleistung auf einer N-Besten Liste

	1	2	5	10	15	20
Kategorie 1	26.20%	25.06%	21.90%	20.74%	19.5%	18.9%
Kategorie 2	22.58%	21.33%	19.97%	19.32%	18.71%	17.93%
Kategorie 3	21.97%	20.77%	19.4%	18.64%	18.37%	17.82%
Kategorie 4	36.21%	35.48%	34.26%	33.05%	32.76%	31.96%
Kategorie 5	24.89%	24.29%	22.83%	22.39%	22.34%	21.58%
Insgesamt	26.62%	25.38%	23.67%	22.82%	22.3%	21.6%

Tabelle 6.3: Fehlerraten auf der N-Besten Liste der best bewerteten Segmentierung mit alternativen Ergebnissen des Einzelzeichenerkenners.

untersucht, die sowohl die Alternativen des Segmentierers als auch des Einzelzeichenerkenners beinhaltet. In diesem Falle werden die Hypothesen mit dem Symbol-Hypothesen-Netz auf die Weise erstellt, wie sie in Kapitel 2 auf Seite 15 eingeführt worden sind. Die Fehlerraten für diesen Versuch sind in der Tabelle 6.4 dargestellt.
Es wird deutlich, daß sich die Erkennungsleistung des Systems in allen Kategorien nochmals verbessert, obgleich die Steigerung zu den vorhergehenden Versuchen nur noch gering ausfällt. Der Einsatz eines Language Models könnte die Fehlerrate von 26.62% auf maximal 18.10% reduzieren. Diese Möglichkeit der Verbesserung ist so gravierend, daß auf ein Language Model, mit dessen Hilfe die beste Hypothese aus der N-Besten Liste gewählt wird, nicht verzichtet werden kann.

	1	2	5	10	15	20
Kategorie 1	26.20%	25.43%	21.95%	19.52%	16.98%	16.26%
Kategorie 2	22.58%	20.61%	19.18%	18.03%	17.03%	15.71%
Kategorie 3	21.97%	20.06%	17.44%	15.14%	14.54%	13.55%
Kategorie 4	36.21%	34.48%	32.52%	30.45%	28.10%	26.63%
Kategorie 5	24.89%	24.18%	22.34%	21.47%	19.46%	18.43%
Insgesamt	26.62%	24.95%	22.69%	20.92%	29.22%	18.10%

Tabelle 6.4: Fehlerraten auf der N-Besten Liste der best bewerteten Segmentierung mit alternativen Ergebnissen des Einzelzeichenerkenners.

Kapitel 7

Sprachmodell

Durch den Einsatz eines Sprachmodells soll gewährleistet werden, daß Hypothesen bevorzugt werden, deren Formelstruktur eine syntaktisch und semantisch korrekte, mathematische Struktur besitzen. In allen modernen Erkennungssystemen wird die letztendliche Entscheidung für eine Hypothese nicht nur von der Bewertung eines Mustervergleichs abhängig gemacht sondern auch von einer Bewertung des Language Models. Dies drückt die Formel von Bayes (7.1) aus: die Entscheidung fällt zugunsten der Wortfolge $\hat{W} = w_1, \ldots, w_n$, dessen Produkt - aus der Wahrscheinlichkeit von der Beobachtung A erzeugt zu werden und der a-priori Wahrscheinlichkeit der Wortfolge - maximal ist.

$$P(\hat{W}|A) = \max_W P(W|A) = \frac{P(W)P(A|W)}{P(A)} \tag{7.1}$$

Die meisten Systeme im Bereich der Formelerkennung arbeiten mit Grammatiken, um das Language Model zu modellieren. In dieser Arbeit soll dagegen der Versuch unternommen werden, das Language Model mit Hilfe von n-Gramm Modellen zu erstellen. Dies hat den Vorteil, daß die aufwendige Suche nach Produktionsregeln einer Grammatik entfällt. Zusätzlich können auch Formeln erkannt werden, die keine korrekte mathematische Struktur besitzen; d.h., falls ein Benutzer eine syntaktisch falsche Formel eingibt, so wird sie von dem n-Gramm Modell akzeptiert. Dagegen müßte bei einem Sprachmodell, das mit einer Grammatik arbeitet, zunächst weitere Produktionsregeln entworfen werden, bevor eine solche Formel mit falscher Struktur erkannt werden kann.

7.1 Grammatiken

Bei der Erkennung mathematischer Formeln wurden in den meisten Arbeiten Parser eingesetzt, die mit Hilfe einer Grammatik prüfen, ob die eingegebene Formel eine korrekte Syntax besitzt. Für die Formelerkennung sind kontextfreie Gramma-

tiken interessant, da für sie Parsing-Algorithmen bereitstehen, die polynomialen Aufwand besitzen: z.B. der Cocke-Younger-Kasami Algorithmus, der einen kubischen Aufwand in der Anzahl der Buchstaben besitzt.
In [3] wird vorgeschlagen, als Lösung eine Definite Clause Grammar zu verwenden. Hierbei handelt es sich um einen top-down Parser, der rechtsrekursive Grammatiken mit Hilfe des Backtracking-Verfahrens bearbeiten kann; d.h. immer wenn der Parsing-Algorithmus keine Produktionsregel mehr anwenden kann, wird der letzte Schritt rückgängig gemacht. Die dann erhaltene reduzierte Teillösung wird versucht, auf einen andern Weg zu lösen. Problematisch ist bei diesem Ansatz, daß sehr oft Schritte zurückgenommen werden müssen, so daß der Algorithmus nicht effizient ist. Deshalb werden Optimierungsvorschläge für die Produktionsregeln der Grammatik vorgestellt, um einen effizienten Parsing-Algorithmus zu erhalten.
Eine bessere Möglichkeit zum Parsen stellen stochastische kontextfreie Grammatiken dar. Hier wird jeder Produktionsregel einer Grammatik eine Wahrscheinlichkeit zugewiesen, so daß allen zulässigen Ausdrücken eine Wahrscheinlichkeit ungleich 0 zugewiesen werden kann. Zusätzlich ist es möglich, auch syntaktisch inkorrekte Formeln zu erkennen, wenn weitere Produktionsregeln eingeführt werden. Da diese Produktionsregeln nur selten aktiv werden sollen, wird ihnen nur eine geringe Wahrscheinlichkeit zugeordnet. Beispiele für diese Vorgehensweise werden in [20] und [24] erläutert.

7.2 n-Gramm Modell

In diesem Abschnitt werden die Grundlagen der n-Gramm Modelle [10] und die Möglichkeiten der Parameteroptimierung vorgestellt, falls nur wenige Datenmengen vorhanden sind [2].
Die Aufgabe des Language Models besteht darin, die Wahrscheinlichkeit für eine Wortfolge $W = w_1, \ldots, w_n$ zu bestimmen. Formal ausgedrückt läßt sich diese Wahrscheinlichkeit wie folgt berechnen:

$$\begin{aligned} P(W) &= \prod_{i=1}^{n} P(w_i|w_1, \ldots, w_{i-1}) \\ &= P(w_1) * P(w_2|w_1) * \ldots * P(w_i|w_1, \ldots, w_{i-1}), \end{aligned}$$

wobei $P(w_i|w_1, \ldots, w_{i-1})$ die Wahrscheinlichkeit ist, daß dem Wort w_i die Wortfolge $w_1, \ldots, w_{i-1}$ vorangegangen ist. Damit wird angenommen, daß die Wahl eines Wortes w_i vollständig abhängig ist von der gesamten Vergangenheit der Wortfolge. Diese Forderung kann nicht gehalten werden, da eine vernünftige Schätzung von $P(W)$ für alle möglichen Wortfolgen undenkbar ist. Statt alle vorangegangenen Wörter in die Wahrscheinlichkeitsberechnung einzubeziehen, wird immer nur ein Fenster der Länge n betrachtet. Dies stimmt mit der Einschätzung überein,

daß der statistische Einfluß eines Wortes der jüngeren Vergangenheit mit Sicherheit höher einzustufen ist als derjenige eines Wortes der ferneren Vergangenheit. Solche Sprachmodelle, die mit der Approximation

$$P(w_i|w_1,\ldots,w_{i-1}) \approx P(w_i|\underbrace{w_{i-n+1},\ldots,w_{i-1}}_{n-1})$$

arbeiten, werden als statistische n-Gramm-Grammatiken bezeichnet. Besonders die Spezialfälle der Unigramm-, Bigramm- und Trigramm-Modelle haben eine große Verbreitung gefunden:

$$\begin{aligned} P(w) &= \prod_{i=1}^{n} P(w_i) \\ P(w) &= P(w_1) * \prod_{i=2}^{n} P(w_i|w_{i-1}) \\ P(w) &= P(w_1) * P(w_2|w_1) * \prod_{i=3}^{n} P(w_i|w_{i-2}w_{i-1}) \end{aligned}$$

Um die Wahrscheinlichkeit für ein n-Gramm $P(w_i|w_{i-n+1},\ldots,w_{i-1})$ zu schätzen, wird normalerweise auf einer großen Textbasis gezählt, wie oft das Wort w_i nach der Wortfolge $w_{i-n+1},\ldots,w_{i-1}$ auftritt und dieser Wert durch die Häufigkeit des Auftretens der Wortfolge $w_{i-n+1},\ldots,w_{i-1}$ dividiert.

$$P(w_i|w_{i-n+1},\ldots,w_{i-1}) = \frac{C(w_{i-n+1},\ldots,w_i)}{C(w_{i-n+1},\ldots,w_{i-1})}$$

Dabei bezeichnet $C(w)$ wie oft eine Wortfolge in der Trainingsmenge vorkommt. Zwar läßt sich durch Vergrößern der Ordnung n eine Sprache immer exakter approximieren, aber dieser Vorgehensweise ist in der Praxis das dramatische Wachstum der Modelldimension entgegengestellt. Denn eine Trigramm-Grammatik mit einem Sprachumfang von 1000 Wörtern beinhaltet bereits 1 Milliarde statistischer Parameter. Deshalb besteht Bedarf an wirkungsvollen Glättungs- und Interpolationsverfahren zur Größenreduzierung der Sprachmodelle.

7.2.1 Äquivalenzklassenbildung

Ein mögliches Verfahren, das Sprachmodell zu verkleinern, besteht darin, die Wörter des Wortschatzes in Äquivalenzklassen einzuteilen. In diesem Fall berechnet sich die Wahrscheinlichkeit einer Wortfolge aus der Wahrscheinlichkeit für die Folge der zugehörigen Äquivalenzklassen:

$$P(W) = \prod_{i=1}^{n} P(S(w_i)|S(w_{i-n+1},\ldots,S(w_{i-1}))),$$

wobei $S(w_{i-n+1}, \ldots, w_{i-1})$ die Äquivalenzklasse für die Wortfolge $w_{i-n+1}, \ldots, w_{i-1}$ bezeichnet. Zum Beispiel ist es bei einem Sprachmodell für Formeln naheliegend, eine Äquivalenzklasse für alle Zahlen einzuführen. Es wird angenommen, daß eine solche Vereinfachung die Leistungsfähigkeit eines Sprachmodells nicht beeinflußt. Bei diesem Ansatz ist es wichtig, geeignete Äquivalenzklassen zu finden. Zwar können die Klassen auch per Hand ermittelt werden, aber besser ist eine automatische Vorgehensweise, wie sie in [10] vorgeschlagen wird. Basierend auf der Definition der Mutual Information Funktion kann ein Ausdruck bestimmt werden, der für eine Klasseneinteilung in M Äquivalenzklassen maximiert werden muß:

$$\hat{I} = \sum_{i=1}^{L} \sum_{j=1}^{M} C(g_j, w_i) \log \frac{C(g_j, w_i)}{C(g_j)},$$

wobei L die Größe des Vokabulars angibt und $C(g_j, w_i) = \sum_{w_k \in g_j} C(w_k, w_i)$ die Wahrscheinlichkeit beschreibt, daß ein Wort w_i einem Wort aus der Äquivalenzklasse g_j folgt. Mit einem agglomerativen Ballungsverfahren läßt sich iterativ eine Klasseneinteilung bestimmen.

7.2.2 Interpolations- und Glättungsverfahren

Die einfachste Variante die Trigrammwahrscheinlichkeiten zu ermitteln, besteht darin, die Häufigkeit der Wortfolge $w_{i-2}w_{i-1}w_i$ in einem großen Text durch die Häufigkeit der Wortfolge $w_{i-2}w_{i-1}$ zu dividieren:

$$P(w_i|w_{i-2}w_{i-1}) = \frac{C(w_{i-2}w_{i-1}w_i)}{C(w_{i-2}w_{i-1})}$$

Falls ein Language Model auf diese Weise erstellt wird, so besitzt es nur eine geringe Leistungsfähigkeit. Denn alle Trigramme $w_{i-2}w_{i-1}w_i$, die in der Trainingsmenge nie aufgetaucht sind, werden durch das Language Model mit $P(w_i|w_{i-2}$ $w_{i-1}) = 0$ bewertet. Diese Wahrscheinlichkeit von 0 wird immer verhindern, daß diese Wortfolge erkannt wird, unabhängig davon wie eindeutig die Bewertung für diese Wortfolge durch den Mustervergleich ausgefallen ist.
Durch den Einsatz von Glättungsverfahren soll dieses Problem umgangen werden, indem die geschätzen Wahrscheinlichkeiten angepaßt werden. Dabei versuchen die Verfahren, die Wahrscheinlichkeitsverteilungen gleichförmiger zu gestalten, indem geringe Wahrscheinlichkeitswerte wie z.B. $P(w_i|w_{i-2}w_{i-1}) = 0$ aufgewertet und hohe Wahrscheinlichkeitswerte abgewertet werden. Insgesamt sollen also nicht nur 0 Wahrscheinlichkeiten verhindert werden, sondern das ganze Modell soll exakter und fehlerfreier werden. Bekannte Glättungsverfahren sind das Verfahren von Jelinek & Mercer [11], das Verfahren von Katz [12], das Verfahren von Witten & Bell [1] und das Verfahren von Kneser & Ney [13]. Ein Vergleich zwischen diesen Verfahren wird in [2] vorgestellt.
Bei allen diesen Glättungsverfahren kommt die Technik zum Einsatz, n-Gramm

Modelle geringerer Ordnung mit n-Gramm Modellen höherer Ordnung zu kombinieren. Diese Kombination wird typischerweise auf 2 Arten vollzogen: die Interpolationsmethode und die Backoff-Methode. Bei der ersten Methode ergibt sich die Wahrscheinlichkeit für eine Wortfolge $w_{i-2}w_{i-1}w_i$ aus einer Interpolation der relativen Häufigkeiten der Trigramme, Bigramme und Unigramme:

$$P(w_i|w_{i-2}w_{i-1}) = q_3P(w_i|w_{i-2}w_{i-1}) + q_2P(w_i|w_{i-1}) + q_1P(w_i),$$

wobei die Gewichte q_i dafür sorgen, daß die Stochastizitätsbedingungen erfüllt sind. Alle n-Gramme niedriger Ordnung fließen damit in die Berechnung ein. Dagegen wird bei der Backoff-Methode nur auf n-Gramme niedriger Ordnung zurückgegriffen, wenn die Häufigkeit des Trigramms zu gering ist. Ansonsten wird angenommen, daß die Trigrammwahrscheinlichkeit $P(w_i|w_{i-2}w_{i-1}) = C(w_{i-2}$ $w_{i-1}w_i)/C(w_{i-2}w_{i-1})$ eine bessere Schätzung ist als die Interpolationsformel.

7.2.3 Perplexität

Mit der Einführung der Perplexität ist es möglich, die Qualität von verschiedenen Sprachmodellen zu messen. Dieses Maß wurde in dem Bereich der Informationstheorie entwickelt und kann als der durchschnittliche Verzweigungsgrad eines Textes angesehen werden, wenn er von dem Sprachmodell erzeugt wird. Um die Perplexität eines Sprachmodells auf einer Menge von Testsätzen $T = (t_1, \ldots, t_n)$ zu bestimmen, wird zunächst die Entropie des Modell ermittelt:

$$H_p(T) = -\frac{1}{W_T}\log(\prod_{k=1}^{n} p(t_k)).$$

D.h., daß für alle Testsätze die Wahrscheinlichkeit berechnet wird, daß sie von dem Sprachmodell erzeugt werden, und anschließend wird dieser logarithmierte Wert durch die Anzahl der Wörter in der Testmenge W_T dividiert. Die Perplexität ergibt sich aus der Entropie nach folgender Vorschrift: $PP(T) = 2^{H_p(T)}$.
Falls z.B. ein Sprachmodell vorliegt, bei dem keine Informationen über bevorzugte Wortfolgen exisitieren und somit eine uniforme Wahrscheinlichkeitsverteilung $P(w_i|w_{i-2}w_{i-1}) = 1/L$ angenommen wird, ergibt sich für die Perplexität $PP(T) = L$. Dieser Wert wird umso kleiner, je besser das Sprachmodell geschätzt werden kann. Somit wird also nach einem Language Model mit der kleinsten Perplexität gesucht.

7.3 Experimente

Um ein Sprachmodell für den Formelerkenner zu erstellen, wurde das Glättungsverfahren von Kneser & Ney [13] angewendet. Dieses Back-Off Verfahren zeichnet sich dadurch aus, daß zur Berechnung der Wahrscheinlichkeit eines n-Gramms

niedriger Ordnung auch die n-Gramm Wahrscheinlichkeiten höherer Ordnung verwendet werden.
Für jede Hypothese $H = w_1, \ldots, w_n$ aus der N-Besten Liste des Erkenners wird eine Bewertung durch das Language Model berechnet $P_{LM}(H) = \prod_{k=1}^{n} p(w_k|w_{k-2}\ w_{k-1})$. Da die Hypothesen aus der N-Besten Liste sich in ihrer Länge deutlich unterscheiden können, sind diese Wahrscheinlichkeiten noch nicht miteinander vergleichbar. Um dies zu erreichen, werden die Wahrscheinlichkeit $P_{LM}(H)$ durch die Länge der Hypothese H dividiert, so daß man für jede Hypothese einen durchschnittlichen n-Grammwert erhält. Zusammen mit der Bewertung durch den Erkenner wird mit dieser Bewertung durch das Sprachmodell eine Interpolation durchgeführt, um die endgültige Bewertung jeder Hypothese zu bekommen.

$$P_{Gesamt}(H) = (1 - \lambda) * P_{Erkenner}(H) + \lambda * P_{LM}(H) \tag{7.2}$$

Durch diese Berechnung sollen Hypothesen aus der N-Besten Liste, die eine korrekte mathematische Struktur besitzen, aufgewertet werden, während Hypothesen mit einer falschen Struktur abgewertet werden.

7.3.1 Datensammlung

Die mathematischen Formeln für das Sprachmodell wurden mit dem Textformatierungssystem LaTeX [14] dargestellt, so daß zu den 72 Zeichen des Vokabulars $(A - z, a - z, 0 - 9, +, -, *, =, (,), \sqrt{}, \sum, \int, \infty)$ noch 4 Syntaxzeichen (_,^ für Exponenten- bzw. Indexbeziehungen und {, } für Syntaxklammern) dazukommen.
Um ein aussagekräftiges Sprachmodell zu erstellen, war es notwendig, eine Datensammlung für LaTeX-Dateien durchzuführen, aus denen die Formeln extrahiert wurden. Als Problem stellte sich heraus, daß in der Mehrzahl der Formeln Zeichen enthalten waren, die von dem Erkenner nicht unterstützt werden, wie z.B. griechische Buchstaben, Mengenoperatoren und Vektorpfeile. Um nicht alle diese Formeln unbenutzt zu lassen, wurden im Sinne der später durchgeführten Äquivalenzklassenbildung solche Zeichen auf bekannte Zeichen abgebildet, z.B. griechische Zeichen auf Kleinbuchstaben.
Ein weiteres Problem stellt die Qualität der gesammelten Formeln dar. Zum einen ist die relative Häufigkeit von Formel der Form a_i äußerst hoch und zum anderen passen die Daten für das Language Model nur bedingt mit den geschriebenen Formeln zusammen. So sind z.B. in den Daten für das Language Model kaum Formeln enthalten, die Zahlenfolgen, Klammerausdrücke oder einfache Ausdrücke der Form $3 + a - 2b * C$ enthalten. Um zu verhindern, daß das Sprachmodel solche Formeln gering bewertet, wurden Formeln dieser Form zu dem Sprachmodell hinzugefügt.
Insgesamt standen für den Aufbau des Sprachmodells rund 3.000 Formeln mit 50.000 Zeichen zur Verfügung. Verglichen mit den Datenbasen aus der Spracherkennung, die über 1 Million Wörter enthalten, ist diese Menge äußerst gering.

Damit aber trotzdem ein sinnvolles Sprachmodell erzeugt werden kann, werden die Zeichen in Äquivalenzklassen eingeteilt, so daß der Umfang des Sprachmodells klein bleibt.

7.3.2 Äquivalenzklassen

Mit Hilfe eines n-Gramm Modells läßt sich überprüfen, ob eine Formel ungefähr eine korrekte mathematische Struktur besitzt. Dagegen lassen sich Zweideutigkeiten z.B. zwischen 2 Buchstaben nur schwer auflösen. So kann mit den vorhandenen Datenmaterial z.B. nicht entschieden werden, ob die Hypothese '$3 + A$' oder '$3 + H$' besser ist. Das Gleiche gilt für die Hypothesen '$3 * 2$' und '$3 + 2$', '$1/2$ und '$1/7$'.
Da solche Zweideutigkeiten nicht aufgelöst werden können und zusätzlich das Sprachmodell klein gehalten werden soll, ist es notwendig, Äquivalenzklassen einzuführen. Insgesamt wurden 4 Experimente mit verschiedenen Äquivalenzklassen durchgeführt, wobei die verschiedenen Klassen per Hand ausgewählt wurden. In der Tabelle 7.1 sind die verschiedenen Klasseneinteilungen abgebildet. Im ersten

Klassen	# Trigramme	Perplexität
6 Klassen: [A-Za-z0-9], [$\sum \int$], [_^], [$+ - * =$], [{}()], [$\sqrt{}$;−]	133	3.22
7 Klassen: [A-Za-z0-9], [$\sum \int$], [_^], [$+ - * =$], [()], [{}], [$\sqrt{}$;−]	184	3.317
8 Klassen: [A-Za-z], [0-9], [$\sum \int$], [_^], [$+ - * =$], [()], [{}], [$\sqrt{}$;−]	286	3.753
9 Klassen: [A-Za-z], [0-9], [$\sum$], [$\int$], [_^], [$+ - * =$], [()], [{}], [$\sqrt{}$;−]	314	3.767

Tabelle 7.1: Verschiedene Äquivalenzklassen für das Sprachmodell

Experiment wurde mit 6 verschiedenen Klassen gearbeitet: die Klasse der Buchstaben und Zahlen, die Klassen für Summen- und Integralzeichen, die Klasse für Exponenten- und Indexzeichen, die Klasse der Operatoren, die Klasse der mathematischen Klammern und Syntaxklammern und die Klasse mit Wurzel- und Bruchzeichen. Diese Klassen wurden in weiteren Experimenten aufgespaltet. Je mehr Klassen verwendet werden, umso höher ist die Perplexität des zugehörigen Sprachmodells und umso mehr Trigramme sind in ihm enthalten.

7.3.3 Ergebnisse

In der Tabelle 7.2 sind die Ergebnisse beim Einsatz der verschiedenen Sprachmodelle abgebildet. Es wird deutlich, daß durch die Verwendung eines Sprachmodells in allen Fällen die Fehlerrate verringert werden kann. Insgesamt unterscheiden sich die Fehlerraten für die verschiedenen Sprachmodelle nur gering. Aber die kleinste Fehlerrate wird mit dem Sprachmodell erreicht, das 9 Äquivalenzklassen umfaßt. Hier läßt sich die Fehlerrate auf 22,64% verringern, was einer Verbesserung von rund 4.0% entspricht.

Wie auf Seite 57 untersucht worden ist, könnte die Fehlerrate sogar auf 18.10%

Sprachmodell	Fehlerrate (%)
ohne	26,6
6 Klassen	23,22
7 Klassen	23,13
8 Klassen	22,89
9 Klassen	22,64

Tabelle 7.2: Fehlerraten beim Einsatz verschiedener Sprachmodelle

gesenkt werden, falls die Auswahl aus der N-Besten Liste (N=20) nicht mit dem Sprachmodell geschieht, sondern immer die optimale Hypothese gewählt wird. Aber wenn bei der Gestaltung der N-Besten Liste immer nur ein Ergebnis aus einer Äquivalenzklasse vom Einzelzeichenerkenner zugelassen wird, dann kann auf der N-Besten Liste (N=20) nur eine Fehlerrate von 20,41% erreicht werden. Dieser Wert deckt sich fast mit dem erzielten Ergebnis beim Einsatz des Sprachmodells.

In der Abbildung 7.1 sind die Fehlerraten für unterschiedliche Gewichtungen der Bewertung des Sprachmodells (9 Klassen) gegenüber der Bewertung des Erkenners abgebildet. Für $\lambda = 0$ für den Parameter aus Gleichung 7.2 erfolgt die Entscheidung ohne Einfluß des Sprachmodells. In diesem Fall wird eine Fehlerrate von 26,6% erzielt. Je höher der Parameter λ gestellt wird, desto geringer wird zunächst die Fehlerrate, dessen minimaler Wert von 22,64% bei $\lambda = 0,6$ erreicht wird. Danach steigt die Fehlerrate wieder an. Falls die Auswahl aus der N-Besten Liste nur dem Sprachmodell überlassen wird ($\lambda = 1$), so wird die Hypothese gewählt, deren Bewertung durch das Sprachmodell maximal ist. Daß dies nicht immer die Hypothese mit der kleinsten Editierdistanz ist, zeigt die höhere Fehlerrate von 24,68%.

Zusammenfassend läßt sich somit sagen, daß mit einem n-Gramm Sprachmodell für die Formelerkennung eine deutliche Verbesserung der Systemleistung erzielt werden kann und daß die Mehrzahl der Fehler korrigiert werden können. Insgesamt verringert sich durch den Einsatz des Sprachmodells die Fehlerrate um rund

4% auf 22.64%.

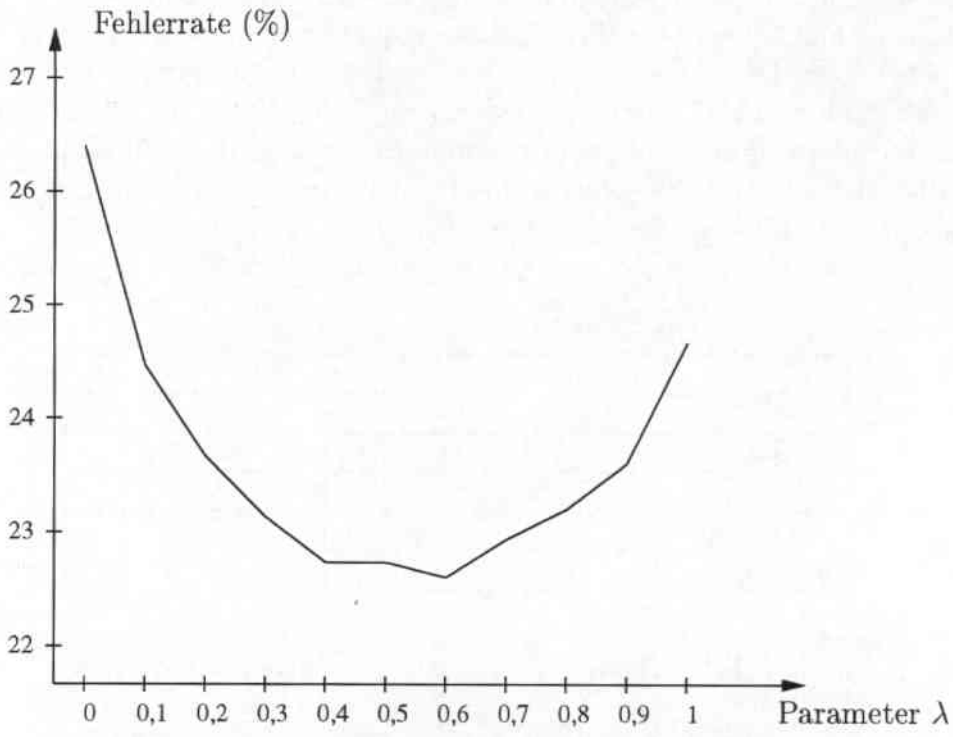

Abbildung 7.1: Fehlerraten bei unterschiedlicher Gewichtung des Sprachmodells

Kapitel 8

Datenbasis

Für den Aufbau eines Formelerkenners war es notwendig, eine eigene Datensammlung durchzuführen. Denn die bisher an der Universität Karlsruhe gesammelten Datenbasen enthalten nur Ziffern, die Zeichen aus dem Alphabet oder ganze Wörter aber keine mathematische Sonderzeichen oder vollständige Formeln. Aus diesem Grund wurde ein eigenes Datensammlungsprojekt initiiert, für welches die Benutzer sowohl einzelne Zeichen als auch vorgegebene Formeln aufschreiben sollten.
Die Eingabe erfolgte auf einem LCD-Bildschirm mit einem speziellen Stift, so daß die Position des Stiftes auf der Bildschirmoberfläche mit einer bestimmten Abtastfrequenz gespeichert werden kann. Damit wird jede Eingabe als eine Folge von Bildschirmkoordinaten repräsentiert. Dem Benutzer wurden keine Vorgaben für die Eingaben gemacht, sondern er sollte die Formeln in für ihn natürlicher Weise aufschreiben. Dadurch enthält die Datenbasis nicht nur unterschiedliche Schreibweisen der einzelnen Zeichen sondern auch unterschiedliche Schreibreihenfolgen der Zeichen innerhalb von Formeln. So konnte z.B. beobachtet werden, daß die Grenzen bei dem Summenzeichen sowohl direkt nach dem Summenzeichen als auch am Ende der Formel geschrieben wurden. Das Gleiche gilt für Klammerzeichen innerhalb von Termen.
Für das Training eines Einzelzeichenerkenners war die Eingabe von einzelnen mathematischen Zeichen notwendig. Es wurden sowohl die Zahlen, die Groß- und Kleinbuchstaben und das griechische Alphabet als auch eine Vielzahl von mathematischen Operatoren wie z.B. (=,+,-,*) gesammelt. Insgesamt mußte jeder Benutzer 163 Einzelzeichen eingeben, wovon 60 Zeichen zur Gruppe der mathematischen Operatoren gehören.
Da bei der Formelerkennung auch die mathematische Struktur der Eingabe erkannt werden muß, ist es erforderlich, daß in der Datenbasis auch Beispiele für die Anordnungen zwischen mehreren Zeichen enthalten sind. Deshalb mußten auch vollständige Formeln gesammelt werden. Auf diesen Daten kann dann z.B. die Anordnung der Zeichen bei Exponenten oder Indices gelernt werden. Zusätzlich wurden Formeln eingegeben, die mathematische Sonderzeichen wie z.B. Brüche,

Wurzel- und Integralzeichen enthalten. Auch bei diesen Sonderzeichen muß das System die spezielle Anordnung der Zeichen lernen. Außerdem kann mit den Formeln trainiert werden, wie eine komplexe Eingabe in die einzelnen Zeichen zu segmentieren ist. Insgesamt muß jeder Benutzer über 90 komplexe Formeln eingeben, die zusammen aus über 900 Einzelzeichen bestehen. Dabei enthält die einfachste Formel nur 2 Zeichen, z.B. bei einem Exponenten, und die längste Formel 34 Zeichen. Die Anzahl der Vorkommnisse der einzelnen mathematischen Sonderzeichen in diesen 90 Formeln kann aus Tabelle 8.1 entnommen werden. Zusätzlich

Sonderzeichen	Anzahl
Exponent/Index	69
Summen- und Produktzeichen	17
Wurzelzeichen	16
Integralzeichen	11
Brüche	28
Grenzwerte	2

Tabelle 8.1: Anzahl der Vorkommnisse der einzelnen mathematischen Sonderzeichen in der Datenbasis

mußte jeder Benutzer noch Formeln für Matrizen und Korrekturgesten eingeben. Diese Daten wurden aber für die Entwicklung des Formelerkenners nicht berücksichtigt. Insgesamt dauerte die Eingabe aller Formeln und Zeichen pro Benutzer ungefähr 30-45 Minuten.
28 Personnen konnten gewonnen werden, eine Datenspende abzugeben, von denen 3 Linkshänder waren. Damit standen zum Aufbau eines Formelerkenners über 4500 Einzelzeichen und über 2500 Formeln zur Verfügung. Alle Daten wurden gesichtet, um zu entscheiden, ob die Schreibweise ordentlich genug war und keine Aufnahmefehler entstanden sind. Zusätzlich wurde für eine kleine Anzahl von Formeln per Hand eine Segmentierung erstellt.
Bei dem Betrachten der Daten mußte festgestellt werden, daß die Zeichen aus dem griechischen Alphabet zum größten Teil nicht brauchbar sind. Dies liegt vor allen Dingen daran, daß die Schreibweise einiger seltener Zeichen vielen Benutzern nicht bekannt waren. Sehr unterschiedlich zeigte sich auch die Schreibweise des Malzeichens (*). Da den Benutzern keine Beschränkung bei der Eingabe gemacht wurde, variiert das Schriftbild des Malzeichens von einem einzigen Punkt bis zu einem Stern aus 4 Linien. Manche Benutzer lassen den Malpunkt in komplexen Formeln auch weg. Diese sehr verschiedenen Schreibweisen führen dazu, daß das Modell des Malzeichens kaum gut trainiert werden kann. Ein weiteres Problem konnte bei manchen Schreibweisen des Wurzelzeichens beobachtet werden. Eigentlich wird ein neues Zeichen erst dann begonnen zu schreiben, wenn das vorherige Zeichen zu Ende geschrieben wurde. Dies gilt aber nicht immer bei

dem Wurzelzeichen, da es vorkommen kann, daß das Zeichen angefangen wird, dann der Term unter dem Wurzelzeichen geschrieben wird und erst zum Schluß das Wurzelzeichen beendet wird. Eine solche Schreibreihenfolge erschwert den Erkennungsprozeß enorm und wird deshalb von dem entwickelten Formelerkenner nicht unterstützt. Auffällig beim Betrachten der Daten war weiterhin, daß durch die Aufnahmehardware viele Zeichen mit einem Aufsetzungsfehler behaftet waren. Dies wird im Schriftbild dadurch deutlich, daß am Anfang des Schriftzuges ein kleiner Haken vorhanden ist. Um diesen Fehler zu entfernen, wird das im Abschnitt 3.1.4 beschriebene Verfahren angewendet.

Kapitel 9

Ausblick

Mit den in den vorherigen Kapiteln beschriebenen Verfahren wurde ein System für die Formelerkennung erstellt, das eine Fehlerrate von 22.64% erzielt. Im Rahmen dieser Arbeit wurden alle Komponenten dieses Systems bestmöglichst und mit Erfolg optimiert, dennoch gibt es eine Vielzahl von Möglichkeiten den Formelerkenner noch weiter zu verbessern.

- Gerade bei dem Einzelzeichenerkenner besteht die Notwendigkeit der Verbesserung. Zwar werden mit den Daten von den 28 Schreibern eine Erkennungsrate von 87.27% erreicht, aber diese sollte sich noch erhöhen lassen, falls mehr Trainingsmaterial vorhanden wäre. Der Aufbau einer größeren Datenbasis würde mit Sicherheit die Performance des gesamten Systems verbessern, da der Einzelzeichenerkenner auch mitverantwortlich für die Segmentierung ist.

- Um mit dem System in der Praxis arbeiten zu können, reichen die im Moment im System zur Verfügung stehenden 72 Zeichen nicht aus. Dafür wäre eine Erweiterung des Zeichensatzes um z.B. trigonometrische Funktionen, weitere mathematische Operatoren und das griechische Alphabet notwendig.

- Die Schwierigkeiten des Segmentierers liegen vor allen Dingen beim Zusammenfassen von Strokes eines Zeichens, die weit voneinander entfernt verlaufen, wie z.B. beim 'i-Zeichen', dem Gleichheitszeichen und besonderen Schreibweisen des Summenzeichens. Hier müßte das neuronale Netz des Segmentierers optimiert werden. Denkbar wäre die Aufnahme von mehr Beispielen dieser problematischen Zeichen in die Trainingsmenge oder die Entwicklung von Merkmalen, die nur bei diesen Zeichen ansprechen, oder die Entwicklung einer Prerecognition Stage [29].

- Wie gezeigt werden konnte, funktioniert die Strukturanalyse äußerst zufriedenstellend. Trotzdem ist vorstellbar, falls keine eindeutige Anordnung der

Zeichen vorliegt, auch während der Strukturanalyse alternative Hypothesen in einer N-Besten Liste aufzunehmen. Zwischen diesen Alternativen würde ein Sprachmodell die beste Lösung auswählen.

- Durch das Sprachmodell konnte eine Verbesserung der Erkennungsleistung von 26,6% auf 22,6% erreicht werden. Weitere Experimente in diesem Bereich sind aber davon abhängig, daß eine größere Datenbasis erstellt wird. Erst damit können weitere Versuche mit neuen Verfahren oder mit unterschiedlichen Äquivalenzklassen gestartet werden.

- Bis jetzt berücksichtigt das Sprachmodell nur Trigramme. Eine Vergrößerung der Ordnung des n-Gramm Modells würde zwar zu einem deutlich größeren Sprachmodell führen, aber viele typische Wortfolgen ließen sich erst dann erfassen, wie z.B. bei den LaTeX-Ausdrücken für $f(x)$,$\int_0^1$ oder a_i: f (x), $\int$ _ { 0 } ^ { 1 }, a _ { i }.

- Der Nachteil von n-Gramm Sprachmodellen liegt darin, daß sie keine Zusammenhänge zwischen weit entfernten Wörtern modellieren können. Doch gerade bei mathematischen Formeln besteht dieser Zusammenhang oft: nach einer offenen Klammer folgt meistens irgendwann die geschlossene Klammer, der Index bei einem Summenausdruck wird meistens auch im Summenterm verwendet und nach einem Integralzeichen folgt meistens ein Term der Form dx. Um diese Zusammenhänge auch mit einem n-Gramm Sprachmodell erfassen zu können, besteht die Möglichkeit, das Sprachmodell um Trigger Pairs zu erweitern [6],[23].

- Eine weitere Verbesserung der Systemleistung ist zu erwarten, wenn das Sprachmodell nicht auf die N-Besten Liste angewendet wird, sondern wenn es in die Suche integriert wird. Dadurch würde man die Suche nach dem nächsten Zeichen abhängig von den vergangenen Zeichen machen und könnte somit die Hypothesenbildung direkt mit dem Sprachmodell beeinflußen.

- Für einen Einsatz in der Praxis wäre eine Erweiterung des Systems für die Eingabe von Matrizen und Vektoren wünschenswert.

Literaturverzeichnis

[1] T. Bell, J. Cleary und I. Witten: Text Compression. Prentice Hall, Englewood Cliffs, N.J.

[2] S.F. Chen und J. Goodman: An empirical study of smoothing techniques for language modeling. Computer, Speech and Language, Vol.13, No.4, Oktober 1999.

[3] K.F. Chan und D.Y. Yan: An Efficient Syntactic Approach to Structual Analysis of On-line Handwritten Mathematical Expressions, in Pattern Recognition, March 1999.

[4] J.G.A. Dolfing und R. Haeb-Umbach: Signal Representations for Hidden Markov Model Based On-line Handwriting Recognition. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, München, 1997.

[5] M. Gronroos: Public Domain Neuronal Network Simulator
ftp://ftp.funet.fi/ftp/pub/sci/neural/magi/backprop

[6] Z. GuoDong und L. KimTeng: Interpolation of n-gram and mutual-information based trigger pair language models for Mandarin speech recognition. Computer, Speech and Language, Vol.13, No.2 April 1999.

[7] P. Haffner und A. Waibel: Multi-State Time Delay Neural Networks for Continuous Speech Recognition. In Advance in Neural Networks Information Processing Systems 4, M. Kaufmann, 1992.

[8] H. Hild und A. Waibel: Connected Letter Recognition with a Multi-State Time Delay Neural Network. In Advance in Neural Networks Information Processing Systems 5, M. Kaufmann, 1993.

[9] I. Guyon, P. Albrecht, Y. Le Cun, J. Denker und W. Hubbard: Design of A Neuronal Network Character Recognition for A Touch Terminal. Pattern Recognition, 24(2):105-119, 1991.

[10] F. Jelinek: Self-Organized Language Modeling For Speech Recognition. In A. Waibel, K. Lee (Hrsg.) Readings in Speech Recognition, Morgan Kaufmann, San Mateo, 1990.

[11] F.Jelinek und R. Mercer:Interpolated estimation of Markov source parameters from sparse data. Proceedings of the Workshop on Pattern Recognition in Practice, Amsterdam, Mai 1990.

[12] S. Katz: Estimation of probalities from sparse data for the language model component of a speech recognizer. IEEE Transactions on Acoustics, Speech and Signal Processing, März 1987.

[13] R. Kneser und H. Ney: Improved backing-off m-gram language modeling. Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 1, 1995.

[14] H.Kopka: LaTeX eine Einführung, Addison-Wesley, 1993.

[15] A.Kosmala und G.Rigoll: Recognition of Online Handwritten Formulas.

[16] S.Manke, M.Finke und A.Waibel: Combining Bitmaps with Dynamic Writing Information for On-Line Handwriting Recognition. In Proceedings of the International Conference on Pattern Recognition, Seite 596-598, Jerusalem, Oktober 1994.

[17] S.Manke, M.Finke und A.Waibel: NPen++: A Writer Independent Large Vocabulary ON-Line Cursive Handwriting Recognition System, In Proceedings of the International Conference on Pattern Recognition, Seite 596-598, Jerusalem, Oktober 1994.

[18] S. Manke: On-line Erkennung kursiver Handschrift bei großen Vokabularen. Dissertation, Shaker Verlag, 1998.

[19] N.E.Matsakis: Recognition of Handwritten Mathematical Expressions

[20] E.G. Miller und P.A. Viola: Ambiguity and Constraint in Mathematical Expression Recognition. American Association of Artifical Intelligence, 1998.

[21] H.Niemann: Pattern Analysis and Understanding, Bd. 4 von Series in Information Sciences, Springer, Berlin Heidelberg, 1990.

[22] L.R. Rabiner: A Tutorial on Hidden Markov Models and Selected Applications in SPeech Recognition. Proceedings of the IEEE, Feb. 1989.

[23] R. Rosenfeld: Adaptive statistcal language modeling:a maximum entropy approach. PhD Thesis, Carneige Mellon University, Boston, 1994.

[24] A. Schulz-Heyn: Maschinelle Erkennung von handgeschriebenen, mathematischen Ausdrücken. Diplomarbeit, Karlsruhe, 1997.

[25] R.Seiler, M. Schenkel und F.Eggiman: Cursive Handwriting Recogniton: Offline versus On-line Recognition. In Proceedings of the International Workshop on Frontiers in Handwriting Recognition, Coulchester, 1996.

[26] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano und k. Lang: Phoneme Recognition Using Time-Delay Neuronal Networks. IEEE, Transactions on Acoustics, Speech and Signal Processing, März 1989.

[27] H-J. Winkler: Symbol Recognition in Handwritten Mathematical Formulas. Int. Workshop on Modern Modes of Man-Machine-Communication, Maribor S. 7/1 - 7/10 , Juni 1994.

[28] H-J. Winkler: Symbol Segmentation and Recognition for Understanding Handwritten Mathematical Expressions. 5th Int. Workshop on Frontiers in Handwritten Recognition, Essex, 1996.

[29] H-J. Winkler: A Soft-Decision Approach For Structural Analysis Of Handwritten Mathematical Expressions. Conference on Acoustics, Speech, and Signal Processing, Detroit, 1995.