Centre for Cognitive Science
University of Edinburgh
2 Buccleuch Place
Edinburgh EH8 9LW
United Kingdom

Speech Recognition using Phonetically Featured Syllables

Todd A. Stephenson
todd@cogsci.ed.ac.uk

22nd September 1998

Masters Thesis

Abstract

Speech can be naturally described by phonetic features, such as a set of acoustic phonetic features or a set of articulatory features. This thesis establishes the effectiveness of using phonetic features in phoneme recognition by comparing a recogniser based on them to a recogniser using an established parametrisation as a baseline. The usefulness of phonetic features serves as the foundation for the subsequent modelling of syllables. Syllables are subject to fewer of the context-sensitivity effects that hamper phone-based speech recognition. I investigate the different questions involved in creating syllable models. After training a feature-based syllable recogniser, I compare the feature based syllables against a baseline. To conclude, the feature based syllable models are compared against the baseline phoneme models in word recognition. With the resultant feature-syllable models performing well in word recognition, the feature-syllables show their future potential for large vocabulary automatic speech recognition.

The larger project work of this paper will appear in "Speech Recognition via phonetically featured syllables", King et al. (*Proceedings of the International Conference on Spoken Language Processing, 1998*).

Contents

Chapter 1

Introduction

1.1 State-of-the-Art Speech Recognition

State-of-the-art continuous speech recognition is typically done with hidden Markov models. These statistical models have been used to do recognition of both whole words and of context-dependent phones, also known as triphones. Hidden Markov models (HMMs), described in Chapter 4, enable robust speech recognisers to be built even in cases where there is a deficiency of certain training data. In these cases of insufficient data, a competitive recogniser can be built by using related training samples to make each model robust.

The speech signal is very complex, and it would be hard to use the signal itself in constructing a recognition model. Therefore, it must be parametrised. There are many ways to parametrise the speech signal for the HMMs. Linear prediction coefficients (LPCs) and mel-frequency cepstral coefficients (MFCCs) (Rabiner and Juang 1993) are two possible methods. Parametrisations provide a compact set of numbers to describe the speech. Not only are they more compact but they efficiently describe the speech. The parameterisation contains more than just the signal energy at a point in time; it contains the properties of that speech at that point in time–such as the shape of the spectrum. The parametrisation used in baselines models for this work is described on page 9.

Given a set of HMMs, a language model is used to indicate the probability of the different models occurring in the context of others. On page 19 I show the application of how a language model is used in recognition.

1.2 Phonetic Features and Speech

In addition to the the LPCs and MFCCs mentioned in section 1.1 phonetic features can also describe the speech signal. So for each successive frame of speech, the features for that frame are specified. This is referred to as a feature stream: for a series in time, you have a sequence of values that the speech went through. For instance, consider the word "sails". The feature stream for VOICENESS in "sails" would be an initial **voiced** in /s/ followed by **unvoiced** throughout the rest of the word. Additionally, the feature stream can be in a state of flux; when going from the unvoiced /s/ to the voiced /a/, there is a period of time when the speech has both voiced and unvoiced characteristics. Deng and Sun (1994) have done work in using articulatory features for speech recognition. Their features are position of the lips, the tongue blade, the tongue dorsum, the velum, and the larynx; these features are based on prior work in speech synthesis (Browman and Goldstein 1990). See Table 5.2 on page 22 for how their results compare with this current work.

In this thesis, my purpose is not to investigate what the best feature set is in representing speech. Rather, it is, using a given a feature set, to see the feasibility of performing speech recognition of syllables. Deng and Sameti (1996) have shown that speech recognition using features as observations has great potential. So, I will show that features can also be used for syllable recognition.

1.3 Prior Work in Syllable Recognition

Syllable recognition is based on the same basic concepts as phones but uses a larger sub-word unit than phoneme models. Continuous speech recognition has the task of recognising the component words of a spoken utterance. While models can be constructed for whole words, models of sub-word units have proven more realistic in constructing robust speech recognition systems. See the discussion in section 4.3 on page 17 for why sub-word units are beneficial.

The feature set in this thesis is based on past syllable work (Kirchhoff 1996). So, since she showed a feature set that works well in syllable recognition, I used her set, modified for English, to do my continued work in syllable recognition. She is using the phonetic features approach to modelling speech. Using a set of six multi-valued features (each ranging from two to eight values), she designed a system for recognising German syllables. Her results will be presented in Table 6.10 on page 33.

1.4 Outline of Dissertation

To begin, I will discuss the data I will be using and creating. Then, I will discuss how I built a feature detector. I will give a general overview to HMMs in Chapter 4. This will be followed in Chapter 5 of how this feature detector does in performing phone recognition; this will include a comparison to a similar phone recogniser based on MFCCs (mel-frequency cepstral coefficients, see page 9). This will prepare for the work presented in Chapter 6; having seen how feature streams work in phone recognition in comparison to MFCCs, this chapter will show how the two methods work in doing syllable recognition. Chapter 7 will analyse the systems presented in this paper from seeing how they both perform in word recognition.

Chapter 2

Data Preparation

2.1 The Corpus

TIMIT (Garofolo 1988) was used as the speech corpus for training and testing all of the systems in this project. It is composed of different speaker types, as given in Table 2.1. Each speaker spoke five phonetically-compact sentences (SX) and three phonetically-diverse sentences (SI). Each speaker also spoke two dialect sentences (SA), but these were not used in this thesis. The SX sentences were created so as to have a wide coverage of possible phone sequences. The SI sentences were taken from actual text corpora. Each SI sentence was spoken by only one speaker whereas each SX sentence was spoke by seven different speakers. In using TIMIT, I divided it into the standard training and testing sets. The testing set has no speakers nor sentences in common with the training set; the test set also has the standard core test subset, which was used when I could not use the full test set for a lack of time. The testing set is used only for final evaluation of a trained system. Now, in training a system, it is useful to have a validation set, an unseen set of data to measure the progress of the training. By determining the performance of the system on the validation set, the system parameters can be optimised accordingly. That is, the system is trained on a large set of data but a separate set of data is reserved so that I could see how the system responds to data it was not trained on. The validation set is never used in any of the training. I used various validation sets in training the systems; they were all formed by removing some of the utterances from the training set and reserving them for validation purposes only. For the phone recognition experiments, a set of 100 randomly chosen utterances from the training set were used in validating both the feature detectors and the phone recognisers. As the project progressed, I saw the need for a more systematically chosen validation set; so, for the syllable recognition experiments, I restarted the whole experiment with a new validation set. In his new validation set, 112 utterances were chosen such that none of them have any speakers or utterances in common with the rest of the training set. They were also chosen so as to have their distribution among dialect regions and between sexes be approximately the same as TIMIT as a whole. The one possible drawback for this approach is that the SX (phonetically-compact) sentences in the validation set occur multiple times. Within the validation set, each SX sentence will occur 7 times. So, while the validation set is different from the training and testing sets, it lacks variety. With either validation set used, I did not use any of its utterances in the training or actual testing. Table 2.1 gives the distribution of the data in TIMIT. The figures in Table 2.1 are for the training/validation sets used in the syllable experiments; the counts for the same sets with the phone experiments are similar. However, with the phone experiments, the validation set had neither unique speakers nor unique utterances from the training set.

		Validation		Training		Testing	
		Count	Percent	Count	Percent	Count	Percent
Dialect Region	1	8	7.1%	288	8.1%	88	6.6%
	2	16	14.3%	584	16.5%	208	15.5%
	3	16	14.3%	592	16.7%	208	15.5%
	4	24	21.4%	512	14.5%	256	19.0%
	5	8	7.1%	544	15.4%	224	16.7%
	6	8	7.1%	264	7.5%	88	6.6%
	7	24	21.4%	584	16.6%	184	13.7%
	8	8	7.1%	168	4.8%	88	6.5%
TOTAL		112	100.0%	3536	100.0%	1344	100.0%
Gender	m	64	57.1%	2512	71.0%	896	66.7%
	f	48	42.9%	1024	29.0%	448	33.3%
TOTAL		112	100.0%	3536	100.0%	1344	100.0%

Table 2.1: Distribution of speakers in TIMIT (Garofolo 1988). The counts for each dialect region and each gender is given for all three sets. Also stated is the percentage that each count occupies in its respective set. NB: The training set above is the original training set provided with TIMIT but with the validation set removed.

2.2 Syllabification

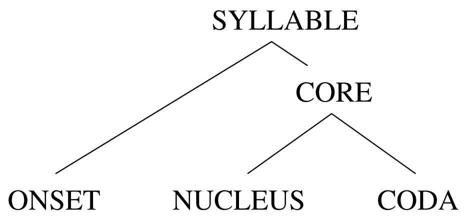

Figure 2.1: The syllable tree in Hyman (1975).

The syllables were divided up first into onset and core, and the core was then divided up into the nucleus and coda, as discussed in Hyman (1975) and shown in Figure 2.1. This resulted in up to three parts in each syllable: the onset, nucleus, and coda, to which I will refer to as *constituents* in this thesis. Every syllable has a nucleus (the vowel) while only some had an onset, a coda, or both. For the constructing, training, and testing of the HMMs, I will not be using the core constituent, but will define the constituents (of a syllable) to be onset, nucleus, and coda. In labelling the data, the following convention was used. The same phone markers were used as in TIMIT, and they are pasted together with an underscore, '_' between them. Additionally, the nucleus would always be surrounded by equal signs, '='. The purpose of the equal signs was to give a boundary between the constituents. So, the syllable p_r_=ey=_d would be a syllable with /p/ and /r/ in the onset, /ey/ as the nucleus, and /d/ as the coda. Note that even in the absence of a onset, coda, or both, the nucleus is still surrounded by a '=', giving syllables such as =ah=, t_=ey=, and =ih=_t_s.

In marking up speech, there are different levels of detail that can be employed. Table 2.2 gives the levels that I am concerned with in this work. TIMIT (Garofolo 1988) was a valuable corpus for this research. It is marked up on SURFACE level in addition to the WORD level. Since the labelling was done carefully by hand, it is accurate and therefore reliable for this work. Now, there is no markup on

```
WORD                                 "argument"

SYLLABLE           =aa=_r            g_y_=ah=              m_=ah=_n_t

LEXICAL            aa    r      g       y       ah      m     ah     n      t

PHONEME       q    aa           g       y       uh      m     eh     n      t

SURFACE       q    aa     gcl      g    y       uh      m     eh     n     tcl

CENTRALITY    SIL--Cent----Nil---------------Full---Nil---Full--Nil-------
CONTINUANT    SIL--Con------------------------------NonCon----------------
FRONTBACK     SIL--Back-----------------------------Front-----------------
MANNER        SIL--Vowel---Occ--------Aprx---Vowel--Nas---Vowel-Nas--Occ--
PHONATION     SIL--Voi-----------------------------------------------UnVoi
PLACE         SIL--Low-----Velar-------------High---Lab---Mid---Cor-------
ROUNDNESS     SIL--Unrou---------------------Rou----Unrou-----------------
TENSE         SIL--Tense---SIL---------------Lax----SIL---Lax---SIL-------
```

Figure 2.2: The Syllabification Process for a Syllable

the SYLLABLE level, however, and this is what I need to train syllable models. Syllable labels are not easily made. While TIMIT does provide a lexicon to help generate the LEXICAL level, the phonemes in the lexicon need to be grouped together as syllables. Furthermore, not all of the phonemes in the lexicon will appear in the SURFACE level (that is, they are deleted) and some extra phonemes will appear in the SURFACE level that were not in the lexicon (that is, they were inserted). So, for the work in King et al. (1998), Taylor developed an algorithm to align the SYLLABLE level labelling with the PHONEME (Note in Table 2.2 that the SYLLABLE level is composed strictly of the items in the LEXICAL level and that the items in these levels do not indicate what exactly was said). See Figure 2.2 for where these different levels fit together. The SURFACE level does not always line up with the LEXICAL level. For example, in Figure 2.2, the glottal stop /q/ is inserted in the SURFACE level, while an /r/ is deleted from the first syllable in the LEXICAL level. So, the syllabification algorithm first divides each WORD into SYLLABLES. This gives the lexical division of the word. The algorithm then groups the SURFACE phonemes according to which SYLLABLE it is determined they belong to. The algorithm is still in a development stage. Complicated insertions/deletions can cause it confusion, and as a results it will sometimes reject sentences for syllabification; so, there are a handful of TIMIT sentences which were not used for syllable training. It was determined that it would be better to have an algorithm that rejects a few sentences than to have one which makes bad judgements on some sentences.

So, the labelling I used for my experiments were formed as follows. TIMIT already provides the WORD and the SURFACE level labellings. The PHONEME level is merely formed by collapsing similar surface PHONES into a common form; for example, the /gcl/ and the /g/, the g-closure and g-release, respectively, in the SURFACE PHONEME level were collapsed into a /g/ on the PHONEME level. The LEXICAL is given by TIMIT; however, since it is just lexical, TIMIT does not give any time boundaries

1. WORD The full word that is spoken
2. SYLLABLE The lexical pronunciation of a syllable within a word. A syllable is composed of the labels in the LEXICAL level even if it is pronounced differently by an individual speaker.
3. LEXICAL The standard, phonemic way to pronounce the word. Only one pronunciation is defined for each word.
4. PHONEME The 39 phone set in TIMIT, where similar surface phones are collapsed together.
5. SURFACE PHONEME The actual phone pronunciation, taken from the 60 phone set in TIMIT.
6. FEATURE The feature-value of each surface phoneme.

Table 2.2: Levels in the Syllabification Process

for the LEXICAL level. So, to get the SYLLABLE level, which is built strictly from components at the LEXICAL level, Taylor's algorithm was used put to syllable boundaries around the appropriate phones in the PHONEME level. The SYLLABLE level labelling is used in the syllable recognition experiments while the PHONEME level is used in the phone recognition experiments. Below the SURFACE PHONEME level in Figure 2.2 are the eight acoustic phonetic feature levels used. They were each formed from the PHONEME level. For each feature, the respective value for the phone was taken. The list of Feature-Values is given in Table 2.3. For a complete mapping between phoneme level phones and their values, see Appendix A.

FEATURE	**Values**		
centrality	sil	cent	full
	nil		
continuant	sil	cont	noncont
frontback	sil	back	fr
manner	sil	appr	fric
	nas	occ	vow
phonation	voi	unvoi	sil
place	sil	cor	cdent
	ldent	glot	high
	lab	low	mid
	pal	vel	
roundness	sil	rou	unrou
tense	non-tense	lax	ten

Table 2.3: Acoustic Phonetic Features & Values

Mapping from the phoneme level to feature level is less than perfect for two reasons. One, it does not take into effect the finer phonetic detail available from the SURFACE level. For example, the unvoiced /gcl/ and the voiced /g/ were combined into a voiced /g/; so, the closure part of the PHONEME level /g/ is really unvoiced but will be treated as being voiced. This method was chosen as part of the work in King et al. (1998); for future work, I advise using the SURFACE PHONEME level to do the feature labelling. Second, it does not account for co-articulation. This model assumes that features change only on the phone boundary. In reality, this is not the case; the features change at different times from each other as phones assimilate with those around them. This needs to be dealt with in future research.

2.3 Cepstral Coefficients

In this work the mel-frequency cepstral coefficients (MFCCs) were computed with a Hamming window of 25 ms, each window shifting 10 ms from the previous. Thus, each window overlapped with others. A pre-emphasis coefficient of 0.97 was used. A 26 channel filterbank was used with a liftering parameter 22. These parameters are commonly used (see Young et al. (1996)). The result was 12 coefficient plus one energy value for each frame of speech.

Chapter 3

Feature Detection using Artificial Neural Networks

3.1 Introduction

In Table 2.2 on page 7 I showed the various ways a word is marked up in this work. Below the SURFACE PHONEME level are the phonetic feature levels. The goal of this work is to build a system that recognises syllables, based on the syllables feature stream. Each syllable has a feature-value stream (as defined in section 1.2 on page 3) that describe it. For automatic speech recognition, this labelling of the feature stream through the syllables needs to be done automatically. NICO (Ström 1997a) is used to build and train artificial neural networks (ANNs). NICO is specifically designed for speech work and is designed to do the recurrent, time-delay work that is useful for many speech recognition tasks. So, for each frame of speech, the ANNs will give a phonetic classification.

For this project, eight ANNs were used. Each one was trained to classify a different, multi-valued acoustic phonetic feature. The eight features and their respective values are given in Table 2.3 on page 8. Each frame of speech will be assigned one value from each feature set. So, for example, the phone /d/ would be classified as CENTRALITY:**nil**, CONTINUANT:**noncont**, FRONTBACK:**back**, MANNER:**approximant**, PHONATION:**voiced**, PLACE:**continuant**, ROUNDNESS:**unrounded**, TENSE:**non-tense**. For the TENSE net, any non-vowel is classified as **non-tense**.

3.2 Architecture of the Nets

All of the nets are recurrent, time-delay neural networks. A generic architecture for the set of ANNs is given in Figure 3.1 on page 11. The architecture used was based both on what was used in Ström (1997a) and in Stephenson (1998).

Regarding the overall architecture of the nets, they all had identical structure except for two items: (1) the number of units in the recurrent hidden layer and (2) the number of output units. Each has an input layer of 13 units: the MFCCs as described in section 2.3. There are then three hidden layers: one computes the delta values (the difference between the current input vector and the previous); the next computes the acceleration values (the difference between the current deltas values and the previous); and the final is the recurrent hidden layer, whose number of units varies for different features. The 13 units in the delta layer each receive a connection from the respective unit in the input layer while the 13 units in the acceleration layer, likewise, each receive a connection from the respective unit in

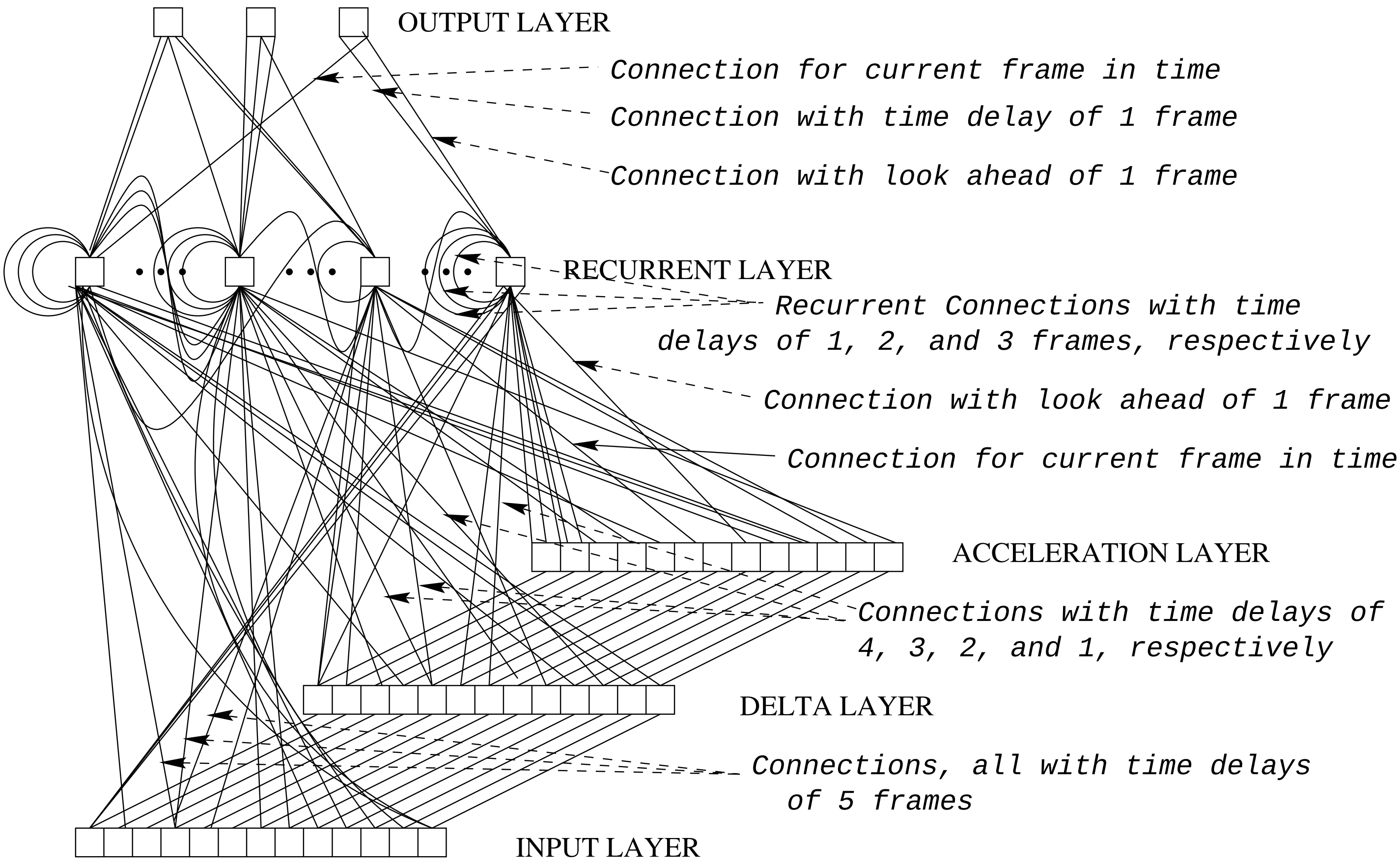

Figure 3.1: Architecture of a Neural Network. Some of the connections have been labelled with sample connection types, indicating the various types of time delays and look aheads.

the delta layer. The delta and acceleration layers give approximate first and second derivative values, respectively, for the given frame of speech.

The recurrent layer is the heart of each ANN. It receives connections from the input, delta, and acceleration layers, which I will from now on refer to as the input group. These connections from the input group to the recurrent layer are both time-delay and look-ahead connections. They cover a period of [-5,+1] frames from the current frame. This value was taken from the example net given in the NICO manual; in future work, it would be worth investigating whether this window should be shifted more to the right (that is, take in more look-ahead context and less time-delay context). By its definition, the units in the recurrent layer also make connections with other units, including themselves, in the recurrent layer. Again, these are specified to have a window, in this case of [-1,-3]; this takes in 3 frames of left context.

The output layer of each net has one unit for each possible value of the given feature. This layer receives connections from the recurrent layer, also with time context. Connections are made with a context of [-1,+1], taking in one frame in both the right and left context.

Stephenson (1998) uses networks with hidden layers of 20, 40, or 80 units; the number of units depends on the complexity of classifying the given feature (see Table 3.1 on page 12). Those nets were fully connected (except as noted within the recurrent layer). Ström (1997b), however, notes the benefits of using large, sparsely connected networks; he explains that networks with large number of sparsely connected units perform better than smaller, fully connected networks with the same number of total connections. So, in determining the architecture of the nets, I wanted to have a lot more units than before without having a big increase in the number of connections. So, based on the number of connections that were in Stephenson (1998) I constructed the current nets; however, I constructed nets with 25% connectivity.

The connectivity points are determined by random. That is, for each possible connection, a function generates a random number between 0 and 1; if the number is less than or equal to 0.25, a connection is made. With 25% connectivity, I constructed nets that had 100, 150, 200, and 300 hidden units. These roughly correspond to the number of connections in the nets that were used in Stephenson (1998). The 100, 150, and 200 hidden unit sparsely connected nets had at least the same number of connections as the 20, 40, and 80 hidden unit fully connected nets, respectively (Table 3.1). Note that FRONTBACK and MANNER both increased from 80 recurrent units to 200 recurrent units. However, PLACE increased to a larger 300 recurrent units to account for the extra two values that it now had. For those features which did not appear in the original paper (TENSE and CONTINUANT), I chose a net size based on my own intuition based on previous work with the nets.

Within the recurrent layer, there is not a straightforward 25% sparse. Rather, a "spread coefficient" (Ström 1997a) of 25 is specified. This is not a percentage number, however. NICO uses it to determine which connections should be made, based on the distance between the two concerned units in the recurrent layer. For example, a connection between, say, unit 3 and unit 4 would more likely be established than a connection between, say, unit 3 and unit 20. So, while the recurrent layer is sparsely connected, there is a greater abundance of connections between close units.

	Nets in Stephenson (1998)				Current Nets			
	Values	Sparse	Connections	Recurrent Units	Values	Sparse	Connections	Recurrent Units
centrality	4	100%	7182	20	4	25%	18798	100
continuant	N/A	N/A	N/A	N/A	3	25%	29641	150
frontback	4	100%	36283	80	3	25%	40837	200
manner	6	100%	37449	80	6	25%	41301	200
phonation	3	100%	15616	40	3	25%	29641	150
place	9	100%	39102	80	11	25%	65031	300
roundness	3	100%	6422	40	3	25%	18716	100
tense	N/A	N/A	N/A	N/A	3	25%	40837	200

Table 3.1: Size of the various ANNs. *Sparse* refers to the connections between the input group and the recurrent layer and between the recurrent layer and the output layer. In both sets, the recurrent layer is sparsely connected.

To summarise the connectivity within the net, the whole net is sparsely connected. Connections going to or from the recurrent layer have a simple 25% sparse connectivity. However, with the recurrent connections, units that are farther away from each other have fewer connections between themselves while the closer units have more connections. This setup will cause closer recurrent units to specialise together in certain classification areas because of the greater number of connections between closer units; this was done because a simple sparse connection among all of the recurrent units would make the network too big to work with easily (Ström 1997a). Furthermore, the units in the recurrent layer will further specialise because they only have certain connections to the input group and to the output layer. Take the PHONATION net, for example. Because of the time window, there can be up to three connections between any given unit in the recurrent layer and a unit in the output layer.

So, with a 25% connectivity, there will most likely be some recurrent units which do not have any connections to the, say, **unvoiced** output unit; so, those units may find their specialisation in **voiced** or **silence** and not **unvoiced**. They can still have impact on **unvoiced** classification as they may have recurrent connections to units which themselves have connections to the **unvoiced** output unit.

3.3 Training of the Nets

Iteration	Global Error	Validation	
		%Correct	Error
1	9.044e-01	83.3	7.909e-01
2	7.824e-01	83.6	7.996e-01
3	7.448e-01	84.5	7.447e-01
4	7.254e-01	84.4	7.381e-01
*** Reducing gain to 5.00e-06. Reduction number 1 (max 10)			
5	6.853e-01	85.6	6.884e-01
6	6.774e-01	85.3	6.885e-01
*** Reducing gain to 2.50e-06. Reduction number 2 (max 10)			
7	6.550e-01	86.1	6.674e-01
8	6.494e-01	85.8	6.697e-01
*** Reducing gain to 1.25e-06. Reduction number 3 (max 10)			

Table 3.2: The first 8 iterations from training the CONTINUANT net. For each iteration, Back Propagation is run for each utterance in the training set. After each iteration, the global error is computed on the training set. Additionally, the correctness and error is computed for the validation set; when the validation set's performance starts to degrade, the gain is cut in half.

Back-propagation through time was used as the training algorithm. The initial gain was set to 1.0e-05 and the momentum to 0.95. The gain determines how much to update a weight if it needs to be corrected. If a weight is determined to be too large, the gain amount will be subtracted from it; likewise, if a weight is determined to be too small, the gain amount will be added to it. The momentum value affects how much the current update in weight is affected by previous update in weights; that is, if the updates have been going in a certain direction, that direction has an amount of *momentum* which takes some time to stop (see Rojas (1991, sec. 8.1.1)). My experience with NICO is that the gain needs to be set to a low value, such as the one above, so that the net can properly train; a high gain would not allow the net to train correctly. Also, from my experience, the momentum can be set high, such as the value above, as long as the gain is sufficiently low. The validation set was used in training to adjust the gain when when a test run of the net on the validation set showed that the net was overlearning. When the net was overlearning, the gain would be cut in half and training would continue. Table 3.2 shows the effect of cutting the gain in training.

3.4 Analysis of Test Results

Table 3.3 shows the test results for each of the feature ANNs. The "% Reduction" column shows that the nets which learned the best are MANNER and PHONATION, both of which decreased their error by almost four-fifths. The nets which learned the worst of the set are FRONTBACK, ROUNDNESS, PLACE, and TENSE, all of which did not even cut their error by even two-thirds. CENTRALITY and

Feature	Ceiling	Actual Error	% Reduction
centrality	52.8%	14.6%	72%
continuant	54.7%	13.9%	75%
frontback	40.8%	15.9%	63%
manner	65.5%	13.5%	79%
phonation	36.5%	7.5%	79%
place	75.2%	27.6%	63%
roundness	21.5%	8.2%	62%
tense	34.5%	12.6%	63%
OVERALL	N/A	46.7%	N/A

Table 3.3: Test Results for Feature ANNs. For each feature, the overall error is stated. Also given is the error that would be given if all frames were given the most common value ("ceiling"). The *% Reduction* column is an indication of how well the net learned its task. It shows how much it how much the net learned beyond just doing ignorant guessing. This is not a standard measure of determining the value of an ANN; but it gives a measure for comparing amongst the nets which have different types of training data. The OVERALL error states the number of frames where at least one of the nets classified incorrectly.

CONTINUANT both had an average gain, in comparison to the other features, as they both cut their error by almost three-fourths.

The nets made intelligent confusions when they made errors. Take the feature PLACE. In continuous speech, the place of articulation is constantly changing. Whenever, changing from a consonant to a vowel, the place of articulation is going to change as the feature values for the vowels are distinct from those of the consonants. This brings in a high level of co-articulation. While I did not have the time to do a study of the full TIMIT test set, the full set of feature streams for this specific sentence (Appendix B) suggests that the PLACE feature changes its values a lot more rapidly than the other features do. Figure 3.2 on page 15 shows the types of confusions that the nets make. For example, take the syllable th_=ih= in Figure 3.2 on page 15. The correct value for the onset /th/ is coronal-dental while the correct value for the nucleus /ih/ high. The net is confused about how to classify both the onset and the nucleus. It thinks that the onset is either labial-dental, coronal-dental, or coronal; so, it makes its confusions among values that are similar. Similarly, with the /ih/, the net confuses it with **high** and **mid**, which also similar values. Table 3.4 on page 15 gives the full confusion matrix for the PLACE net. It verifies that over the entire test set, PLACE does get confused over similar places of articulation. See Appendix C for all of the confusion matrices from the TIMIT test set.

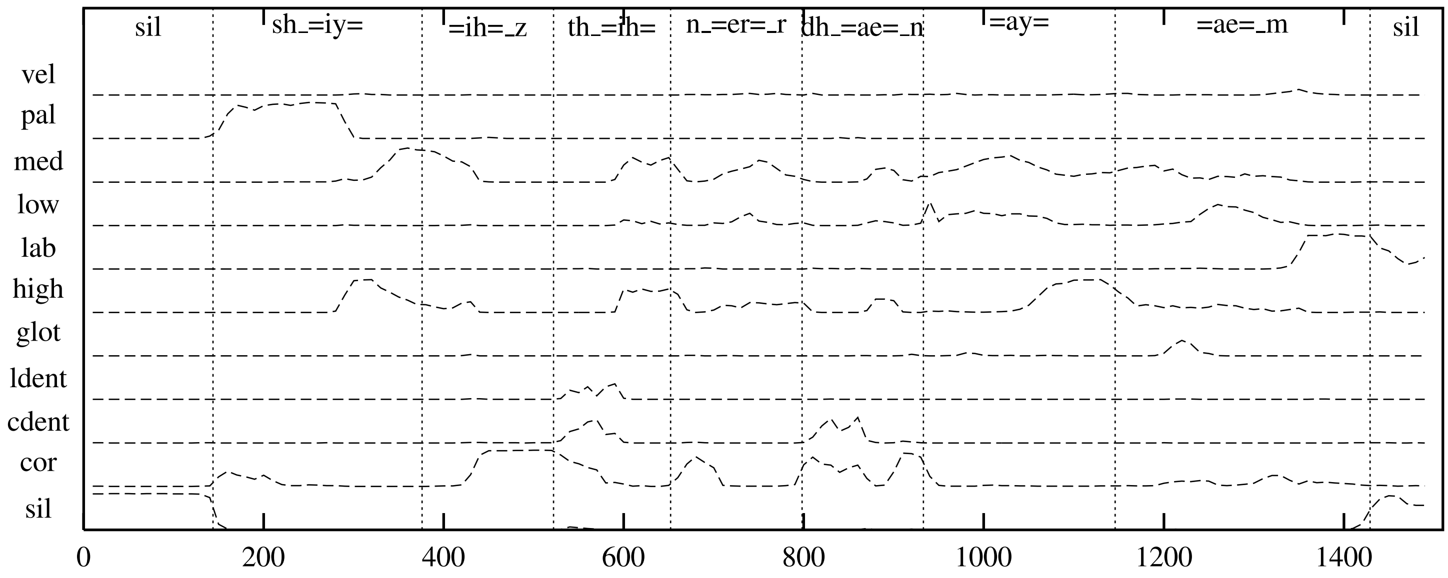

Figure 3.2: Feature Stream for place

	sil	cor	cdent	ldent	glot	high	lab	low	mid	pal	vel
sil	89.4	4.0	0.3	0.4	0.3	1.0	1.3	0.7	1.0	0.1	1.5
cor	1.9	80.4	0.7	0.7	0.1	4.0	2.8	1.6	2.6	1.1	4.2
cdent	4.9	29.7	43.0	8.0	0.2	5.2	3.1	1.3	1.3	0.1	3.2
ldent	2.7	10.3	2.6	73.3	0.6	2.0	4.1	0.9	0.7	0.5	2.1
glot	11.7	9.4	0.5	2.2	58.0	4.8	2.0	2.5	2.8	0.8	5.1
high	0.5	6.7	0.3	0.2	0.1	70.9	1.1	5.9	11.3	0.2	2.8
lab	2.8	13.3	0.6	1.1	0.3	2.3	74.2	0.9	1.8	0.1	2.7
low	0.8	6.5	0.2	0.4	0.2	12.4	1.3	51.3	24.6	0.1	2.2
mid	0.5	5.6	0.1	0.3	0.1	18.3	1.4	15.7	55.9	0.1	2.1
pal	0.9	18.0	0.0	0.7	0.1	1.1	0.1	0.1	0.2	77.9	0.7
vel	1.8	13.0	0.3	0.5	0.4	7.4	1.8	1.5	2.8	0.2	70.4

Table 3.4: Confusion Matrix for PLACE

Chapter 4

Hidden Markov Models

4.1 Introduction

The state-of-the-art system in doing speech recognition is to use Hidden Markov models (HMMs), (Rabiner and Juang 1993, ch. 6), which are stochastic models of speech. A hidden Markov model consists of a set of states and a set of transitions between certain states (see Figure 4.1). Each state has its own probability density function (pdf) that is used to determine the probability that a given frame of speech is generated by that state; this pdf is described by a means vector and a variance vector. Furthermore, the transitions between the states have probabilities of being used. So, each HMM generates a sequence of observation vectors, each vector having a probability. The probability over the HMM is calculated by taking the product of each transition traversed along with the probability of all the observation vectors generated. For continuous speech recognition, multiple HMMs will be concatenated. Probabilities are also utilised in determining which string of HMMs to use. In concatenating the models, there needs to be probabilities of going from one model to another. These probabilities are encoded in the language model, which says how likely certain occurrences of words (or sub-words) are.

An HMM is traversed by passing through one state for each frame. As a state is entered, a probability is generated from the pdf according to the likelihood that this state generated the current frame. Before processing the next frame of speech, an transition must first be followed. This transition can either go onto a successive state, back to the same state, or backwards to a previous state. Except where noted in the case of the phoneme models *sil* and *sp*, all of my HMMs had transitions that only lead to the succeeding state and back to the same state; this means there were no skip transitions and no backwards loops.

In addition to each HMM having two parts to it, the states and the arcs, the states have multiple parameters. For the purposes of these experiments, the states have two vectors: a mean vector and a

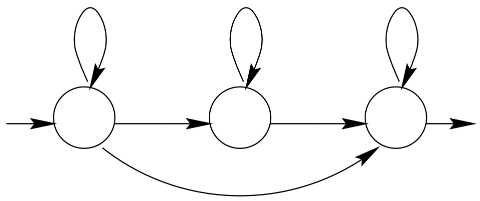

Figure 4.1: A three-state HMM.

variance vector. These vectors define the pdf.

4.2 Training

A large set of labelled training files is needed to train an HMM. Preferably, the labels should have time boundaries and not just the transcriptions. Once you have a set of labelled speech files with time boundaries, the parameters in each HMM can then be estimated.

Initial training is done on isolated models. That is, a model exists for each unit that is to be recognised. Using the time-labellings, the appropriate segments of speech from the training files serves as the training observations. For example, to train a model to recognise /ae/, the time labels indicate which frames of speech had /ae/ utterances. These /ae/ utterances are extracted from the speech and used for training the isolated model for /ae/.

On these isolated models the first thing to do is Viterbi training (Rabiner and Juang 1993). The Viterbi algorithm assigns each frame of data to a state in the HMM. This gives a state sequence for each training data. This process also determines the likelihood that the HMM with its current parameters models that training data. This process is repeated until the likelihood can not be increased.

After doing the Viterbi training on each model, their parameters are re-estimated using Baum-Welch training (Young et al. 1996). It applies the Forward-Backward algorithm which gives the probability of each frame being generated by each state. These probabilities are then used to update the HMM parameters.

Even with accurately transcribed data, there will be slight errors in the time boundaries. So, the isolated training above will not always give the optimum level of training. Therefore, after training on isolated models, embedded training is employed for further re-estimation. In embedded training, the isolated models are trained in the context of each other. That is, for each utterance in your training set, all of the models that are indicated in the labelling of that utterance are concatenated together. This creates a model for the whole utterance and uses the concatenated set of HMMs to do Baum-Welch re-estimation. The Forward-Backward algorithm then trains the models using its own judgement for where the model boundaries occur.

4.3 Clustering

It is impossible to get enough training data of whole words to train all the whole word models. In any given speech corpus, there will be words that are common and words that are rare. While training whole word models for the common words will not be a problem, training whole word models for the new, unseen words is impossible. Using sub-word models instead of whole words permits the addition of new words to the recognition dictionary. That is, by piecing together different sub-word models, a model is made for a word that has never been seen in training. Syllable recognition uses the same concept as phone models, but on a different level. Most words can be created with a finite number of syllables. So, like syllables from different words can be pooled together to make a model for each syllable. Also, the different syllable models can be used to synthesise unseen syllables, and, therefore, unseen words. This is the reason why sub-word models, particularly phone models, are so prevalent. Since there are only 10,000 syllables in English (Rabiner and Juang 1993, chap. 8), we can easily collect enough continuous speech in order to train a system to recognise all of these syllables. While there will be syllables that are not covered in the sample, models for them can be synthesised from the existing models, as shown in Table 6.13 on page 35. That is, these syllables will have similarities to other syllables in the training data. In these cases, data from related syllables can be pooled together

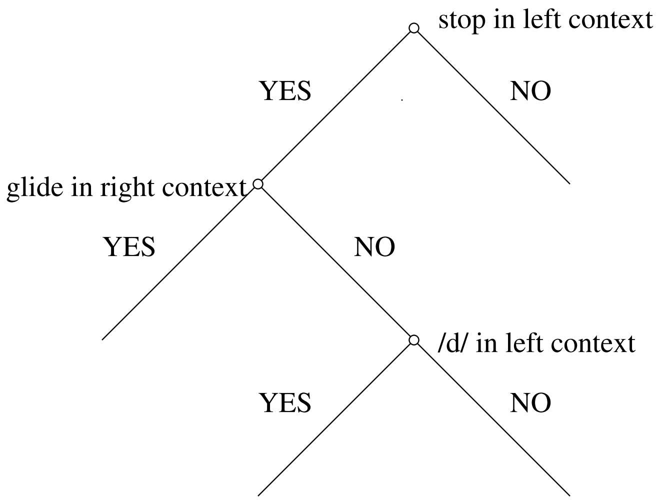

Figure 4.2: Decision Tree for triphones.

for robust training. For example, say that the word /k_=ae=_t/ occurs infrequently in training (see page 6 for a description of this syllable labelling convention); for a more robust /k_=ae=_t/ syllable, we can use the /k/ from /k_=uh=/, the /ae/ from =ae=_m, and the /t/ from /n_=ih=_t/ to obtain related data from training the syllable /k_=ae=_t/.

The training in section 4.2 is sufficient for doing small model sets where each model has enough training samples for robust training. However, when using larger sets of models, such as with triphones (context-dependent phones) or syllables, there will not be enough training data for all of the models. In these large sets of models, there will be models that have parts of themselves in common with each other. Take the triphone r-aa+n; this HTK-style notation means that it is model of the phone /aa/ in the context of /p/ to its left and /n/ to its right. In a set of triphones, there would also be a triphone for p-aa+n. Now, consider that there is enough data to train either of these two triphones. One way to increase the data available for both of these triphones is to cluster at least one of their states. That is, let certain of their states be defined by the same pdf. In this case it would make sense to cluster the final state of the two HMMs because they both have a right context of /n/. So, whenever, say, r-aa+n is trained, the final state of p-aa+n will also be updated since it shares the same final state as r-aa+n; likewise, when p-aa+n is trained, the final state of r-aa+n will be benefit.

To determine which states to cluster, I use decision tree clustering. A decision tree is a binary tree with queries at each node. These queries are phonetic questions. For example, one question in the case of phoneme models would be “Does the model have a stop to its left context?” Figure 4.2 gives an example of a decision tree. The ordering of the questions on the tree are determined both according to the distribution of the training data and on the similarity between the training data. That is, the questions in the decision tree will be constructed so that similar models (those whose pdf’s are judged close by a given distance metric) end up close to each other on the decision tree; starting at the top of the tree, nodes will be created as long as some of the clusters fall below a specified minimum number of training frames.

4.4 Recognition

To do recognition, Viterbi alignment (Young et al. 1996) is done. A bigram language model is employed. A bigram states the probability of pairs of units (such as words, phones, or syllables) occurring together. That is, the phrase “the man” will occur a lot more often in a corpus than the phrase “the and”; in fact, the second phrase will probably never occur in the corpus. So, say we have recognised the word “the”; the word “man” will be a lot more probable than “and” as the next word to recognise. For the syllable models, I constructed a bigram by looking at the occurrences of syllables or phones in the training data. There will be many syllable combinations that do not occur in the training data (notably, those syllable which are only found in the testing or validation data). So, for those combinations that occur zero times in the training data, a unigram backoff model is used. In these cases, they will be assigned the unigram probability of 1; that is, they will be given the probability as if the syllable only appeared once in the training data. A unigram does not take context into account; it does not matter what the previous word was. For phone recognition, I did not compute it myself but was provided with a backed-off bigram for TIMIT phones.

Chapter 5

Phone Recognition

5.1 Introduction

The standard method of doing speech recognition today involves using context-dependent phones, also known as triphones. That is, a set of models that recognise, for example, the phoneme /p/ with /ae/ to its left and and /r/ to its right; there would then be a separate model for recognising /p/ when it occurs in other contexts, such as with /m/ to its left and /s/ to its right. This is due to the co-articulation (Catford 1988) that occurs between neighbouring phones.

5.2 Training Phone HMMs

The first step to this was to train a set of monophone models: for the 52 TIMIT phones (the stop closures were combined with their respective stop releases) and also for *sil* and *sp*. Table 5.1 details how I create and trained the cross-word triphones. Each phone was the standard three-state model with transitions with no skip or reverse transitions allowed, except as specified above for *sp* and *sil*. The first step to training triphone models is to create monophone models. These then serve as the basis for triphone models. However, there is not enough data to train each triphone model. So, similar states of triphones of a phone are clustered together so that their data can be pooled for robust training. For these models I used an already prepared set of queries, modified to include all the phones in my set. Only triphones of the same phone can be clustered. So, a triphone of /p/ can not be clustered with a triphone of /b/. However, a triphone of /p/ can be clustered with a similar triphone of /p/.

5.3 MFCC Phones

To determine the value of using the feature streams, I first had to train a similar set of models that were trained on the standard MFCCs. This will set a baseline for measuring the effectiveness of the feature set I have used.

The MFCC phone HMMs were tested with the full TIMIT test set. The 52 phones were at this point collapsed down to the standard number of 39 phones; that is, if, say, the phone /ix/ were recognised, I would consider it as if it had recognised either /ix/ or the similar /ih/. Table 5.3 gives the error for the recogniser.

1. Train isolated monophone models
    (a) Do Viterbi training on each model to estimate the means, variances, and transitions for each state.
    (b) Do Baum-Welch training on each model to further re-estimate the means, variances, and transitions for each state.
2. Perform embedded training monophone models
    (a) Do five iterations of Baum-Welch embedded training on all models.
    (b) Fix the *sp* and *sil* models. *sp* is given one emitting state and is given a transition to skip that state. This state is tied to the centre state of *sil*, which is given transitions in both directions between its first and last states.
    (c) Do two iterations of Baum-Welch embedded training on all models.
3. Create triphone models
    (a) Clone each monophone into all respective triphones found in the training set.
    (b) Tie transition matrices of all triphones of the same phone.
4. Cluster triphone models
    (a) Do two iterations of Baum-Welch embedded training on all models.
    (b) Do decision tree tying of the triphones.
    (c) Synthesise unseen triphones using the decision tree.
5. Train embedded triphone models
    (a) Do three iterations of Baum-Welch embedded training on all models.

Table 5.1: Creating cross-word triphones

5.4 Feature Phone HMMs

The feature phone HMMs were trained in a similar manner to the MFCC phone HMMs above. The only difference is in that the feature HMMs used a different observation vector than the MFCC HMMs. While the MFCC HMMs used the MFCC coefficients, the energy values, the first derivatives, and the second derivatives, the feature HMMs used the feature streams from the artificial neural nets as their observation vectors. Furthermore, no delta or acceleration values were put into the HMMs because these are already accounted for in the neural nets themselves. Table 5.3 on page 22 gives the error rate for the phones trained from the features.

On page 3, I mentioned the work already done by Deng and Sameti (1996) in using features for phone recognition. Using such features, they produced competitive error rate of 26.46%, as shown in Table 5.2.

Accuracy	73.54%
Error rate	26.46%

Table 5.2: Error Rate for 39-Phone Set Phone Recognition in Deng and Sameti (1996) on 24 speakers in the TIMIT test set.

5.5 MFCC vs. Feature Phone HMMs

The MFCC and the feature phones were trained in identical manner and produced virtually identical results. The figures for both sets do not represent the optimum amount error possible. The purpose of this experiment was not so much to achieve the lowest error possible in the literature. Rather, it was to take the standard procedure for creating triphones Young et al. (1996), and create both the standard MFCC and the new acoustic phonetic feature HMMs to see how the feature HMMs measure up against a standard.

MFCC	Feature
37%	36%

Table 5.3: Error rates for triphone HMMs using the 39 phone TIMIT set. Figures are from the full test set.

Chapter 6

Syllable Recognition

6.1 Introduction

The idea of using larger units than phones is not new. Many speech recognition systems in production recognise whole words. However, these systems have small vocabularies, which make it more practical to train models for whole words as opposed to individual phones. However, when dealing with large vocabularies, it is impractical to train models for each word. As discussed in Chapter 4, this is because (1) it would be hard to gather enough samples of each word to build a robust recogniser and (2) it would be impossible to synthesise new words.

6.2 Questions in Constructing Syllable Models

6.2.1 Topology

An important part of speech recognition is how to model the speech. In this research, I chose hidden Markov models because it is a proven way to do speech recognition and because there are already tools (i.e., HTK) available to train HMMs and use them in speech recognition. Having decided on HMMs, I needed to decide how to best set them up to provide the best syllable models. As shown in Figure 4.1 on page 16, an HMM is composed of a number of states and of transitions between certain states. So, I needed to decide (1) how many states to have in the syllable models and (2) which states to have transitions between.

6.2.2 Clustering

In any speech recognition system, each model needs to have a large pool of training data. In section 4.3 on page 17 I showed the need for clustering data in building robust models. In constructing the phone models in Chapter 5, I was following a standard recipe for what to cluster. However, the data for the phones was being clustered based on the phones' contexts while the syllables are not defined according to their context because one of the reasons I am building syllable models is because they should, in theory, not be affected much by their context. So, in clustering the training data, I need to decide what criteria will be used to determine if training data for multiple HMM states should be pooled together. Furthermore, I need to determine how big the clustered pools of data should be. If the pool of clustered data gets too big, the data in the pool will become too varied. However, if the pool is too small, then the model will not be robust enough.

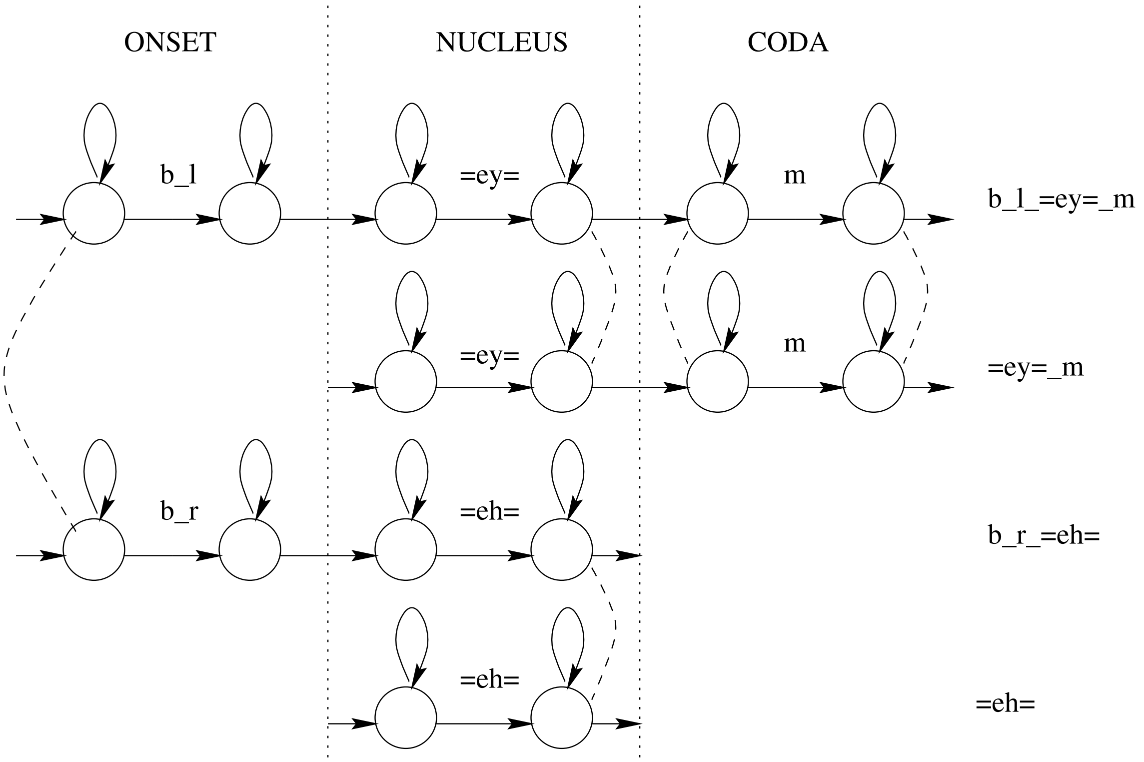

Figure 6.1: A hypothetical set of HMMs with two state constituents with a possible tying included.

6.2.3 Training

Just as the topology and the clustering discussed above are important to the performance of the speech recogniser, the training method needs to be done well. A proper algorithm for training needs to be established so that the models represent the training data well. However, care needs to be taken that the models do not represent the training data too closely but that they perform well on new speakers and utterances.

6.3 Solutions to Constructing Syllable Models

All of my experiments into the optimal HMM topology were done on the feature HMMs, not on the MFCCs. This is because the purpose of this work is investigate the feasibility of doing phonetically featured syllable recognition. So, after investigating the best topology for feature HMMs, I will fix the topology type and use that for the MFCC HMMs. Future work would include whether MFCC syllables are better with a different topology.

6.3.1 Topology

For each syllable, a different HMM topology was used, depending on how many constituents it had. A set number of states was put into each HMM, depending on which constituents it had in it. The most basic trial was to put one emitting state in an HMM for each constituent in the syllable. There were no skips allowed amongst the states. That is, the only transitions within the HMM were self-connecting transitions and transitions going from states to the successive states. See Figure 6.1 for a what the HMM topology looked like for two state constituents.

The initial topology of one state per syllable constituent was trained according to the method described in Table 6.7 on page 32 and a recognition experiment was performed using the sentences in the validation set. The results for this run are in Table 6.2 on page 26.

With such a high error rate 58.93%, I thought that perhaps the 1/2/3 state models were not capturing all the complexity of the syllables. For example, syllables with complex codas and complex onset and a diphthong nucleus, such as k_l_=ay=_m_d, would have to be represented by only 3 state model, the first state designed for k_l, the second for =ay=, and the final for m_d. Furthermore, the standard topology for phone recognition is to use a 3 state model (Young et al. 1996). So, a syllable model, which is naturally bigger than phones, should have more than 3 states. So, as a first step, I tried putting 2 states into each syllable constituent. This gave model sizes of 2, 4, and 6 states for one, two, and three constituent syllables, respectively.

Using 2 states per syllable constituent showed a significant improvement in runs on the validation set as shown in Table 6.2. By doubling the number of states in each HMM, the error rate went from 58.93% to 44.78%, a reduction of almost one-quarter of the error.

With good results in increasing the number of states per constituent from one to two, I tried increasing the number of three. So, now there would be HMMs of sizes 3, 6, and 9. In the case of nuclei and of *simple* onsets and simple codas, this would be equivalent complexity to the standard phone models used above. That is, there are three states assigned for each segment in the syllable. Naturally, there are less than three states available for each segment with complex onsets/codas.

Running the three-state models on the validation gave the error rate of 34.13%, as shown in Table 6.2. Like going from one to two states, going from two to three states per syllable constituent, cut the error rate by almost one-quarter.

Continuing on, four state constituents cut the error rate from the three state constituents by a smaller 2% (as opposed to the previous 11%), giving an error of 31.94% on the validation set. It was now evident that I was reaching the optimum number of HMM states per syllable constituent because the gain by adding much less than was it was before.

To determine if I had reached the maximum number of states for optimum performance, I examined a topology of 5 states per constituent. Using 5 states per constituent results in the error rate rising from what was achieved previously with 4 states per constituent.

Syllable	Onset States	Nucleus States	Coda States	Total
NUCLEUS	0	4	0	4
ONSET–NUCLEUS	4	4	0	8
NUCLEUS–CODA	0	4	4	8
ONSET–NUCLEUS–CODA	4	4	4	12

Table 6.1: HMM Architecture for Four State Constituents

To summarise the trial with different numbers of states, the optimum number of states was reached once the number of states per constituent exceeded the maximum number of phonemes possible in that constituent. Table 6.3 on page 26 shows the number of syllables with with various lengths of onsets and codas. Table 6.2 puts the results for all the various size constituent states together. Furthermore, the number of states needed per constituent is consistent with the standard recipe that was used in constructing the phone models. That is, the phone models use three states per phone. Now, from Table 6.3, the most common constituent size is to have one phone in the constituent. So, a constituent state count of three would make sense. Indeed, the optimum number of states was almost reached

Constituent State Size	Error Rate	
	Validation Set	Test Set
1	58.93%	67.56%
2	44.78%	50.15%
3	34.13%	40.22%
4	31.94%	38.01%
5	37.36%	44.74%

Table 6.2: Error rates for different size Feature HMM constituent state sizes. By varying the number of states in an HMM, the error rate changes dramatically. The best number was four states per constituent. For nucleus only models, this would give a four state model; but full syllables with onsets and codas would have twelve state models.

# of phonemes in constituent	# of ONSETS	# of CODAS
0	146	505
1	2038	1354
2	696	919
3	67	167
4	0	2

Table 6.3: Number of phonemes in onsets and codas. For the syllables in the training data, this indicates the distribution regarding their sizes. Most syllables have codas and onsets with only one phoneme in them.

with three states per constituent. The optimum level of four makes sense when you consider that some constituents have more than one phone. For future development, a system topology where simple onsets/codas have, say, three states and complex onsets/codas have, say, five states would worth trying. So, Table 6.1 states the layout of the number of states per constituent in the optimum HMM topology.

6.3.2 Decision Tree Clustering of Constituent States

Discussion of Method

The purpose of clustering data is to have enough training observation vectors to estimate the values for each state. I need enough training data for each state so that the models are robust for recognising new speakers and new utterances. However, I do not want to cluster data so much as to have a lack of variety in the models available. For example, when clustering onsets, I could cluster the initial states of all onsets that start with a /p/. This would provide a good, general recogniser for syllables that start with /p/. However, I need a recogniser that can distinguish among the different syllables that start with /p/. Now, from Table 6.4 on page 27, I showed that with many of the syllables having few examples, there would be many that would be considered under-trained. So, those constituents that have similar states that should be tied together.

Within the nucleus, a vowel can only be clustered with an identical vowel. If I were to cluster only identical onsets (or codas), I would lose the benefit of being able to cluster similar parts of similar onsets. So, within onsets, any onset can be clustered with another onset; it is the same with codas: any coda can be clustered with another coda. So, take the onsets, k_r, p_r, and k_l. Now, if we are dealing with, say, two-state constituents, onset state #1 of k_r can be tied with onset state #1 of k_l while onset

Frequency in Training Data	Number of Syllables of this type	Percentage of Syllables
0	465	13.6%
1	924	27.1%
2-3	482	14.1%
4-6	166	4.9%
7-9	563	16.5%
10-99	731	21.4%
100-999	80	2.3%
1000+	2	0.1%
TOTAL	3413	100.0%

Table 6.4: Spread of Syllables in Training Data. SIL and dh_=ah= are the two 1000+ syllables.

state #2 of k_r can then be tied to state #2 of the different sounding onset p_r. So, this make more general tying available.

Many syllables of them occur infrequently, or not at all, in the training set. Table 6.4 gives an indication of the frequency of the syllables in the training data. It shows that a large majority of the syllables that we have data for actually occur a handful of times. That is, 76.2% of the syllables in the training list have fewer than 10 samples in the training data. Furthermore, 27.1% occur only once in the training data. So, it is not practical to build up a straightforward recogniser with many of the syllables having so little training data.

While most of the syllables have limited training data, one of the main advantages in training HMMs is that you can cluster the HMM states. It involves clustering onsets with onsets, nuclei with nuclei, and codas with codas.

Some mechanism has to be used to determine which constituents are preferred to be tied. There are different mechanisms to do this. One would be to manually go through the data, indicating which states should be tied together. Another would be to use data-driven clustering (Young et al. 1996); this would tie states which are similar to each other. Another way to cluster would be to use decision tree clustering; in decision tree clustering, questions (typically phonetic questions) are asked of all the states to determine which to tie. Since I decided to use decision tree clustering, queries were constructed. Unlike the decision tree used with phones (which was a modified version of a pre-existing tree), the queries were based strictly on the feature sets used. Taking the feature value PHONATION:**voiced**, the queries in Table 6.5 on page 28 were constructed.

Implementation of Method

HTK (Young et al. 1996) is designed for training phone or whole word models. So, since it can do either, it can also do syllable models. The problem arises, however, in that its decision-tree clustering is designed for clustering models with a similar topology. That is, take the, say, one-state constituent models. If we have the nucleus-coda syllable =ay=_l and the onset-nucleus-coda syllable p_=ay=_d, we may want to cluster the nuclei together. In the first case, state 1 is the nucleus while in the second case state 1 is the onset with state 2 being the nucleus. So, we want to tie state 1 of =ay=_l to state 2 of p_=ay=_d, as shown in Figure 6.2 on page 29. The decision tree clustering only wants to tie identically-labelled states.

So, for the decision tree clustering, I both modified HTK's decision-tree clustering algorithm and made my own scripts. The first step in the process is to break all of the syllable models up into their constituent parts. For example, if we take the syllable p_=ay=_d, there would be constituent HMMS

1. Questions for tying ONSETS.
   (a) Does a PHONATION:**voiced** phone occur anywhere in the onset?
   (b) Does a PHONATION:**voiced** phone occur as the final phone in the onset?
   (c) Does a PHONATION:**voiced** phone occur as the initial phone in the onset?
   (d) Does a PHONATION:**voiced** phone occur in the nucleus (next to the onset)?
   (e) Does a PHONATION:**voiced** phone occur with other onset phones to its left and right?
   (f) Does a PHONATION:**voiced** phone occur by itself in the coda?
2. Questions for tying NUCLEI.
   (a) Does a PHONATION:**voiced** phone occur in the onset (next to the nucleus)?
   (b) Does a PHONATION:**voiced** phone occur in the coda (next to the nucleus)?
3. Questions for tying CODAS.
   (a) Does a PHONATION:**voiced** phone occur anywhere in the coda?
   (b) Does a PHONATION:**voiced** phone occur as the final phone in the coda?
   (c) Does a PHONATION:**voiced** phone occur as the initial phone in coda?
   (d) Does a PHONATION:**voiced** phone occur an the nucleus (next to the coda)?
   (e) Does a PHONATION:**voiced** phone occur with other coda phones to its left and right?
   (f) Does a PHONATION:**voiced** phone occur by itself in the coda?

Table 6.5: Queries used for the feature-value of PHONATION:**voiced**. Each query returns a yes/no answer for the appropriate syllable constituent given to it. Similar questions also exist for each feature-value combination as well as for each phone available. A sample question related to phones is "Does a /t/ occur anywhere in the onset?"

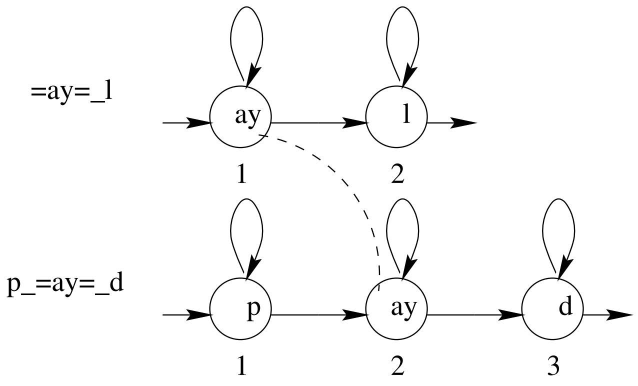

Figure 6.2: The decision tree tying provided with HTK will not allow tying across different numbered states

for the onset states from p_=ay=_d, another for the nucleus from p_=ay=_d, and a third for the coda from p_=ay=_d. Likewise, constituent HMMs are created for the nucleus from =ay=_l and for the coda from =ay=_l. Note that like constituents will not be collapsed; that is, there will be a constituent HMM for the =ay= in p_=ay=_d and a separate one for the =ay= in =ay=_l, as show in Figure 6.3 on page 30. So, in effect there are multiple cases of identical syllabic constituents, but they stand for syllabic constituents from different syllables. This is similar in thought to triphones: all triphone models recognise the same phone but from different contexts.

To actually accomplish the above, I did the following process. I created some dummy models, each of which represented an individual constituent from a specific syllable. The state(s) in these dummy HMMs are then tied to their respective states in the original HMMs. This was done as a way of copying the mean and variance vectors from the appropriate states in the original syllable models to those in the dummy HMMs. Since it is not straightforward how to specify whether the original state's values is used or the dummy's states values are used in the tied state, a better way of copying needs to be developed for future work.

With the dummy HMMS now created, all of the concerned HMMs now have the same number of states (see Figure 6.3). These HMMs are almost at the stage where I could use the original decision tree tying provided with HTK. However, the HTK algorithm is designed for triphones. I had to modify the code to allow decision tree tying to be done on syllables. The code expected regular expressions to occur in the context portions of triphones; I modified the code so that it would accept the regular expressions in syllables instead of triphones. Clustering is done for each syllable constituent and unseen syllable constituents are synthesised. Onset states will be tied to other onset states, nucleus states to other nuclei states, coda states to other coda states. Within each constituent, the tying is ordered; that is, the first state in one constituent model can only be tied with the first state in another constituent; likewise, a second state can only be tied with second state in another constituent and so on. Figure 6.3 shows how this allows the vowels from =ay=_l and p_=ay=_d to be tied. No tying can be done across constituents; for example, a /t/ in an onset can not be tied with a /t/ in a coda.

After the clustering is done, unseen constituents are synthesised. That is, my goal is to be able to recognise every syllable in TIMIT, even if it does not occur in the training data. So those syllable constituents that only occur in the validation or testing set, but not the training, will need to be synthesised using the decision tree. Then, the states in all of the constituents are tied back to their respective states in the syllable models; in effect, they are reassembled. It is highly desirable in future work to have an algorithm which will do decision tree clustering for HMM's of different sizes and across different

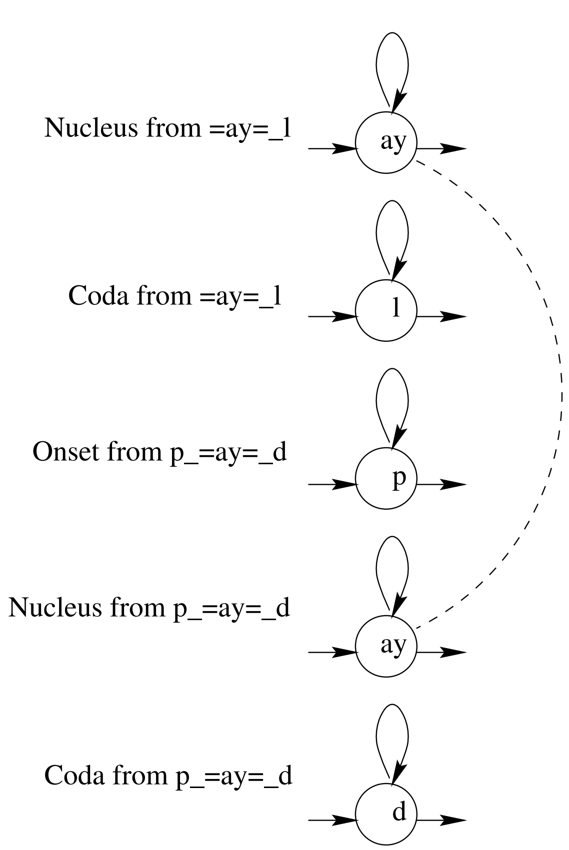

Figure 6.3: Breaking the HMMs up into their constituents makes every HMM in the decision tree clustering have the same number of states. This allows tying to be done. When using, say, two states per constituent, each of the above models would be two states long; the same with three, four, etc.

states without splitting up the syllables into constituents, but I did not have the time to fully develop this. However, this workaround described was effective.

Pool Sizes

In clustering HMM states together, I am aiming to make the states be more robust models of the speech they are aiming to recognise. This is done by pooling the data that belongs to two different states and, hence, making the states have identical pdf's. So, on one hand, I want to cluster many states together so as to give them access to more training data. However, I do not want to cluster too many states together; doing so will make it harder for the recogniser to distinguish different sounding sounds which have been clustered together. This amount of clustering is flexible. When determining which states to cluster together, I look at how much of the training data was assigned to each state in the Baum-Welch algorithm. So, some states may have only had as little as one training example to update its pdf while some may have had hundreds of examples. The states that have few training examples are the ones I am more concerned in clustering with others. Those with fewer training examples will pool their data with other, similar states so that a specified minimum number of training frames are in the pool.

To see what the optimum pool size is, I tried reclustering and retraining the models many times. That is, starting with step 4c in Table 6.7, I set the "state occupation" threshold to different amounts

State Occupations	Validation Set Error
50	96%
100	32%
200	33%
300	32%
400	32%
800	34%
2000	39%
4000	44%

Table 6.6: Experiments with the pool size of training data, using 4 states per constituent.

to see what the effect would be. I would then continue on with the training from that step to the end of the table and see the performance of the models on the validation set.

Table 6.6 gives the results for these experiments. As you can see, the syllable models are not very sensitive to the state occupation threshold enforced upon them. A threshold of, say, 100 had the same error rate as 400 even though the set with a threshold of 400 had fewer parameters to estimate. It is necessary to do this clustering, as the experiment with a threshold of 50 shows; providing a very low level of clustering produces untrained models. However, in the case of the four-state constituent HMMs here, the amount of clustering and pooling of data done is not critical as long as a minimum amount is done. So, the amount of tying does not have a big effect on recognition error.

6.4 Training Syllable HMMs

For training syllables, I used a similar method as with phones. Table 6.7 on page 32 shows how to train the models.

There were 2948 different syllables in the training set (see Table 6.4 on page 27). It is not feasible to follow the pattern used in the training of isolated models of each phone. To do the two-step process of Viterbi estimation and Baum-Welch re-estimation takes on the order of 1 hour on a Sun Ultra per model. So, even the processing were to be spread out over a network of 10 machines, it would take 300 hours to do the isolated training of all the models. Furthermore, with most of the syllables having so little training data available, the resulting models for these syllables would not be good predictors for speaker-independent recognition.

In spite of this, I still did isolated-model training. I did this using the same methodology as in the phones: using Viterbi estimation, followed by Baum-Welch re-estimation. However, I used simplified models to do it. Just like the preliminary step to creating triphone models in Chapter 5 was to create monophone models, the preliminary step in training syllable models is to train simplified syllable models. These simplified models were based both in the syllable structure and on the nucleus of each syllable. All syllables that had the same syllabic constituents were grouped together according to their nuclei. For example, a model was created called ONSET_=aa=. So, all syllables that had an /aa/ nucleus, with an onset (but no coda, in this case) were pooled together to estimate a single model for ONSET_=aa=. This model would then represent a whole syllable that has any onset, a nucleus of /aa/, and no coda. Table 6.8 gives examples of what was done. Figure 6.4 on page 33 graphically illustrates how this is done.

1. Train simplified syllable models
    (a) Do Viterbi training on each model to estimate the means, variances, and transitions for each state.
    (b) Do Baum-Welch training on each model to further re-estimate the means, variances, and transitions for each state.
2. Performed embedded training on simplified syllable models
    (a) Do five iterations of Baum-Welch embedded training on all models.
3. Create actual syllable models
    (a) Clone each simplified syllable model into all respective syllable models found in the training set.
    (b) Tie transition matrices of all models that have the same simplified syllable structure.
4. Cluster triphone models
    (a) Do one iteration of Baum-Welch embedded training on all models.
    (b) Split the syllable HMMs into one constituent model for each constituent.
    (c) Do decision tree tying of the constituent models.
    (d) Synthesise unseen constituents using the decision tree.
    (e) Paste the constituent models back into syllable models.
5. Perform embedded training of triphone models
    (a) Do five iterations of Baum-Welch embedded training on all models.

Table 6.7: Creating syllables

Model			Clustered Syllables
	=iy=		=iy=
ONSET	=iy=		b_=iy= b_r_=iy= d_r_=iy= f_l_=iy= ...
	=iy=	CODA	=iy=_ch =iy=_t =iy=_v =iy=_z ...
ONSET	=iy=	CODA	m_=iy=_t_s p_l_=iy=_z s_=iy=_m v_=iy=_l_d ...

Table 6.8: Simplified Clustering of Like Syllables for initial training for all syllables with a nucleus of =iy=

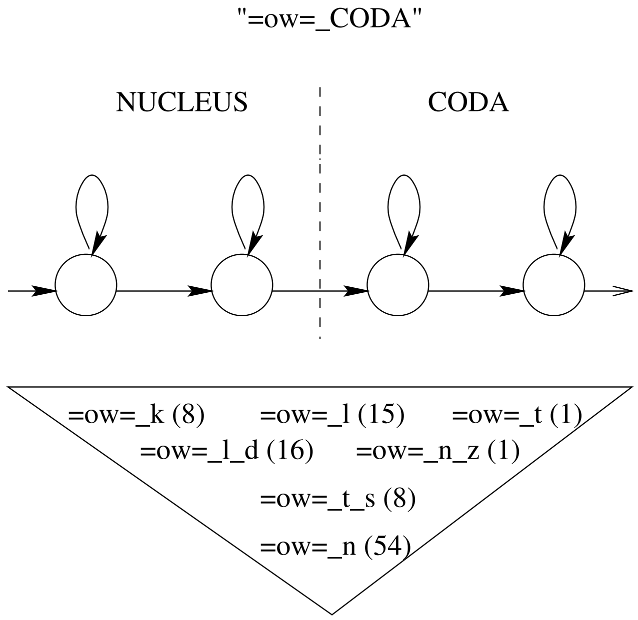

Figure 6.4: A simplified HMM and training data for =ow=_CODA. Next to each syllable is the number of times that syllable occurs in the training data. All occurrences of all of these syllables will be used in training the simplified model.

Features	MFCC
38%	37%

Table 6.9: Error rates for features vs. MFCCs on the full test set.

6.5 Evaluation

The experiments above describe how I arrived at the chosen parameters to use in the final feature syllable system. The parameters that for the optimal feature syllable model were four states per constituent and 300 training frames per state.

Once I had found the best topology and training method for constructing feature syllable models, I took that and applied it to train syllable models using MFCCs as their observation vectors. This may not be the best way to find an optimum accuracy for MFCC based syllables. However, I am assuming that the method that proved well for feature streams will provide a competitive set of MFCC syllable models.

As can be seen from Table 6.9, MFCCs and features provided nearly identical syllable recognition results. This speaks well for the feature models, even though they lag slightly behind the MFCCs. MFCCs are well established as a good parametrisation of speech. This makes it impressive that the less developed feature stream parametrisation should give a newly identical result, just like in the

Feature Syllables
51.9%

Table 6.10: Error rate for Kirchhoff (1996) on 90 minutes of German speech taken from Kohler et al. (1994)

Sentence type	Syllable Error Rate	Sentences Correct
phonetically-diverse (SI)	53%	3%
phonetically-compact (SX)	28%	21%
ALL	38%	14%

Table 6.11: Error rates on Feature Syllables in the test set. This compares the rates for the SI sentences verses the SX sentences.

	Num New Syllables	Num Total Syllables	Per
SI	89	8288	1%
SX	929	12003	8%

Table 6.12: Occurrences of New Syllables in parts of test set. The SX sentences, which have better recognition results, have more unseen syllables (not seen in the training data).

phone models.

The feature syllables had vastly different results on the SX versus the SI sentences (Table 6.11). The error rate on the SI sentences were almost twice that of the the SX sentences. Considering that the SI sentences are real sentences and not the artificial ones created for the SX, this implies that the system would do poorly for recognition of real speech.

To examine the reason why the SI sentences produced different results from the SX sentences, I tried looking at the number of new, synthesised syllables in each part of the test set. Table 6.12 gives the breakdown for this. This shows that it was not because of dealing with unseen syllables that SI utterances did poorly.

Having seen that the SX test sentences did well even with a greater portion of unseen syllables, I decided to see how the test set did in recognising unseen syllables. Table 6.13 shows that syllables with few actual training examples did well in the test recognition. The correctness rating on the syllable level does not increase with those syllables with an abundance of training data. This suggests that the clustering was very effective in training robust models for syllables with little or no training data.

Frequency in Training Data	% Correct in sub-test set
0	64%
1	59%
2-3	55%
4-6	69%
7-9	60%
10-99	59%
100-999	55%
1000+	56%
OVERALL	58%

Table 6.13: Correctness in full test set, based frequency in training data. These values are computed from the transcriptions in the recognition output. These transcriptions are lined up against the original labelling. The correctness above is computed by taking every syllable in the original labelling and determining how many are correct in the aligned transcription. Note: There were many test sentences that were recognised correctly in their entirety. The figures do not take these correct sentences into account because of the figures from the correct sentences not being readily available; hence the "sub-test" used is all test utterances with at least one error.

Chapter 7

Phones vs. Syllables

	MFCC PHONES	FEATURE SYLLABLES
Syllable Error	14%	26%
Sentences Correct	63%	41%

Table 7.1: Recognition rates on CORE TEST set for word models

The trained MFCC phone models and feature syllable models from Chapters 5 and 6, respectively, were used to do whole word recognition. The MFCC triphone system has been well developed and experimented with; so, it should have a good word recognition rate. The results from the phone recognition and syllable recognition experiments in the earlier chapters indicate that featured syllables should also do well in word recognition. As shown in Table 7.1, the MFCC triphone system does better than the featured syllable system on the core test set. The error rate for the feature syllables is almost twice that of the MFCC phones. Nevertheless, the results are good when considering that the work with feature syllables is still in the early stages.

Chapter 8

Conclusion

8.1 Project Summary

This thesis has investigated a new method for doing speech recognition, that of using phonetic features. Throughout the thesis I have compared my new work to other experimental work being done and to established state-of-the-art systems. This involved building a standard MFCC triphone system to serve as a baseline for building the new feature syllable system.

In order to do recognition of phonetically featured syllables, a feature detector is vital. In automatic speech recognition, all processing starting from the input speech signal to the recognised speech must be done automatically by the computer. Artificial neural networks provided an accurate way of classifying the speech phonetically (Chapter 3). They are also able to see the relations between related feature values.

Having this feature detector enabled the building of a triphone recognition system (Chapter 5). This system was built using a standard method. To measure the performance of the feature triphone system, its results were compared to an identically trained triphone system that used MFCCs as their observation vectors. By having nearly identical recognition results as the already proven MFCCs, the features showed themselves to be efficient descriptors of speech.

Having demonstrated their good performance in phone recognition the features then did well in syllable recognition (Chapter 6). The performance of the syllable models is sensitive to their the topology of the HMM. Decision tree clustering proved to be effective in training the syllables as syllables with no training data did just as well as those with plenty of training data.

The ultimate task in constructing sub-word models is to recognise whole words. So, for an overall test of phonetically featured syllable recognition, word recognition was done using their models. This was compared against the baseline MFCC triphone models. This showed that the feature syllables do not yet perform as well as the state-of-the-art models. However, the MFCC triphone models have been extensively studied and developed. So, a recognition result close to the state-of-the-art shows the great potential for feature syllables models. As feature syllable models are investigated more, they could outperform the performance of the triphone systems.

8.2 Future Work

There are many avenues for investigating this area more. One are currently being investigated is the use of different feature systems to describe speech. This thesis has only looked at a specific multi-valued feature system, but King et al. (1998) is also looking into binary valued feature systems. Additionally, a better phonetic labelling, as described in section 2.2, needs to be implemented.

The ANNs used in this work may not be at their optimum. That is, their architecture was based on previous work done in Stephenson (1998). The ANNs had an architecture that was proven in related work, and therefore provided feature detection that was suitable for this initial investigation into modelling syllables using features. However, as this work progresses in the future, additional ANN architectures should be examined.

My experiments with the HMM topology dealt with the number of states in each constituent. They assumed an equal number of states for each constituent. Future work would investigate whether different constituents should have a different number of states from each other. Onsets and codas may need more states than nuclei; complex onsets and codas may need more states than their simple counterparts. I recommend looking into whether other topologies should be implemented.

The decision tree clustering can be examined more. Within onsets and within codas, I did not put any restrictions on which onsets and codas could be tied with other onsets and codas, respectively. Perhaps only identical onsets and codas should be tied together.

There are other parameters that can be added to the HMM state, which I did not use. I did not use streams or mixtures. Streams may add to the performance of the models, as each feature can be treated as a different stream and weighted according to their performance. While I did not use delta and acceleration coefficients in the feature models, they may add useful information to the models.

The clustering and tying system for the syllable models involved a lot of computing time. If a more efficient algorithm can be incorporated into HTK, this would save on the training time.

Acknowledgements

I would like to thank my supervisors Stephen Isard and Simon King for teaching me what they know about speech recognition and for guiding my work in this thesis.

Bibliography

Browman, C. and Goldstein, L. (1990). Gestural specification using dynamically-defined articulatory structures. *J. Phon.*, pages 299–320.

Catford, J. C. (1988). *A Practical introduction to phonetics*. Oxford University Press, Oxford.

Deng, L. and Sameti, H. (1996). Transitional speech units and their representation by regressive markov states: Applications to speech recognition. *IEEE Transactions on Speech and Audio Processing*, 4(4):301–306.

Deng, L. and Sun, D. X. (1994). A statistical approach to automatic speech recognition using the atomic speech units constructed from overlapping articulatory features. *Journal of the Acoustical Society of America*, pages 2702–2719.

Garofolo, J. S. (1988). *Getting started with the DARPA TIMIT CD-ROM: An acoustic phonetic continuous speech database*. National Institute of Standards and Technology (NIST), Gaithersburgh, MD.

Hyman, L. M. (1975). *Phonology: theory and analysis*. Holt, Rinehart and Winston, New York.

King, S., Stephenson, T., Isard, S., Taylor, P., and Strachan, A. (1998). Speech recognition via phonetically features syllables. In *Proceedings of the International Conference on Spoken Language Processing*.

Kirchhoff, K. (1996). Syllable-level desynchronisation of phonetic features for speech recognition. In *International Conference on Spoken Language Processing*, volume 4.

Kohler, K., Lex, G., Pätzold, M., Simpson, A., and Thon, W. (1994). *Handbuch zur Datenaufnahme und Transliteration in TP14 von BERBMOBIL-3.0*. IPDS, Kiel. VM Technisches Dokument Nr. 11.

Rabiner, L. and Juang, B.-H. (1993). *Fundamentals of Speech Recognition*. Prentice Hall PTR.

Rojas, R. (1991). *Neural networks: a systematic introduction*. Springer-Verlag, Berlin.

Stephenson, T. A. (1998). Artificial neural networks in recognition of phonetic features in speech. M.Sc. Project.

Ström, N. (1997a). Phoneme probability estimation with dynamic sparsely connected artificial neural networks. *The Free Speech Journal*.

Ström, N. (1997b). Sparse connection and pruning in large dynamic artificial neural networks. In *Proceedings of Eurospeech*.

Young, S., Jansen, J., Odell, J., Ollason, D., and Woodland, P. (1996). *HTK manual*. Entropic.

Centre for Cognitive Science Publications

Other Cognitive Science Masters Theses, Research Papers and Working Paper collections are available electronically at

`http://www.cogsci.ed.ac.uk/ccs/publications/`

For paper copies of the Centre's internal publications, please contact:

The Librarian
Centre for Cognitive Science
The University of Edinburgh
2 Buccleuch Place
Edinburgh EH8 9LW
UK
Email: `librarian@cogsci.ed.ac.uk`
Fax: +44 131 650 4587

Appendix A

Feature-Value Mapping

	centrality	continuant	frontback	manner	phonation	place	roundness	tense
aa	central	cont	back	vowel	voiced	low	unrou	tense
ae	full	cont	front	vowel	voiced	low	unrou	lax
ah	central	cont	front	vowel	voiced	low	unrou	lax
ao	full	cont	back	vowel	voiced	mid	rou	tense
aw	full	cont	back	vowel	voiced	low_mid	unrou_rou	tense
ay	full	cont	nil_front	vowel	voiced	mid_high	unrou	tense
b	nil	noncont	front	occlusive	voiced	labial	unrou	non-tense
ch	nil	noncont	front	fricative	unvoiced	palatal	unrou	non-tense
d	nil	noncont	front	occlusive	voiced	coronal	unrou	non-tense
dh	nil	noncont	front	fricative	voiced	coron-dent	unrou	non-tense
eh	full	noncont	front	vowel	voiced	mid	unrou	lax
er	full	noncont	back	approx	voiced	velar	unrou	non-tense
ey	full	cont	front	vowel	voiced	mid_high	unrou	tense
f	nil	cont	front	fricative	unvoiced	lab-dent	unrou	non-tense
g	nil	noncont	back	occlusive	voiced	velar	unrou	non-tense
hh	nil	cont	back	fricative	unvoiced	glottal	unrou	non-tense
ih	full	cont	front	vowel	voiced	high	unrou	lax
iy	full	cont	front_nil	vowel	voiced	high_mid	unrou	tense
jh	nil	noncont	front	fricative	voiced	palatal	unrou	non-tense
k	nil	noncont	back	occlusive	unvoiced	velar	unrou	non-tense
l	nil	noncont	front	approx	voiced	coronal	unrou	non-tense
m	nil	noncont	front	nasal	voiced	labial	unrou	non-tense
n	nil	noncont	front	nasal	voiced	coronal	unrou	non-tense
ng	nil	noncont	back	nasal	voiced	velar	unrou	non-tense
ow	full	cont	nil_back	vowel	voiced	high	unrou_rou	tense
oy	full	cont	back_front	vowel	voiced	high	rou_unrou	tense
p	nil	noncont	front	occlusive	unvoiced	labial	unrou	non-tense
r	nil	noncont	back	approx	voiced	coronal	unrou	non-tense
s	nil	cont	front	fricative	unvoiced	coronal	unrou	non-tense
sh	nil	cont	front	fricative	unvoiced	palatal	unrou	non-tense
t	nil	noncont	front	occlusive	unvoiced	coronal	unrou	non-tense
th	nil	noncont	front	fricative	unvoiced	coron-dent	unrou	non-tense
uh	full	cont	back	vowel	voiced	high	rou	lax
uw	full	cont	back	vowel	voiced	high	rou	tense
v	nil	noncont	front	fricative	voiced	lab-dent	unrou	non-tense
w	nil	noncont	front	approx	voiced	labial	rou	non-tense
y	nil	noncont	back	approx	voiced	velar	unrou	non-tense
z	nil	cont	front	fricative	voiced	coronal	unrou	non-tense
zh	nil	cont	front	fricative	voiced	palatal	unrou	non-tense
sil	0	0	0	0	0	0	0	non-tense

Appendix B

Feature Stream Example

The following plots are of the outputs from the full set of artificial neural nets. They are done on the TIMIT test sentence of dr2/fjwb0/sx5. The overall correctness numbers for the the nets on this specific utterance are given in Table B.1

Net	% Correct
centrality	94.5%
continuant	92.4%
frontback	82.1%
manner	86.2%
phonation	93.1%
place	69.0%
roundness	100.0%
tense	79.3%
Overall	51.7%

Table B.1: Correctness for dr2/fjwb0/sx5

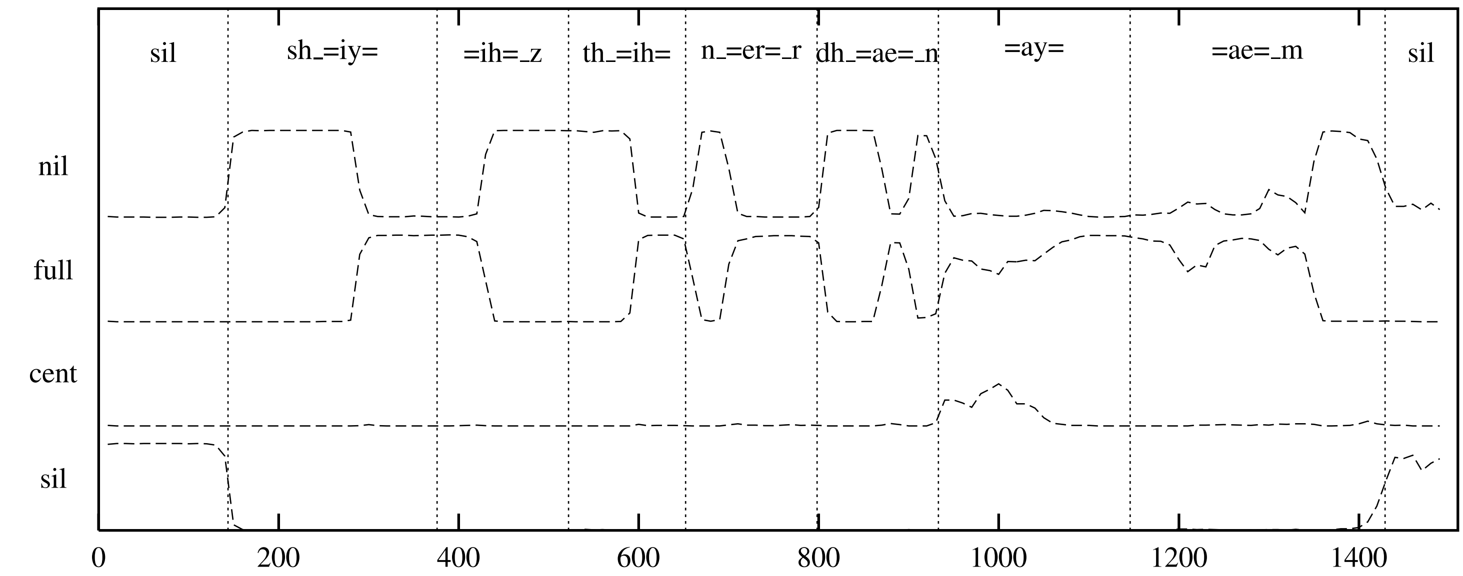

Figure B.1: Centrality

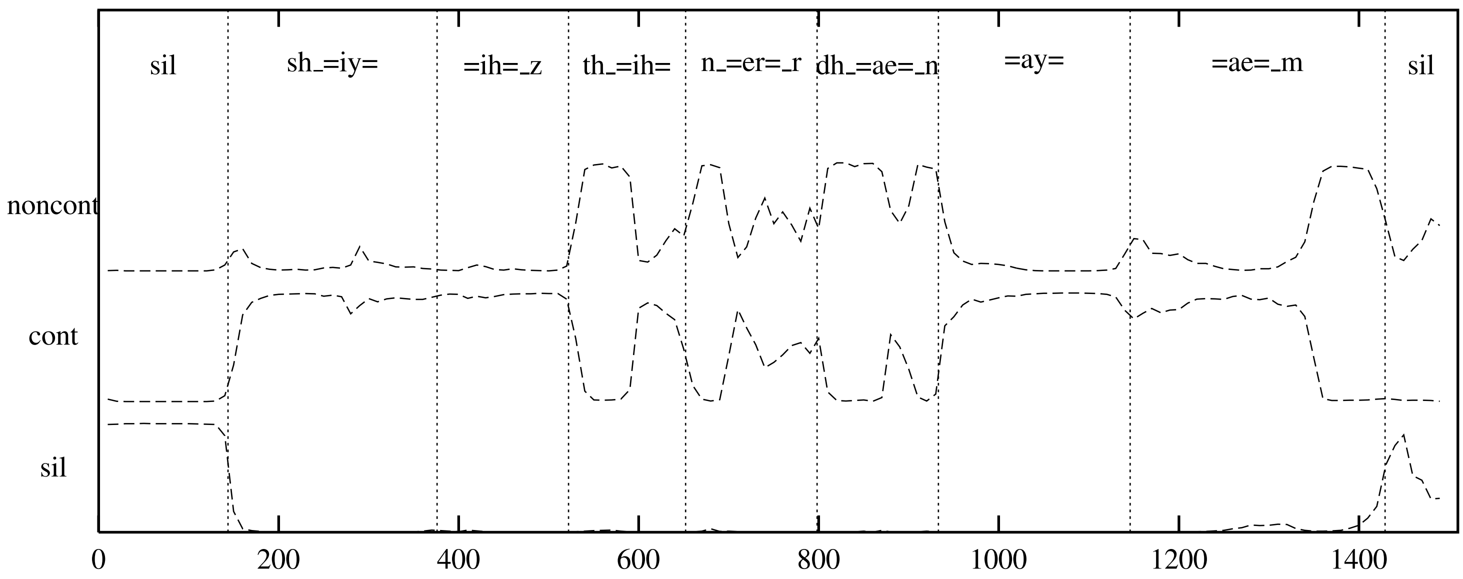

Figure B.2: Continuant

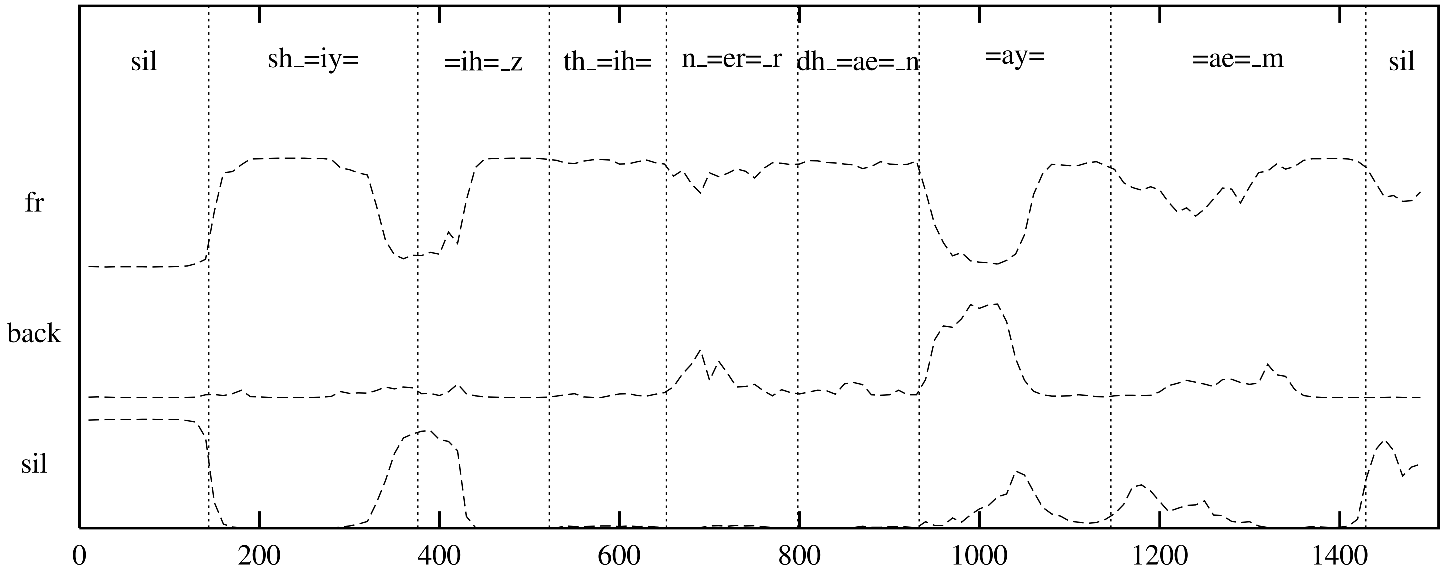

Figure B.3: Frontback

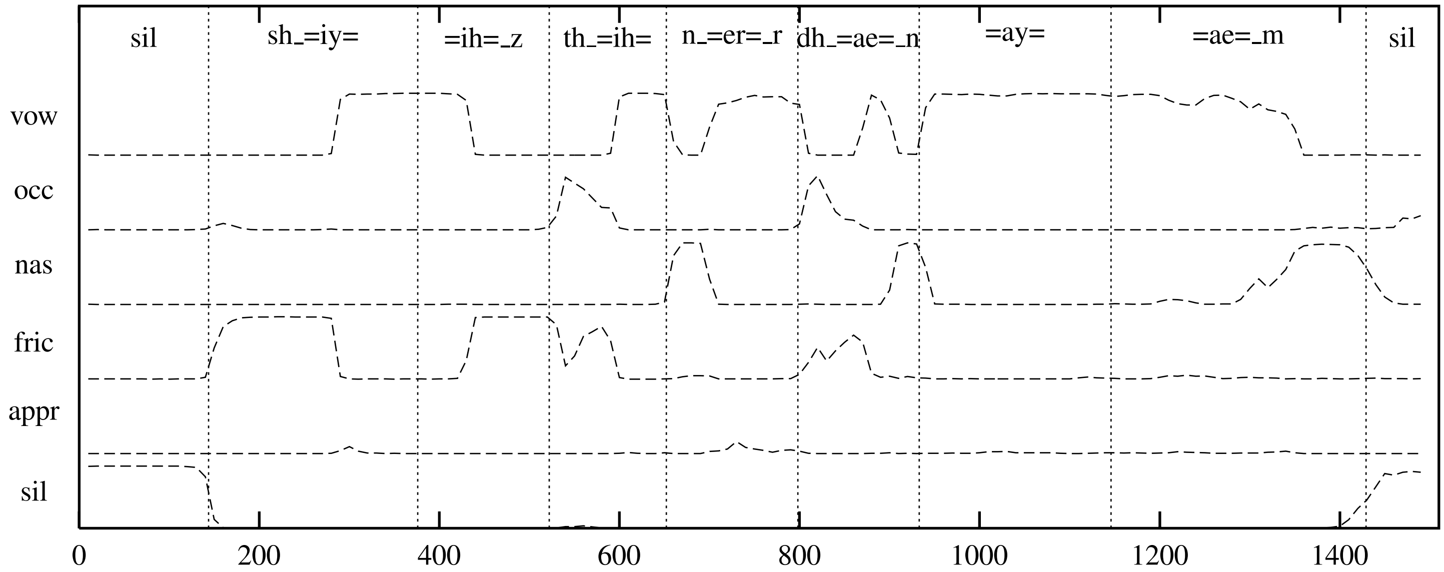

Figure B.4: Manner

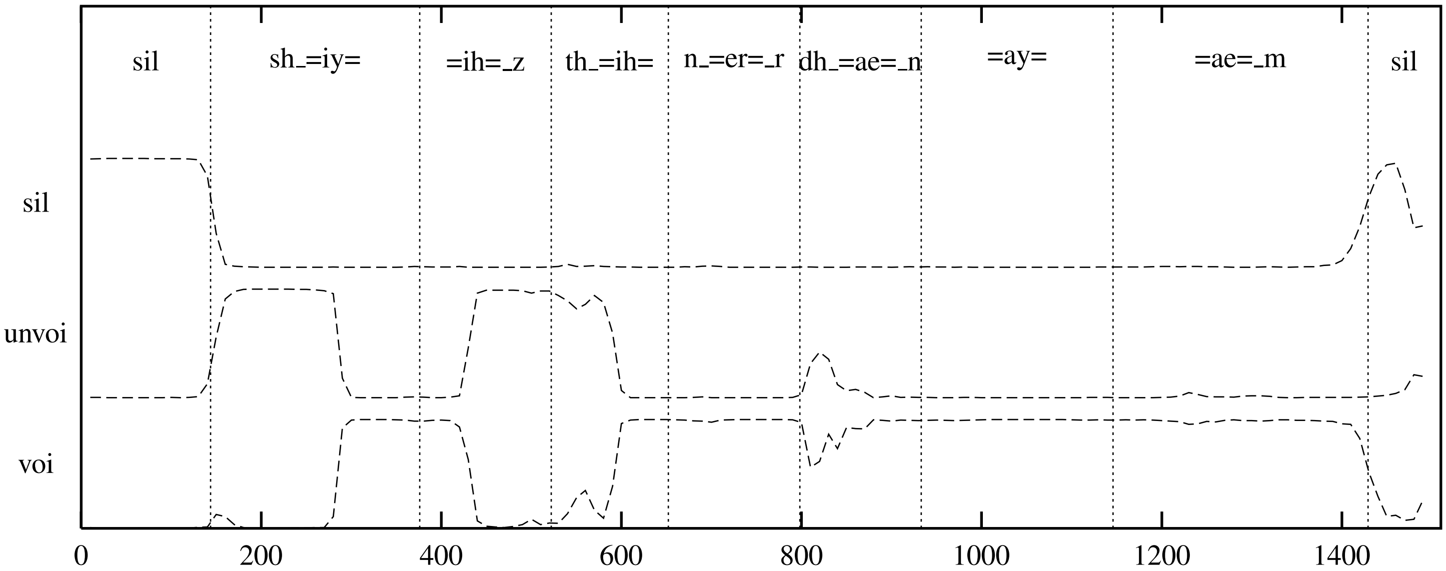

Figure B.5: Phonation

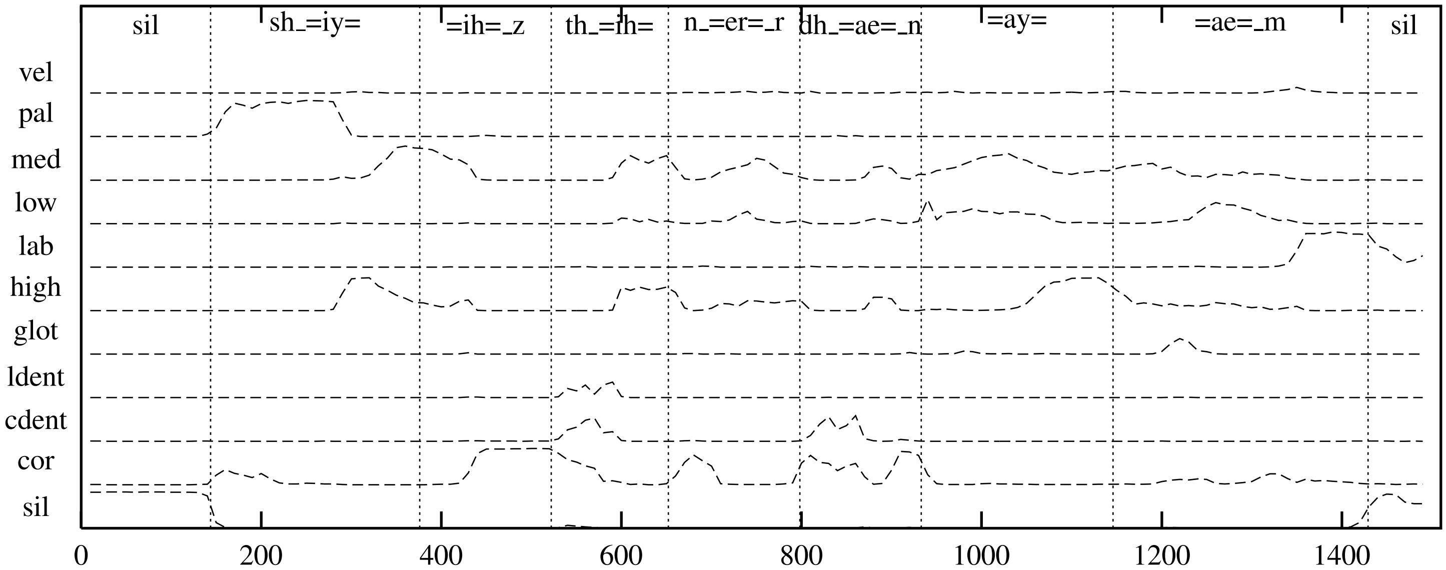

Figure B.6: Place

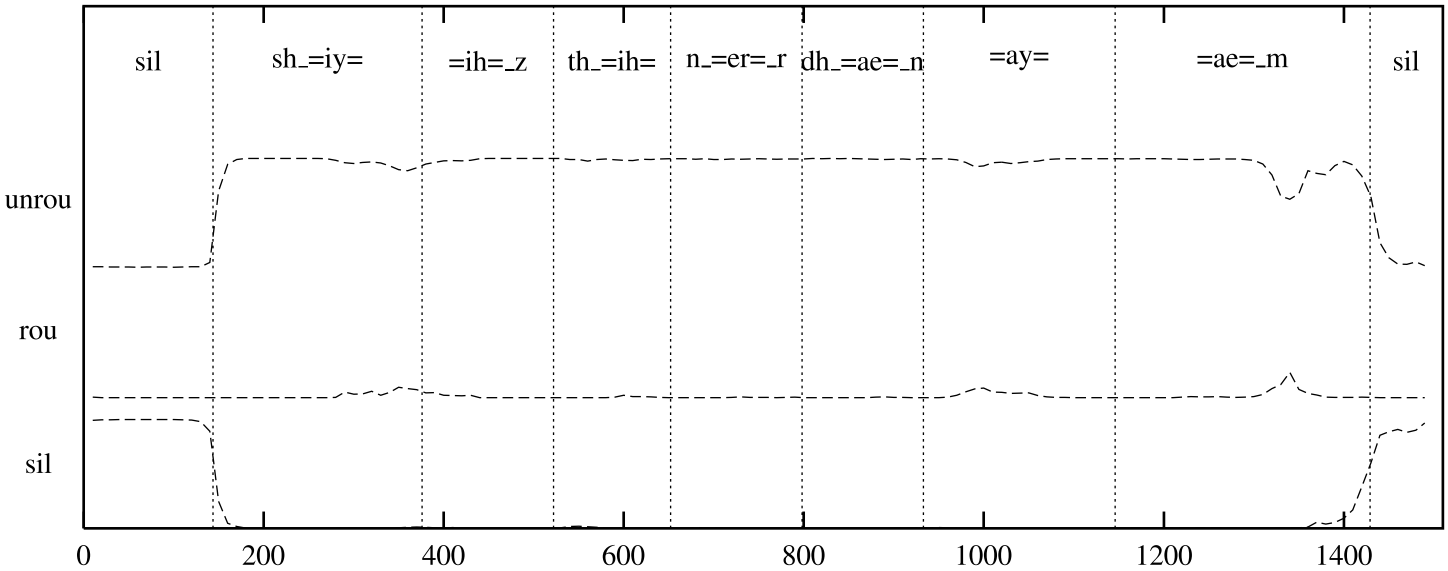

Figure B.7: Roundness

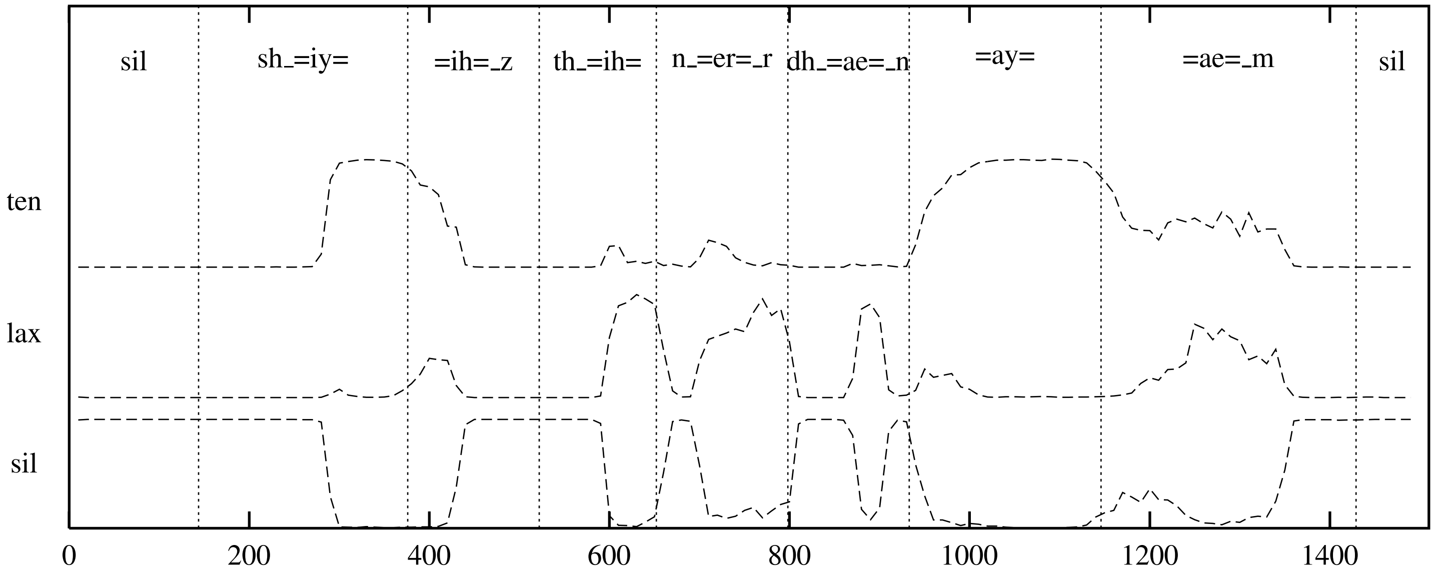

Figure B.8: Tense

Appendix C

ANN Confusion Matrices

	s i l	c e n t	f u l l	n i l
sil	87.4	0.4	2.6	9.5
cent	0.9	34.3	53.3	11.5
full	0.4	3.4	87.7	8.4
nil	1.7	0.7	7.5	90.1

Table C.1: Confusion matrix for CENTRALITY

	sil	cont	noncont
sil	87.4	4.5	8.1
cont	0.8	89.2	10.0
noncont	1.6	16.3	82.1

Table C.2: Confusion matrix for CONTINUANT

	sil	back	fr
sil	75.1	8.6	16.4
back	3.0	71.7	25.3
fr	2.2	6.2	91.6

Table C.3: Confusion matrix for FRONTBACK

	sil	appr	fric	nas	occ	vow
sil	89.2	1.4	2.0	1.0	3.2	3.1
appr	0.8	70.8	1.7	1.1	1.2	24.3
fric	2.1	1.1	87.7	0.8	5.0	3.3
nas	2.1	3.1	2.3	79.9	3.3	9.2
occ	3.0	1.0	5.0	1.8	87.0	2.3
vow	0.5	5.2	1.2	0.8	0.9	91.4

Table C.4: Confusion matrix for MANNER

	voi	unvoi	sil
voi	95.0	4.2	0.8
unvoi	9.4	88.2	2.4
sil	7.8	4.3	87.9

Table C.5: Confusion matrix for PHONATION

	sil	cor	cdent	ldent	glot	high	lab	low	mid	pal	vel
sil	89.4	4.0	0.3	0.4	0.3	1.0	1.3	0.7	1.0	0.1	1.5
cor	1.9	80.4	0.7	0.7	0.1	4.0	2.8	1.6	2.6	1.1	4.2
cdent	4.9	29.7	43.0	8.0	0.2	5.2	3.1	1.3	1.3	0.1	3.2
ldent	2.7	10.3	2.6	73.3	0.6	2.0	4.1	0.9	0.7	0.5	2.1
glot	11.7	9.4	0.5	2.2	58.0	4.8	2.0	2.5	2.8	0.8	5.1
high	0.5	6.7	0.3	0.2	0.1	70.9	1.1	5.9	11.3	0.2	2.8
lab	2.8	13.3	0.6	1.1	0.3	2.3	74.2	0.9	1.8	0.1	2.7
low	0.8	6.5	0.2	0.4	0.2	12.4	1.3	51.3	24.6	0.1	2.2
mid	0.5	5.6	0.1	0.3	0.1	18.3	1.4	15.7	55.9	0.1	2.1
pal	0.9	18.0	0.0	0.7	0.1	1.1	0.1	0.1	0.2	77.9	0.7
vel	1.8	13.0	0.3	0.5	0.4	7.4	1.8	1.5	2.8	0.2	70.4

Table C.6: Confusion matrix for PLACE

	sil	rou	unrou
sil	86.2	0.6	13.2
rou	0.5	47.4	52.1
unrou	1.1	2.0	96.8

Table C.7: Confusion matrix for ROUNDNESS

	sil	lax	ten
sil	94.2	2.2	3.6
lax	14.1	69.2	16.7
ten	11.2	10.1	78.6

Table C.8: Confusion matrix for TENSE

Appendix D

Related Papers

This thesis is part of a larger research project being done at the University of Edinburgh. Related papers by myself and others can be found under the following URL on the World Wide Web: `http://www.cstr.ed.ac.uk/projects/espresso/`.