Die Evaluierung unserer Modelle soweit sollte eine Word Accuracy im Bereich von 40% ergeben (das ist übrigens schon ziemlich gut, wenn man bedenkt, wie schwierig die Aufgabe ist!). Wir nennen diesen Wert unsere Basiserkennung und wollen die zukünftigen Verbesserungen unseres Erkenners in Form von relativen Verbesserungen der Word Accuracy zu diesem Wert messen.

Wir beginnen nun mit der schrittweisen Verfeinerung unseres Basiserkenners. Damit wir mit den vielen Versionen, die wir erzeugen werden, nicht durcheinanderkommen, werden wir die jeweils zu einer Version gehörigen Dateien in spezielle Sub-Directories speichern. (Wir haben damit bereits begonnen: hmm1, hmm2, ...)

2.7 Baum-Welch-Training

In der klassischen Vorgehensweise von HTK folgt auf das Bootstrapping und das erste segmentale Viterbi-Training (HInit) ein segmentales Baum-Welch-Training. Dazu dient der Befehl HRest. Das Baum-Welch Training auf fixe Segmente ist besonders günstig bei Ganzworterkennern. In unserem Falle handelt es sich aber um einem phonem-basierten Erkenner, wo ein sog. 'embedded training' wirkungsvoller ist. Deshalb unterlassen wir aus Zeitgründen das Nachtraining der Segmente mit HRest.

Trotzdem soll hier kurz auf den Unterschied zum vorangegangenen Viterbi-Training eingegangen werden:

Beim Viterbi-Training werden die Parameter der HMM aus den Beobachtungen neu geschätzt, die längst des wahrscheinlichsten (Viterbi-)Pfades durch das Modell gesammelt wurden. Dies ist genaugenommen nicht ganz korrekt, da ein HMM zu einer gegebenen Vektorfolge im Prinzip unendlich viele mögliche Zuordnungen der Vektoren auf die States des Modells erlaubt. Beim Viterbi-Training wird aber nur die Zuordnung (Pfad) mit der höchsten Gesamtwahrscheinlichkeit für das Training berücksichtigt. Alle anderen Pfade werden nicht berücksichtigt.

Diese Näherung wird beim Baum-Welch-Training aufgehoben. Baum-Welch berücksichtigt mit Hilfe rekursiver Schätzformeln tatsächlich alle möglichen Pfade durch das HMM. Dadurch entstehen robuster geschätzte Parameter sowohl in den statistischen Verteilungen (Mixturen) als auch in den Übergangswahrscheinlichkeiten. Natürlich ist Baum-Welch etwas aufwendiger als ein simples Viterbitraining, aber der Aufwand steigt nicht exponentiell.

Für unseren speziellen Fall der Verbmobil-Korpora hat sich allerdings keine Verbesserung gegenüber dem Viterbitraining ergeben, weswegen wir diesen Schritt aus Zeitgründen hier nicht durchführen.

2.8 Editieren von HMM: Optionales Pausenmodell zwischen Wörtern

Bisher werden Pausen nur am Beginn und Ende jeder Äußerung ('!ENTER' und '!EXIT' modelliert durch das HMM /<p:>/) sowie an Stellen modelliert, wo der Sprecher eine deutliche Pause macht ('!SIL' ebenfalls modelliert durch /<p:>/). Nun kommen aber sehr kurze Pausen zwischen Wörtern noch sehr viel häufiger vor. Ein einfacher Weg, dies besser zu modellieren, ist das Anfügen eines sog. T-Modells an jedes Wort in unseren Lexica.

Ein T-Modell hat nichts mit Fords berühmten ersten Serienwagen zu tun, sondern bezeichnet ein HMM, das optional komplett übersprungen werden kann. In HTK ist dies sehr leicht möglich, indem man eine Transition vom virtuellen ersten State in den virtuellen letzten State des Modells erlaubt. Z.B. hat folgende Übergangswahrscheinlichkeiten-Matrix einen solchen Sprung:

```
<TRANSP> 4
0.000000e+00 8.000000e-01 0.000000e+00 2.000000e-01
0.000000e+00 5.796092e-01 4.203908e-01 0.000000e+00
0.000000e+00 0.000000e+00 7.602808e-01 2.397192e-01
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
```

Zunächst ergänzen wir unsere Aussprachen in beiden Lexica um ein angehängtes Pausenmodell: Kopieren Sie zunächst beide Lexica nach TESTDEV-sp.lex bzw. TRAIN-sp.lex (Sicherungskopien). Man könnte jetzt mit einem Editor an jede Aussprache in TESTDEV.lex und TRAIN.lex noch das Phonem /<sp>/ anhängen. Einfacher ist es, folgenden SED-Befehl zu verwenden:

```
% cd ~/HTK-KURS/GRUPPE#
% cp TRAIN.lex TRAIN-sp.lex
% cat TRAIN.lex | sed 's/$/ <sp>/' > TRAIN.lex.tmp
% mv TRAIN.lex.tmp TRAIN.lex
```

Ein typischer Eintrag schaut dann so aus:

```
'"ubel'Q y: b @ l <sp>
```

Editieren Sie nochmal beide Lexica und entfernen die /<sp>/ Modelle nach den Wörtern 'ohne Semantik' (!ENTER, !EXIT,...).

Nun erzeugen wir das /<sp>/ T-Modell selber:

```
% cd ~/HTK-KURS/GRUPPE#
% mkdir hmm2
% cp hmm1/* hmm2/
% cd hmm2
% cp '<p:>' '<sp>'
```

Editieren Sie nun die Datei '<sp>' und machen folgende Änderungen:

- `'~h "<p:>"'  =>  '~h "<sp>"'` (Makroname)
- Ändern Sie die Anzahl der States auf 3 (inklusive virtueller States!)
- Löschen Sie die States 2, 4 und 5 (nur 3 bleibt!)
- Löschen Sie in der Übergangsmatrix die entsprechenden Spalten und Zeilen (Summe der Zeilen muß 1 sein!).

Kopieren Sie Ihre momentane Phonem-Liste PHONEME nach PHONEME-sp und editieren Sie dann die Datei PHONEME und fügen das neue Modell hinzu. Fassen Sie alle (jetzt 48) Modelle erneut zu einem MMF hmm2/VM.mmf zusammen (siehe letzte Stunde).

Zusätzlich soll jetzt noch der einzige emmitierende State unseres neuen Modells /<sp>/ an das bestehende /<p:>/-Modell gebunden werden. Man nennt dies auch 'tying'. Praktisch bedeutet dies, daß der State nur an einer einzigen Stelle wirklich gespeichert ist, und mehrere HMM sich diesen State 'teilen' (to share). Der Vorteil von geteilten Parametern ist natürlich die Tatsache, daß dadurch mehr Trainingsmaterial pro State zur Verfügung steht, und somit die Parameter robuster geschätzt werden können. Technisch geschieht das 'tying' mit Hilfe von sog. Makro-Einträgen

(ähnlich dem Makro-Eintrag '∼h'). An der entsprechenden Stelle im HMM Makrofile, wo ein Parameter geteilt (oder gebunden) werden soll, steht dann nur noch ein Makro-Befehl, der auf die Stelle verweist, wo die Parameter tatsächlich gespeichert sind.

Der HMM-Editor HHEd kann solche gebundenen Strukturen automatisch erzeugen, und auch die richtige Übergangs-Matrix für ein T-Modell eintragen:

Editieren Sie ein HHEd-Skript 'sp.hed':

```
% cd ~/HTK-KURS/GRUPPE#
% cat > sp.hed
AT 1 3 0.3 {<sp>.transP}
AT 2 4 0.1 {<p:>.transP}
AT 4 2 0.1 {<p:>.transP}
TI p_st {<p:>.state[3],<sp>.state[2]}
^D
%
```

Der erste 'AT'-Befehl fügt eine weitere Transition von State 1 nach State 3 in unser /<sp>/-Modell ein; die beiden nächsten 'AT'-Befehle fügen noch ein paar Transitionen in unser normales /<p:>/-Modell ein, um es robuster zu machen. Der 'TI'-Befehl bindet den 2. State des /<sp>/-Modells an den 3. State des /<p:>/-Modells.

```
% HHEd -H hmm2/VM.mmf -w hmm2/VM+sp.mmf -T 1 sp.hed PHONEME
% less hmm2/VM+sp.mmf
...
% mv hmm2/VM+sp.mmf hmm2/VM.mmf
```

Jetzt ist es Zeit für einen weiteren Test mit unserem Development-Testset. Um uns diesmal die Zeit zu verkürzen, verwenden wir ein Skript TEST-PAR, das den Test automatisch in mehrere kleinere Tests zerlegt, diese an verschiedene, gerade unbenutzte Rechner schickt, die Ergebnisse wieder sammelt und ausgibt.

Vergleichen Sie das Ergebnis mit dem der Modelle in hmm1. Testen Sie die Signifikanz der Unterschiede mit Hilfe des Programms 'signif'. Ist die Einführung von Pausenmodellen zwischen den Wörtern bereits ein Gewinn?

2.9 'Embedded Training'

2.9.1 Vorbereitung

Bisher haben wir die einzelnen Phonem-Modelle mit gelabelten Trainingssegmenten geschätzt. Im Prinzip ist das die Technik, die in der traditionellen Worterkennung verwendet wird: Man hat zu jedem Wort eine Anzahl von Trainingsbeispielen und trainiert das Modell genau auf diese einzelnen Beispiele. Für die Erkennung fließend gesprochener Sprache hat dieses Verfahren einige Nachteile:

- Wörter/Phoneme werden in fließender Sprache anders realisiert als in Isolation
- Es gibt viel zu wenig sicher gelabeltes und segmentiertes Material
- Die Aufteilung in Wörter/Phoneme beim Erkennungsprozess wird zu wenig berücksichtigt

Daher wurden schon relativ früh Verfahren entwickelt, die mit kompletten Äußerungen fließender Sprache trainieren, ohne daß sie eine Segmentierung in Wörter/Phoneme benötigen. Solche Verfahren werden meist mit dem Begriff 'embedded training' (etwa: 'eingebettetes Trainieren') bezeichnet.

Ein 'embedded training' mit Phonemen macht im Prinzip folgendes:

1. Nimm eine komplette Äußerung (z.B. ein Turn), lies aus einem Transkript die zugehörigen Phoneme und hänge sie zu einer langen Kette zusammen.
2. Verkette die zugehörigen Phonem-HMM zu einem riesigen Modell und führe ein sog. 'forced Viterbi alignment' durch. Das ist - ganz analog zum Viterbi-Training in HInit - eine Zuordnung, welche Merkmalsvektoren auf welche States unseres Riesenmodells fallen.
3. Verwende die gefundene Segmentierung und schätze die Modelle mit Viterbi-Training nach.
4. Ersetze die bisher verwendeten Modelle und mach weiter bei Punkt 2, bis sich die globale mittlere Wahrscheinlichkeit der Modelle nicht weiter verbessert (mit anderen Worten: bis sich die Zuteilung von Merkmalsvektoren auf States nicht mehr ändert).
5. Speichere die zuletzt geschätzten Modelle ab.

In HTK wird genau eine Iteration (Schritt 2-4) eines solchen 'embedded training' vom Tool HERest durchgeführt. Logischerweise betrifft ein solches Training jetzt nicht mehr, wie bei HInit oder HRest, nur ein einzelnes HMM, sondern alle auf einmal. Außerdem muß gewährleistet sein, daß sich jede Trainingsäußerung *bündig* durch eine Kette von HMM darstellen läßt. Das bedeutet z.B., daß auch Geräusche und Pausen korrekt modelliert werden müssen (man kann keine 'Lücken' lassen).

HERest erledigt nicht automatisch den Punkt 1, die Erstellung einer Transkription pro Trainings-Äußerung. Wir haben jetzt zwei Möglichkeiten eine solche zu erstellen:

A) Wir nehmen unser Trainings-Lexikon TRAIN.lex und expandieren damit die Wort-Referenzen des Trainings-Sets (TRAIN.REF.mlf). Damit haben wir für jede Trainingsäußerung eine *kanonische* Aussprache erzeugt (zur Erinnerung: unsere Lexica enthalten die kanonische Aussprache isoliert gesprochener Wörter!). Das ist der klassische Weg, weil in den meisten Fällen zum Trainings- material keine andere Information zur Verfügung steht. Das Tool HLEd kann das mit dem Befehl 'EX' für uns erledigen:

```
% cd ~/HTK-KURS/GRUPPE#
% cat > expand.led
EX
^D
% HLEd -l '*' -d TRAIN.lex -i TRAIN.KAN.mlf expand.led TRAIN.REF.mlf
% less TRAIN.KAN.mlf
```

Das neue MLF TRAIN.KAN.mlf enthält nun für jedes Wort genau die Aussprache, die wir auch beim Testen mit dem Test-Lexikon TEST.lex erwarten würden (inklusive /<sp>/-Pausenmodelle!).

B) Anstatt der kanonischen Aussprache, die ja in Realität nur für isoliert gesprochene Wörter gilt, könnten wir auch gleich die Transkripte verwenden, die in TRAIN1.mlf gespeichert sind. Diese haben wir zum Bootstrapping verwendet und stammen aus der MAUS Segmentierung. Technisch ist das tatsächlich auch kein Problem. HERest liest diese Labels und ignoriert

einfach die Zeitinformation. Dann haben wir jedoch ein Mißverhältnis zwischen akustischen Modellen und dem Aussprache-Lexikons beim Testen. Warum? Weil wir die akustischen Modelle so trainieren, wie sie tatsächlich geäußert wurden und beim Testen ein Lexikon mit der kanonischen Aussprache verwenden.

Aus diesem Grunde wollen wir hier die Variante A verfolgen Wir können auf die Variante B später zurückkommen, wenn wir eine bessere phonetische Modellierung des Aussprache-Lexikons versuchen.

2.9.2 'Embedded Training' mit HERest

HERest hat - ähnlich wie HVite - eine Unzahl von Optionen und Parametern. Die wichtigsten sind in folgendem typischen Aufruf erklärt:

```
% cd ~/HTK-KURS/GRUPPE#
% mkdir hmm3
% HERest -t 250.0 150.0 1000.0 -v 0.0001 \
 -H hmm2/VM.mmf -I TRAIN.KAN.mlf -T 1 \
 -S ../TRAIN.123.slist -M hmm3 -A PHONEME
```

Optionen:

```
-t #1 #2 #3 : Pruning - Zur Beschleunigung des Trainings wird auch
 hier, wie bei HVite, ein Pruning durchgeführt, das nur
 diejenigen Pfade überleben läßt, die innerhalb eines
 Bereichs von '#1' unter dem besten liegen (log. WKs).
 Es kann daher vorkommen, daß der beste Pfad 'verloren'
 geht und das Training abbricht. Damit so etwas nicht
 vorkommt, wird der Pruningfaktor '#1' dann um den Wert
 '#2' erhöht und das Training erneut gestartet. Das kann
 so oft passieren, bis '#3' erreicht ist. Wenn dann immer
 noch kein Erfolg eintritt, bricht das Training dieser
 Äußerung entgültig ab und es wird eine Fehlermeldung
 generiert. Meistens handelt es sich dann um einen
 'mismatch' zwischen Signal und Referenz-Wortkette.
-v #        : minimale Varianz in den Kovarianz-Matrizen
-H nam      : Filename des HMM-Makro-Files (MMF) mit den
 Phonem-Modellen
-I nam      : Filename des MLF mit den Laut-Referenzketten des
 Trainingskorpus
-T #        : Trace-Level
-S nam      : Filenamen des 'scripts' mit den Trainingsfiles des
 Trainingskorpus
-M nam      : Directory, wohin das neue MMF geschrieben werden soll
-A          : Wiederhole Kommandozeile für Logging
```

Dieser Beispiel-Aufruf führt genau eine Iteration mit dem gesamten Trainingskorpus durch. (Zur Erinnerung: wir hatten das gesamte Trainingskorpus von ca. 30 h in drei Teile von je 10 h aufgeteilt.)

Testen Sie den Aufruf von HERest wie oben angegeben. Was schätzen Sie, wie lange eine Iteration dauern wird? (Messen sie die mittlere Dauer für eine Äußerung und multiplizieren mit der Anzahl der Äußerungen im Trainings-Korpus.)

Brechen Sie den Prozess wieder ab.

2.10 Betrachtungen zur Konvergenz

HERest liefert i.G. zu HInit und HRest keine Informationen zur Konvergenz der Modelle. Es existiert lediglich ein formaler Beweis, daß ein Training, wie es HERest durchführt, auf die gegebenen Trainingsdaten konvergiert. Mit anderen Worten: nach jeder Iteration kann die globale Wahrscheinlichkeit der Daten gegeben die Modelle nur höher oder schlimmstenfalls gleich bleiben.

Die Frage ist aber: Will man das überhaupt?

Stellen Sie sich in einem Gedankenexperiment vor, das Trainingsset bestehe nur aus 100 Äußerungen. Jetzt trainieren wir unsere Phonem-Modelle solange, bis sich die globale Wahrscheinlichkeit nicht mehr ändert. Die Modelle sind nun optimal an die Trainings- daten angepasst. Zu unserem Erstaunen ergibt aber eine Auswertung mit unserem 'testbed', auf 100 Sätze, die nicht im Training-Set vorkommen, nur mäßige Ergebnisse. Wir haben den typischen Fall der *Überadaptation*. D.h. die Modelle können zwar nun die 100 Sätze im Trainingsset optimal modellieren, aber alles andere ziemlich schlecht, weil sie nicht mehr *generalisieren*.

In der Theorie der statistischen Modelle kommt dieses Problem nicht vor, da alle Statistik immer davon ausgeht, daß das Trainingsmaterial unbegrenzt groß ist. In der Praxis ist das natürlich bei weitem nicht der Fall.

Man kann das Problem umgehen, indem man das *Abbruch-Kriterium* anders definiert und sagt: Trainiere solange, bis die Wort-Akkuratheit auf einem unabhängigen Development-Sets ein Maximum erreicht."Beachten Sie, daß die Wort-Akkuratheit bei weiterem Training wieder absinkt! (Warum?)

2.11 Übung HEREST

Schreiben Sie ein Skript HEREST, das in einer Schleife solange abwechselnd HERest und HVite aufruft, bis die 'Word Accuracy' sich weniger als 0.1% verbessert.

Hinweise:

- Jede neue Version der HMM soll in ein separates MMF geschrieben werden, damit die alte Version nicht überschrieben wird. Am besten in ein Sub-Directory hmm# (# : Iterations-Nummer) (Es kann ja sein, daß die vorherige Version die beste war!)
- Ermitteln Sie die 'Word Accuracy', indem sie den Output von Ihrem 'testbed' TEST entsprechend filtern.
- Nehmen Sie als Start-Modelle die ergänzten Modelle in hmm2/VM.mlf.
- Nehmen Sie als Training-Set TRAIN.123.slist und als Trainings- Transkripte TRAIN.KAN.mlf.
- Testen Sie die Funktionalität ihres Skripts zunächst, indem Sie die echten Trainingsläufe geeignet simulieren (z.B. indem Sie ein 'echo' davor setzen; dann wird der nachfolgende Befehl nicht ausgeführt, sondern nur ausgegeben). Erst wenn Sie ganz sicher sind, daß das Skript funktioniert, starten Sie es 'in reality'.
- Protokollieren Sie den Fortschritt, damit Sie leichter Fehler entdecken können.

Probleme, die auftreten können:

- Ein Trainingsfile kann nur ein Geräusch und keine Wörter enthalten.
  Überprüfen Sie Ihr Wortreferenz-MLF auf solche Files und löschen Sie diese in der File-Liste des Trainings-Korpus. Kopieren Sie dazu das Skript ../TRAIN.123.slist in Ihr Gruppen-Directory. (Das awk Skript 'detect-zero.awk' in HTK-KURS kann z.B. helfen, die 'leeren Files im MLF zu finden.)
- Ein Phonem wird zu wenig oft realisiert.
  Da wir bereits mit einer nur halb so großen Stichprobe erfolgreich das Bootstrapping durchgeführt haben, ist damit eher nicht zu rechnen.
- Rechnen in der C-Shell ist nur mit Integerzahlen möglich; Rechnen Sie also in Hundertstel Prozent (30.34% = 3034)

Testen Sie Ihr Skript als Hausaufgabe (bitte wie immer mit 'nice' und mit Protokoll im Hintergrund laufend!). Nehmen Sie als Start-Modelle hmm2/VM.mmf und brechen Sie nach einer Iteration ab. Die neuen Modelle sollten dann in hmm3/VM.mmf liegen.

Sollten Sie sich mit dieser Aufgabe überfordert fühlen (was keine Schande ist) können Sie sich ein Muster-Skript aus ../MUSTER/HEREST kopieren. Gehen Sie dann gemeinsam das Skript durch und versuchen Sie zu verstehen, was darin gemacht wird. ACHTUNG: Dieses Skript muß noch für Ihre Anwendung angepasst werden!

Vertiefung für technisch Interessierte

Der rechenintensivste Teil solcher Trainingsdurchläufe ist die 'Re-Estimation' von HERest. Daher kann HERest im sog. *Parallel-Modus* aufgerufen werden (Option '-p #'). Die Grundidee dabei ist, daß man das Trainings-Set in mehrere Teile aufteilt (z.B. in unserem Falle TRAIN1-3) und HERest für jeden Teil getrennt auf einem eigenen Rechner laufen läßt. Dazu ruft man HERest z.B. mit der Option '-p 1' auf und die Schätzwerte werden zusammen mit ihren Häufigkeitszählern in sog. *Akkumulatoren* gesammelt. Jeder der Parallel-Prozesse schreibt seine akkumulierten Daten in ein separates File. Wenn alle Prozesse fertig sind, wird HERest noch einmal - diesmal mit der Option '-p 0' - aufgerufen und die einzelnen Accumulatoren zu den entgültigen Schätzwerten zusammengefasst.

Beispiel:

Nehmen wir an, wir haben drei Rechner (host1 - 3) zur Verfügung. Die Berechnung einer Iteration im Parallel-Modus könnte dann wie folgt aussehen:

```
# Paralleler Aufruf für drei Teilkorpora, Accumulatoren werden
# nach hmm4 geschrieben
ssh host1 HERest -t 250.0 150.0 1000.0 -v 0.0001 \
 -H hmm3/VM.mmf -I TRAIN1.mlf -T 1 \
 -S ../TRAIN.1.slist -M hmm4 -A -p 1 PHONEME &
ssh host2 HERest -t 250.0 150.0 1000.0 -v 0.0001 \
 -H hmm3/VM.mmf -I TRAIN3.mlf -T 1 \
 -S ../TRAIN.2.slist -M hmm4 -A -p 2 PHONEME &
ssh host3 HERest -t 250.0 150.0 1000.0 -v 0.0001 \
 -H hmm3/VM.mmf -I TRAIN3.mlf -T 1 \
 -S ../TRAIN.3.slist -M hmm4 -A -p 3 PHONEME &
# Warte bis die Jobs alle fertig sind (z.B. mit Semaphoren)
...
# Liste der Accumulatoren-Files erstellen
```

```
ls hmm4/*.acc >! ACC
# Accumulatoren zusammenfassen und neue Modelle schätzen
HERest -v 0.0001 -H hmm3/VM.mmf -T 1 -M hmm4 \
 -S ACC -A -p 0 PHONEME
# Accumulatoren löschen
rm hmm4/*.acc
```

In diesem Beispiel sind die ganzen Probleme mit dem Aufruf von 'ssh' nicht berücksichtigt! Wundern Sie sich also nicht, wenn das nicht auf Anhieb funktioniert! Wenn Sie sich ein funktionierendes Parallel-Skript anschauen wollen, hier ist eines: ∼/HTK-KURS/TOOLS/HEREST-PAR

Der folgende Aufruf dieses Programms, trainiert iterativ die HMM in hmm2/VM.mmf und verdoppelt gleichzeitig die Anzahl der Gaußglocken (Mixturen) nach jeder 4. Iteration (SPLIT=4). Vor jeder Verdoppelung (also auch alle 4 Iterationen) werden die HMM gegen das Development Test Korpus getestet (TESTMOD=4). Die Worterkennungsrate für die Start-HMM (StartMMF=hmm2/VM.mmf) wird mit 42% vorgegeben (InitialWA=4200), so dass der erste Test übersprungen wird.

```
nohup ./HEREST-PAR StartMMF=hmm2/VM.mmf InitialWA=4200 SPLIT=4 TESTMOD=4 >& /dev/null &
```

Das Skript erzeugt nach jeder Iteration ein neues Verzeichnis mit den trainierten HMM. Testergebnisse werden auch in diese Verzeichnisse geschrieben. Ein Log-File wird im Verzeichnis LOGS erzeugt.

Kapitel 3

Weitere Schritte

Im Rahmen dieses Proseminars ist uns leider nicht die Zeit gegeben, unser Basissystem weiter zu verfeinern. In diesem Kapitel soll aber zumindest eine Übersicht gegeben werden, wie die weiteren Schritte hin zu einer besseren Performanz aussehen könnten.

3.1 Optimierung der Parameter

HVite hat einige Parameter, die den Suchprozess erheblich beeinflussen, die wir in unseren Experimenten der Einfachheit halber auf Erfahrungswerte gesetzt haben. Klassisch müssen diese Parameter für jede neue Erkennungsaufgabe neu optimiert werden. Praktisch bedeutet dies, daß man eine geeignete Test-Iteration mit variierenden Parametern durchführt und daraus die optimalen Werte ermittelt.

Die wesentlichen Parameter sind:

- Pruning
- LM-Gewicht
- WordEndPenalty

Dabei ist zu unterscheiden zwischen Parametern, die monotones Verhalten oder 'Gipfelverhalten' zeigen. Beispielsweise ist zu erwarten, daß der Parameter 'Pruning' zu monoton fallenden Worterkennungsraten führt, waehrend der Parameter 'LM-Gewicht' mit Sicherheit einen optimalen Punkt aufweist ('Gipfel'). Fuer Gipfelverhalten können Verfahren wie die Gauss-Optimierung angewandt werden, bei monotonem Verhalten müssen geeignete Grenzwerte definiert werden, bei denen die Optimierung abbricht.

3.2 Mehr Daten

Bisher haben wir nur mit ca. 30 h Sprache das Training durchgeführt. Übliche Spracherkenner heutzutage arbeiten mit hunderten von Stunden. Im allgemeinen gilt immer noch die Regel, daß mehr Trainingsdaten auch bessere Ergebnisse liefern, vor allem wenn die Daten aus dem Anwendungsbereich stammen.

3.3 HMM-Typen

Bis jetzt haben wir mit sog. Monophonen HMM gearbeitet, d.h. jedes Modell repräsentiert genau eines von 48 'Phonemen'. Dabei wird nicht berücksichtigt, daß Laute in verschiedenem Links-Rechts-Kontext sehr unterschiedliche Ausprägungen haben (Stichworte: Formantverläufe, Vokalqualität, Assimilationen). Mögliche HMM-Typen sind:

- Diphone : Modelle mit Links- oder Rechts-Kontext
- Triphone : Modelle mit Links- und Rechts-Kontext
- Silben: -

Bei der Erweiterung von Monophonen zu Tri- bzw. Di-Phonen (meistens wird eine Mischform aus beiden verwendet) ergibt sich das übliche Problem der 'data sparsity', d.h. daß es nicht genügend Vorkommen für alle denkbaren Tri-Phon-Kombinationen im Trainingsmaterial geben kann. Deshalb werden Techniken wie das 'binden' (tying) von Substrukturen oder sogar ganzen HMMs angewandt, ähnlich wie wir es mit dem 'sp' Pausenmodell bereits kennengelernt haben. Entscheidend ist dabei die Auswahl der zusammengefassten Strukturen. Das kann automatisch durch Clusterverfahren oder durch sogenannte 'Decision Trees' erfolgen. HHEd bietet zu beiden Techniken eine Vielzahl von Spezialbefehlen zur Manipulation der HMM an.

3.4 Language Modelle

Es hat sich immer wieder gezeigt, daß das LM einen ganz entscheidenden Einfluss auf die Performanz hat. Daher gibt es einen ganzen Forschungszweig, der sich nur mit der Modellierung auf dieser Seite beschäftigt. Alternativen zum einfachen Bigram-Modell sind:

- Tri- bis Pentagram-Modelle
- Modelle mit variablem Kontext
- Statistische Parser

HTK bietet dazu spezielle LM Tools an; der Anwender muß selber dafür sorgen, daß HTK eine gültige Standard-Lattice zur Verfügung steht.