TR-IT-0285

SDB: Segment selection

Dave Sann

November 1998

ABSTRACT
This technical report describes the SDB and selection of segments for concatenative speech synthesis. It includes descriptions of the use of *sdb_batch* (segment selection), instructions on database specification and construction, cost function implementation and parameter specification. There are also a number of comments on existing work and some suggestions for future progress.

Contents

Chapter 1

Introduction

This technical report describes the SDB. The SDB (Segment DataBase) is a database of speech segments. It can be used as the basis of segment selection for concatenative speech synthesis.

The SDB is intended as a replacement for the previous segment database within CHATR, the UDB. It was implemented to resolve various problems. Firstly the code itself was/is extremely difficult to follow and the programming style, which uses many global variables in both C code and the runtime 'lisp environment', leads to confusion and therefore many potential bugs. In addition to this it was seen as being important to separate the segment selection section of CHATR cleanly from the rest of the process. Code for prediction (before) and segment concatenation (after) had become confused and the distinction between the processes became unclear. This was especially true for the data which existed throughout the system but actually belonged to specific separate processes.

Chapter 2

Search and Selection

2.1 The Search for Segments

The theory of searching for segments is very simple.

Given each of our target segments, a number (which may be all the potential segments found in the database) of candidate segments is selected, according to their target cost. The target cost is specified by some target function, which determines how well the candidate segments matches the ideal, target segment. If the search limits the number of possible candidates, then the target cost should include some estimate of the cost that joining the given segment might incur.

Once candidates have been selected for all the targets, a simple search is conducted that identifies the minimum cost path through the segments. This search consists of propagating a number of paths through the matrix of candidates, segment by segment. The lowest scoring (least cost) path that emerges at the end of the search is the best path.

This path is defined by the target cost already discussed, and the joint cost. The joint cost is a measure of the cost that will be incurred when joining two signal segments together. This joint cost attempts to have some correlation with perceptual reality. That is a high cost should reflect a noisy (poor quality) joint and vice versa.

The complexity of the search is determined by the number of candidates we allow for each segment and the number of paths that we allow to be propagated at each stage in the search. For best results these should be unlimited. For best speed, they should be as small as possible.

2.2 Suitability functions

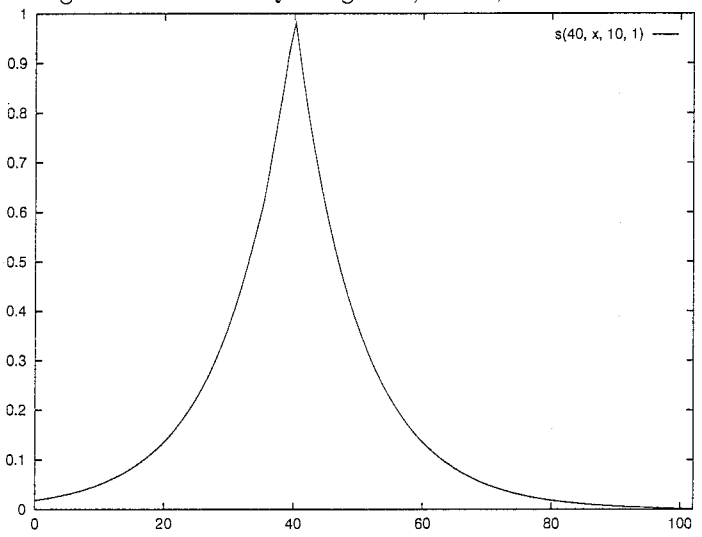
Suitability functions are extensively employed in the target and joint functions which follow. a suitability function is defined by the following formula.

$$S = e^{-(\frac{|X-x|}{T})^R+b} + min$$

where, $b = log(max - min)$ $\quad 0 \leq min < max \leq 1$

X : target value of some variable

Figure 2.1: suitability: target 40, rate 1, tolerance 10



x : actual value

T : tolerance

R : rate

min : minimum value

max : maximum value

The following figures show suitability functions for various values of T, R, min and max.

2.3 Target Functions

The target functions currently implemented for the selection use the following criteria.

- f0; at start middle and end of segment.
- power; at start, middle and end of segment.
- duration
- phonetic context match

MTarget Function

Multi target function that allows you to specify target costs for each of several classes of segment (phoneme). The classes are as follows,

- vowel

Figure 2.2: suitability: target 40, rate 2, tolerance 10

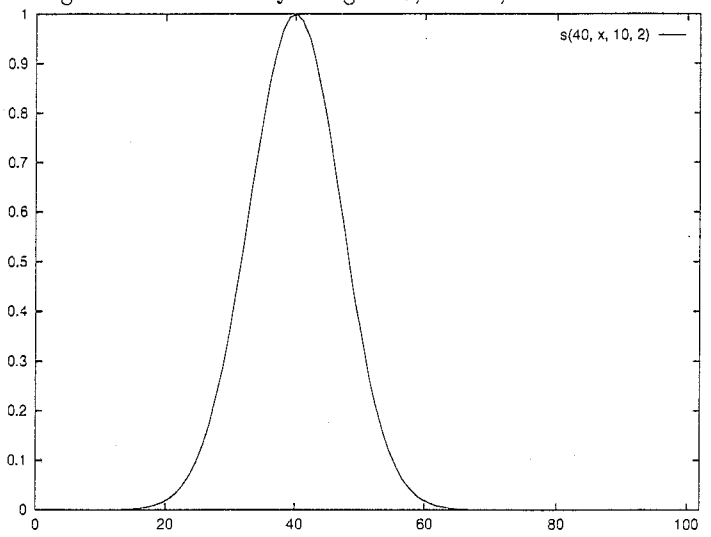


Figure 2.3: suitability: target 40, rate 10, tolerance 10

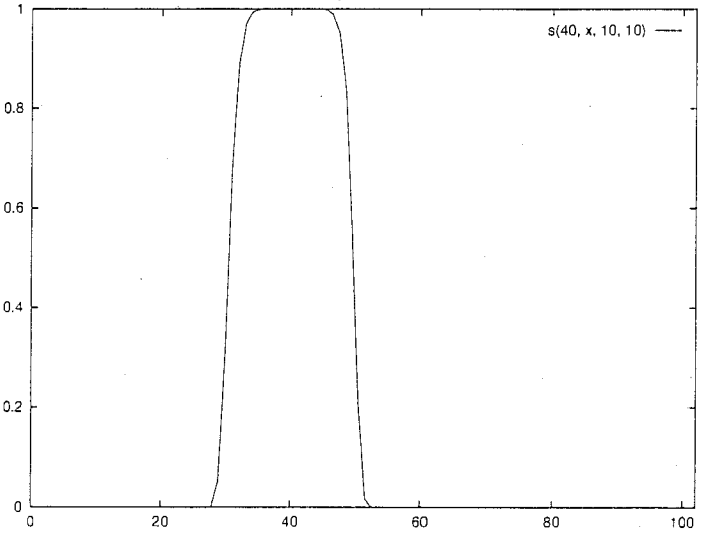
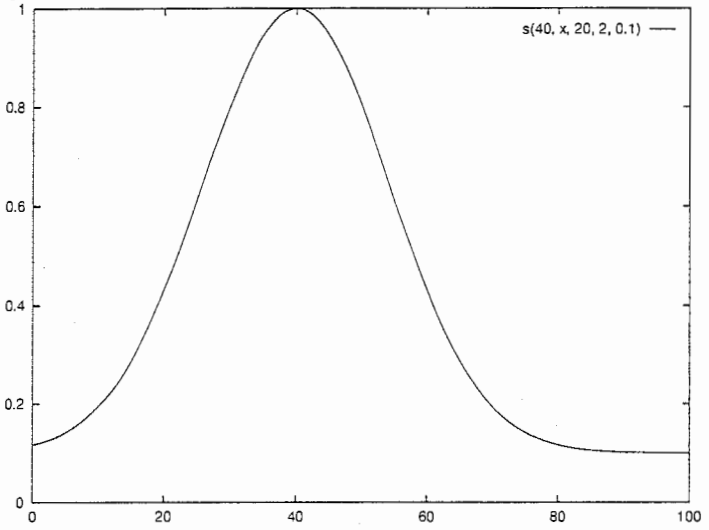


Figure 2.4: suitability: target 40, rate 2, tolerance 20, min 0.1



- liquid
- nasal
- fricative
- affricate
- plosive
- silence

Classes that do not have values specified take on a default value.

2.4 Joint Functions

Joint functions implemented currently use the following criteria,

- f0 jump at joint
- delta f0 mismatch
- power jump at joint
- delta power mismatch
- spectral jump at joint (including delta and acceleration coefficients if desired)

MJoint Function

Joint functions that allow specification of joint costs specifically for joints between different classes of segment. The classes are the same as in the MTarget function. So some example joints under consideration are,

- vowel → liquid
- fricative → affricate
- ...

Classes that do not have values specified take on the default value.

2.5 HMM Based Functions

There are two HMM based functions that have been initially explored but require further testing. One is employed as an estimate of the cost of concatenating segments when their phonetic context does not match exactly. The second is used as a normalizing correction when measuring the error between two spectral frames of segments being examined for concatenation. In both of these methods it is necessary to take into account unseen contexts (Contexts which never occur in the training data). This is done using HTK [htk] clustering.

HMM Target Cost

The target cost is calculated from the difference in states of context dependent tri- or bi- phone HMMs. Biphones are used in the case of small training sets. The figure 2.5 Illustrates how this distance is calculated. It is first necessary to train the HMMs. In the biphone case, only one state is interesting to us. This state is that affected directly by the context. In the case of right biphones it is the last state and in the case of left biphones it is the first state [1]. If triphones are used then both the first and last states are of interest. 'Uninteresting' states, that is those which are only minorly affected by the context, are tied to better estimate their parameters.

The distance `X+Y`, `X+Z` is given by the kullbark leibler distance [kullbark] between the dependent states (the last state only in this case). This distance gives a measure of the similarity of the distributions of the two states and therefore a measure of the similarity of the frames produced by the two states.

If biphones are used then both left and right biphones must be trained in order to account for both of the contexts of the phoneme.

HMM Joint Cost

The HMM joint distance is really a normalization. Rather than simply take the root mean square error between two spectral frames at a joint boundary as is done in a reduced fashion in CHATR versions up to 0.98. The error between each element of the spectral frames is normalized according to the variance in the state of the corresponding HMM context model. So for an X+Y joint, the two

[1] This is true for simple left to right non-skip HMMs. For more complex HMM architectures, this may not be the case.

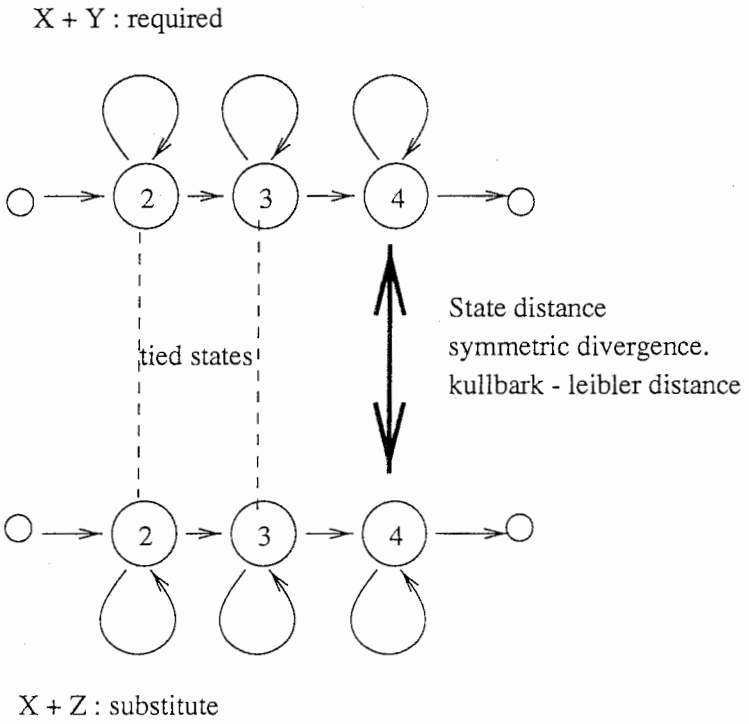


Figure 2.5: HMM based phonetic context distance

models X+Y and X-Y could be used (last state and first state respectively). The result of this normalization is that if particular elements of the spectra which represent this state have a large variance, the effect of their error on the overall error is reduced. We want to do this because we expect that largely varying elements will have large errors and therefore will always be the most significant feature in an unnormalized model. There is an underlying assumption when we conduct this kind of normalization that the large changes in widely varying elements are less perceptually important. This assumption is not necessarily correct.

2.6 Adding New Cost Functions to the Search

The addition of new target and joint functions or modification of old functions is very simple. It is simply necessary to subclass (C++) the base target or joint function class or extend an existing cost function class. The joint functions have both access to the segments being dealt with and the DBFile from which they come so functions can take account of context and also employ any information which is held in the DBFile itself.

2.7 Parameter Specification for Target and Joint Functions

The key point to understand in parameter specification is how to specify suitability functions, in terms of,

- tolerance
- rate
- min and max

the suitabilities can be specified as follows,

- tolerance only, rate defaults to two (Gaussian)

  ```
  { tolerance }
  ```

  Example, tolerance is 10 at all points. rate is 2.

  ```
  { 10 }
  ```

- tolerance and rate of convergence

  ```
  { tolerance rate }
  ```

  Example, tolerance is 10 at all points. rate is 1.

  ```
  { 10  1 }
  ```

- tolerance at zero, a gradient and the rate of convergence. The function $y = Mx + C$ is used to calculate the tolerance at a particular point. C is the tolerance when the target is zero. M is the gradient. x is the value of the target. Using this feature the tolerance can be changed depending upon the target value.

  ```
  { tolerance_zero tolerance_gradient rate }
  ```

  Example, tolerance at 0 is 10, gradient 1 and the rate is 1. So when the target is 10 the tolerance is 20. when the target is 50 the tolerance is 60 ...etc.

  ```
  { 10 1 1 }
  ```

- specify the change in tolerance as a point and a gradient.

  ```
  { tolerance_x tolerance_y tolerance_gradient rate }
  ```

  Example, same suitability function as above,

  ```
  { 100 110 1 1 }
  ```

- specify the tolerance value in terms of two points.

  ```
  { tolerance_x1 tolerance_y1 tolerance_x2 tolerance_y2 rate }
  ```

  Example, same suitability function as above,

  ```
  { 0 10 100 110 1 }
  ```

All of the above suitability specifications may have an additional minimum and maximum set to limit the range of the suitability function. For example,
A suitability function as above, but limited to the range 0.1 to 0.95. That is never 100 percent correct but never so bad that it will dominated over everything else.

```
{ { 0.1 0.95 }  0 10 100 110 1 }
```

The following grammar specifies the parameters and how they are to be expressed in order to manipulate the cost functions and the search.

```
// TOP LEVEL (ENTRY POINT)
//  all caps are non terminal symbols.
//  all small letters are terminal symbols.
//  | means 'or'
// '//' is a comment
GUSPARAMS: CANDMAX    |            // number of cands in search
           beam       { BEAM           } | // the search beam
           target_fn  { TARGET_FN      } | // the target function
```

```
            joint_fn  { JOINT_FN       } | // the joint function
            sjoint_fn { START_JOINT_FN } | // the terminal start
                                  // function (joint from beginning)
            ejoint_fn { END_JOINT_FN }     // the terminal end fn (joint
                                  // to end)


// use this to specify the number of candidates allowed for each
// requested segment. 0 means unlimited
CANDMAX : candmax INTEGER;

// use this to specify the type of beam to use (currently no choice)
BEAM    : fsbeam {FSBEAM}

// use this to specify the width of the search beam that is the number
// of paths propagated at each step of the search. 0 is unlimited
FSBEAM : width INTEGER;

// this specifies the target function for scoring target candidates
TARGET_FN : target_fn{ MTARGET_FN }

// MTarget function individually specifiable targets for classes of
// phonemes
MTARGET_FN : mtarget_fn {
  // phoneme distance function definition
  phonemeDist { PHONEME_DIST }
  // target suitability functions
  suitabilities{ MTARGET_DEFAULT MTARGET_TARGET }

}

// this is used to choose the phoneme distance function hand or hmm
PHONEME_DIST : PHDIST_HAND | PHDIST_HMM

// hand coded phoneme distance function configuration. Set the values
// to control the distances of different phoneme contexts used on the
// left and the right of the target segment.

// target segment stream : ... tprev targ tnext ...
// candidate segment     : ... cprev cand cnext ...
// tprev is compared to cprev
// cnext is compared to cnext
// targ == cand always.
PHDIST_HAND  :
hand{
  // maximum values of the phoneme distance
  vv_max     DOUBLE; // vowel vowel
  vc_max     DOUBLE; // vowel cons
  vsil_max   DOUBLE; // vowel silence
  cc_max     DOUBLE; // cons  cons
```

```
  csil_max   DOUBLE; // cons sil

  // relative importance of vowel features (see phoneme_set
  // definitions) for vowel comparison. All are subsequently normalized
  // so that the sum of the weights is one.
  height DOUBLE; // height difference weight
  front  DOUBLE; // fronting difference weight
  round  DOUBLE; // rounding difference weight
  length DOUBLE; // length difference weight

  // relative importance of consonant features (see phoneme_set
  // definitions) for consonant comparison. All are subsequently
  // normalized so that the sum of the weights is one.
  type   DOUBLE; // consonant type mismatch  (binary)
  place  DOUBLE; // consonant place mismatch (binary)
  voiced DOUBLE; // voicing mismatch         (binary)
}

// hmm based phoneme distances
PHDIST_HMM :  hmm FILE ; // FILE is hmm distances file


// used to specify the default values for the target suitabilities
MTARGET_DEFAULT : default { MTARGET_SUITS }  | EMPTY

// used to specific class based suitabilities for target suitabilities
MTARGET_TARGETS : target  { MTARGET_TARGET } | EMPTY

// specifies all the target suitabilities
MTARGET_TARGET  : MTARGET_TARGET | EMPTY
// PHONEME_CLASS suitability targets
MTARGET_TARGET  : PHONEME_CLASS {MTARGET_SUIT}

// specifies target suitability
MTARGET_SUIT : MTARGET_SUIT | EMPTY
MTARGET_SUIT :

// quinPhoneme, constant value added if no triphone match or triphone
// matches but the phoneme after that does not. Basically encourages
// longer segments
quinPhoneme DOUBLE ;                        |
// (applied to value given by phoneme dist for scaling)
phoneme      { SUITABILITY_FN } |
duration     { SUITABILITY_FN }           | // duration mismatch
f0           { TRITARGET_SUITABILITY_FN } | // f0 mismatch
dF0          { SUITABILITY_FN } | // delta f0 mismatch (both left and right
                                  // deltas in one)
power        { TRITARGET_SUITABILITY_FN } | // power mismatch
dPower       { SUITABILITY_FN }             // delta power (as f0)
```

```
TRITARGET_SUITABILITY_FN : TRITARGET_SUITABILITY_FN | EMPTY
TRITARGET_SUITABILITY_FN :
  start{SUITABILITY_FN} | // start point suitability
  mid  {SUITABILITY_FN} | // mid point suitability
  end  {SUITABILITY_FN} | // end point suitability
  ends {SUITABILITY_FN} // start and end point specified in one


// ================================================
// Joint_functions

// use the joint functions to control the scoring of joint cost between units
// mjoint_fn1: costs

// joint_offset = constant value added regardless of anything else if
// the two segs are not adjacent in the database

// f0 : f0 mismatch suitability ( tolerance is taken from the midpoint
// of the two values == mean tolerance)

// dF0: delta F0 mismatch ( tolerance is mean tolerance )

// power : power mismatch suitability ( tolerance is taken from the
// min value of the two == min tolerance)

// dPower: delta power mismatch ( tolerance is mean tolerance )

// spectrum d_spectrum dd_spectrum

// suitability of the root mean squared error of the spectrum frames
// at the joint boundary (target is 0, tolerance taken at 0)

// mjoint_fn2: differs from mjoint_fn1 only in the spectral cost,
// which uses (where possible) the comparison of three spectram frames
// at the boundary (concatenation point) - just to get a larger
// context.

// mjoint_fn_hmm: differs from mjoint_fn1 only in the spectral cost,
// uses one frame at the boundary still, but the rmse error is
// normalised according to the variances of the bands og the spectrum,
// as indicated by the states of the relevant HMMs

JOINT_FN : mjoint_fn1    { MJOINT_FN } |
           mjoint_fn2    { MJOINT_FN } |
   mjoint_fn_hmm { JOINT_HMM MJOINT_FN }

JOINT_HMM: BINARY_JOINT_HMM | ASCII_JOINT_HMM

// binary format is mike Schuster's binary format
BINARY_JOINT_HMM :
```

```
   "left_hmm_model_file"  ;
   ""                     ;
   "right_hmm_model_file" ;
   ""                     ;

// ASCII format is HTK ASCII format
ASCII_JOINT_HMM :
   "left_hmm_model_file"  ;
   "left_model_hmmlist_file" ;
   "right_hmm_model_file" ;
          "right_model_hmmlist_file" ;


// specifies suitability functions for joint costs. If
// allow_real_joints is false then adjacent segments in the database
// will be given very high joint cost to prevent them from being
// selected.

// suitabilities are specified per joint class that is individually for
// vowel -> nasal liquid -> fricative ... etc. Unspecified
// suitabilities take on the default.
MJOINT_FN :
  allow_real_joints BOOLEAN ; |
  suitabilities {MJOINT_SUITABILITIES_DEFAULT MJOINT_SUITABILITIES_JOINT} |
  EMPTY


  // specify defaults for all joint classes
MJOINT_SUITABILITIES_DEFAULT:
  default { MJOINT_SUITABILITIES } |
  EMPTY

  // specify specific joints for joint classes
MJOINT_SUITABILITIES_JOINT  :
  joint { MJOINT_SUITABILITIES_JOINT_FROM } |
  EMPTY

  // specify class from
MJOINT_SUITABILITIES_JOINT_FROM : MJOINT_SUITABILITIES_JOINT_FROM | EMPTY
MJOINT_SUITABILITIES_JOINT_FROM : PHONEME_CLASS { MJOINT_SUITABILITIES_JOINT_TO }

// specify class to
MJOINT_SUITABILITIES_JOINT_TO : MJOINT_SUITABILITIES_JOINT_TO | EMPTY
MJOINT_SUITABILITIES_JOINT_TO : PHONEME_CLASS { MJOINT_SUITABILITIES }

// specify suitabilities
MJOINT_SUITABILITIES:
  joint_offset DOUBLE ;       |
  f0 {SUITABILITY_FN}         |
  dF0{SUITABILITY_FN}         |
```

```
  power {SUITABILITY_FN}       |
  dPower{SUITABILITY_FN}       |
  spectrum {SUITABILITY_FN}    |
  d_spectrum {SUITABILITY_FN}  |
  dd_spectrum {SUITABILITY_FN}


// specify the suitability functions for the joint to nothing for the
// first and last segments
START_JOINT_FN:
  stjoint{TJOINT_SUITABILITIES_DEFAULT TJOINT_SUITABILITIES_JOINT} | EMPTY
END_JOINT_FN  :
  etjoint{TJOINT_SUITABILITIES_DEFAULT TJOINT_SUITABILITIES_JOINT} | EMPTY

// specify defaults
TJOINT_SUITABILITIES_DEFAULT :
  default {TJOINT_SUITABILITIES} | EMPTY
TJOINT_SUITABILITIES_JOINT   :
  suitabilities { TJOINT_SUITABILITIES_JOINT_FROM }

// specify specific class based from
TJOINT_SUITABILITIES_FROM : TJOINT_SUITABILITIES_FROM | EMPTY
TJOINT_SUITABILITIES_FROM : PHONEME_CLASS {TJOINT_SUITABILITIES_TO}

// specify specific class based to
TJOINT_SUITABILITIES_TO : TJOINT_SUITABILITIES_TO | EMPTY
TJOINT_SUITABILITIES_TO : PHONEME_CLASS {TJOINT_SUITABILITIES}

// only power is used. target is 0, tolerance is taken from 0
TJOINT_SUITABILITIES : power {SUITABILITY_FN} | EMPTY


// ================================================
// suit = exp(-(pow(((X-x)/tolerance), rate)) + c) + min
// c = log (max - min)  0 <= min < max <= 1;
// tolerance is calculated as Y = mX + c
// where m and c are specified using
//  intercept and gradient
//  1 point and gradient
//  2 points
SUITABILITY_FN :
// tolerance (rate = 2, gradient = 0)
SUITABILITY_LIMITS DOUBLE |
// tolerance rate (gradient = 0)
SUITABILITY_LIMITS DOUBLE DOUBLE |
// intercept gradient rate
SUITABILITY_LIMITS DOUBLE DOUBLE DOUBLE |
// xpoint ypoint gradient rate
SUITABILITY_LIMITS DOUBLE DOUBLE DOUBLE DOUBLE |
// xpoint ypoint xpoint1 ypoint1 rate
```

```
SUITABILITY_LIMITS DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE

SUITABILITY_LIMITS: EMPTY |              // 0 1 defaults
                    { DOUBLE } |         // max
                    { DOUBLE DOUBLE }    // min max

EMPTY:
BOOLEAN : 0 | 1 // 0 is false 1 is true
INTEGER : [0-9]+
DOUBLE  : [0-9]*[.]?[0-9]+
PHONEME_CLASS : vowel | liquid | nasal | fricative |
                affricate | plosive | silence
```

Chapter 3

sdb_batch

The sdb_batch is a utility that will take commands from a batch file or from std input) and execute those commands. Commands include things such as database loading, segment selection and testseg. A full listing of the commands follows, all commands must be terminated by a semi-colon.

3.1 Usage

standard usage,

```
sdb_batch batch_file
```

reading from std input

```
sdb_batch -
```

3.2 commands

- database name db_specification_file ;

  Load an ascii format database, under the given name and specified by the database specification file given. If there already exists a database under the given name, this is a nop.

- binary_database name binary_db_file;

  Load a binary format database, under the given name, found in the given file. If there already exists a database under the given name, this is a nop.

- delete_database name;

  Delete the database found under the given name. Prerequisite to loading a new database under a name that already exists.

- select name;

  Select the database loaded under the given name. An exception is thrown if the so named database is non-existant.

- load_spectra name wspec.lib;

  Load spectra from the wspec lib given for the named database. An exception is thrown if either the named db does not exist or the wspeclib file is non-existant. If spectra are already loaded, this is a nop.

- delete_spectra name;

  Delete loaded spectra fro the named database. Prerequisite to changing the spectra that are loaded for a particular database.

- outdir output_directory;

  Set the directory into which output files will be directed. This directory must exist.

- log logfile;

  Commands are echoed to this file as they are executed. Only the file name is required the directoy is set using outdir. There are some special values,

unset or set to "none"	no log is produced
set to cout	stdout
set to cerr	stderr

  All other values are taken as filenames.

- include batch_file;

  include commands found in the named batch file. An exception is thrown if this file does not exist.

- gus_params parameters_file;

  Load the GUS (search) parameters found in the given file.

- testseg filenumber outfile [units] [sel] [binary_sel] [path];

  execute a testseg. Excluded the given file and then resynthesize. The file number identifies the number of this file in the database (determined by the db specification file). Outfile is the *basename* of the output files that will be written. They will be written into *outdir*. There are four types of output file to choose,

option	file information
units	units file suitable for use with unitplay (concat)
sel	ascii data showing the targets and selected segments
binary_sel	binary version of sel (for use with the chatrizer)
path	path information (overall, target and join costs for each step in the search - best path only)

- savesegs filenumber <udb| sdb> outfile;

  save segments in udb or sdb format. This only savese segments for testsegs, that is. the data in the database. the output file is written to *outdir*.

- segfile segment_file outfile [units] [sel] [binary_sel] [path];

  Select segments based on the segments specified in the given segment file. otherwise, works the same as testseg.

- print "text...";

  print the given text to the logfile. Useful to notify/mark the completion of commands in the batch (especially when reading from stdin)

- phonemes outfile;

  Print a summary of the phonemes and their counts in the database to the given file. extension is `phonemes`.

- files outfile;

  List the files in the database and their file numbers to the given `outfile`. extension is `files`.

- exclude_file filenumber [filenumber...];

  exclude the listed files from the database. This means that segments found in these files cannot be selected in a search.

- unexclude_file filenumber [filenumber...];

  unexclude excluded files from the database. This will re-include files which have been excluded.

- exclude_seg filenum segnum [filenum segnum...];

  Exclude the identified segments from the database. This means that they will not be used in a search.

- unexclude_seg filenum segnum [filenum segnum...];

  Unexclude the identified segments from the database. This means that, if excluded, they can now be used again in the search.

- gus_debug outfile;

  switch on copious debugging information. Written to outfile.

Chapter 4

SDB Database Formats

The SDB can handle two kinds of database format. The first is an ASCII format, described below. The second is it's own binary format. The binary format is simply produced by loading an ASCII format database and saving as binary. ASCII format databases have the advantage that they are flexible and share data sources. Binary databases are, however, much faster to load.

4.1 Binary Databases

Binary databases can be created from ASCII databases using the program *SDBcompile*. This simply loads the ASCII DB and then saves it in binary format. The usage is

```
SDBcompile infile outfile

infile  = ascii db specification file
outfile = binary db filename
```

4.2 ASCII Format Databases

Making a segment database is very simple. We require 3 pieces of (synchronized) information.

file	purpose
label	timings for label boundaries
f0	F0 values of the signal
power	power values of the signal

4.3 File Formats

Label Files

Label files are expected to be in ESPS format.

Database Specification Files

The database is the specified using a database specification file. The format of this file is as follows,

```
dbname:       <SPEAKER NAME>
db_directory: <PATH TO DIRECTORY CONTAINING LAB, F0 AND POWER DIRECTORIES>
sample_rate:  <WAVE FILE SAMPLE RATE - required but unused>
phoneme_set:  <PHONEME SET>
files:
<FILE BASE NAME>
<FILE BASE NAME>
...
;
```

The file basenames are a list of the files to be included in this database. This list makes it possible to have multiple databases using the same data but being of differing sizes.

F0 and Power - frame based data

F0 and power files are loaded from frame based data (data is sampled at regular intervals). The file contains the following header fields. (Please note that there must be no space between the field name and the colon separator in these files.)

item	purpose
frame_count	number of frames in the file (this must match)
frame_duration	duration or interval between sampled frames
start_time	time of the start of the first frame in the data file

Sampling of this nature often employs a window. This windowing means that the frames do not start exactly in line with the beginning of the datafile from which they are calculated. The *start_time* field is used to ensure correct alignment of the frames with the data in the datafiles.

```
frame_count:    <int NUMBER OF FRAMES>
frame_duration: <double DURATION OF EACH FRAME>
start_time:     <double OFFSET TIME OF THE FIRST FRAME>
frames:
0
0
0
0
0
0
0
171.63686
180.56895
146.52338
125.80508
127.78784
119.78716
121.6573
...
```

4.4 Database Directory Structure

The database directory is expected to follow the following structure,

```
.../my_db/
          lab/
              basename1.lab
              basename2.lab
                 ...
          f0/
             basename1.f0
             basename2.f0
                 ...
          power/
             basename1.power
             basename2.power
                 ...
```

The following directory structure is useful for making databases using different selections of files,

```
.../my_db/
          db.autolabel/
             lab   -> ../data/lab_auto
             power -> ../data/power
             f0    -> ../dasta/f0
          db.handlabel/
             lab   -> ../data/lab_hand
             power -> ../data/power
             f0    -> ../data/f0

          db.whatever/
                ...etc


        data/
                lab_auto/
                  base1.lab
                  base2.lab
                lab_hand/
                  base1.lab
                  base2.lab
                power/
                  base1.power
                  base2.power
                f0/
                  base1.f0
                  base2.f0
                f0_hand/
                  base1.f0
```

```
        base2.f0
    ...etc
```

Then, in the database specification files the datebase directories will be, respectively,

db.auto_lab : .../my_db/db.autolabel

db.hand_lab : .../my_db/db.handlabel

whatever : .../my_db/db.whatever

The use of data sharing is important to keep the storage needs for the raw database data as small as possible.

Spectral Information

Spectral information supplied by HTK lacks one vital component, that is the window size that was employed when the data was created. Without this information the data cannot be aligned correctly with the wave file that it represents. The program `htk2wspec` found in the sdb `bin/` directory takes HTK files and a window size and saves this new information. This has the added advantage that it removes the third party copyright/licensing problems from the file format. Spectra are then compacted into a single file (wspeclib) using the `filelib` utility.

Chapter 5

Making the SDB Files

There are a number of utilities available that can be used to make the ASCII format files required for loading by an SDB. Many of these are currently third party products. However provided the final ASCII files are in the correct format, any data extraction methods can be used.

The following text describes the processes currently used to produce the necessary files. We assume that label and wav files are already available and there will be no discussion of the production of these files.

5.1 F0 and Power Extraction

ESPS get_f0 is used for f0 and power extraction. This requires only the wave files, in ESPS format. The files produced by get_f0 are then post-processed by a short perl script which produces the ASCII format files for the SDB.

The perl script is called *esps.processf0s.pl.* for a brief summary of usage, type,

```
esps.processf0s.pl -help
```

the overall process is then,

```
foreach file ( filebases ) {
 btosps -c '' -f DB_SAMPLE_RATE wav/file.wav esps/file.esps
 get_f0 -P get_f0.config esps/file.esps getF0/file.f0
}

esps.processf0s.pl -verbose -outdir . -nonorm getF0/*.f0
```

5.2 Filebases

A list of the filebases of the files in the database is created using the perl script *filebases.pl.* For a brief summary of usage, type,

```
filebases.pl -help
```

5.3 Make

The following GNU style Makefile links all the processes together, and providing the directory structure described is followed will produce all the files required by an SDB for loading. Except the database specification file, which is up to the user.

Directory Structure

```
.../db_data/
        get_f0.config    ; config file for esps get_f0
        wav/*.wav
        lab/*.lab        ; not required for make.
        getF0/           ; empty directory into which
                         ; get_F0 output files will be placed
```

For *db_lx* lx esps files should be available in *esps_lx* and an dir called getF0_lx created. For *db_hf0* (hand f0) the getF0 directory should be replaced with a link to the directory containing the hand corrected f0s. Write permission should be removed from these, for safety.

Makefile

```
EPDA = /home/as71/ding/Pitch/epda

wav_files      = $(shell ls wav/*.wav)
esps_files     = $(wav_files:wav/%.wav=esps/%.esps)
lx_files       = $(wav_files:wav/%.wav=esps_lx/%.lx)
getf0_files    = $(wav_files:wav/%.wav=getF0/%.f0)
getf0_h_files  = $(wav_files:wav/%.wav=getF0_h/%.f0)
getf0_lx_files = $(wav_files:wav/%.wav=getF0_lx/%.f0)
epda_files     = $(wav_files:wav/%.wav=epda/%.epda)

# only one of the possible files
formant_files = $(wav_files:wav/%.wav=formant/%.f0)

SAMPLE_RATE = 16000

all :
        @echo choose a target!

db     : f0 power filebases
db_lx  : f0_lx power filebases
db_hf0 : f0_h power filebases

esps    : $(esps_files)
getf0   : $(getf0_files)
epda    : $(epda_files)
f0_h    : make_fp_h
```

```
f0_lx   : make_fp_lx
f0      : make_fp
power   : make_fp
f0_epda : make_f0_epda

esps/%.esps : wav/%.wav
        @echo $< -> $@
        btosps -c '' -f $(SAMPLE_RATE) $< $@

epda/%.epda : wav/%.wav
        @echo $< -> $@
        $(EPDA) $< $@ $(SAMPLE_RATE) 0
# no smooth

getF0/%.f0 : wav/%.wav get_f0.config
        @echo $< -> $@
        -btosps -c '' -f $(SAMPLE_RATE) $< - | \
         get_f0 -P get_f0.config - $@

getF0_lx/%.f0 : esps_lx/%.lx get_f0.config
        @echo $< -> $@
        -bhd $< - | btosps -c '' -f $(SAMPLE_RATE) - - | \
         get_f0 -P get_f0.config - $@

make_fp : $(getf0_files)
        esps-processf0s.pl -verbose -nonorm -outdir . $^

make_fp_lx : $(getf0_lx_files)
        esps-processf0s.pl -verbose -nonorm -outdir lx $^

make_fp_h : $(getf0_h_files)
        esps-processf0s.pl -verbose -nonorm -outdir f0_h $^

filebases: $(wav_files)
        filebase.pl $^ > $@
```

Chapter 6

Data and File Locations

This chapter lists the current placement of my data files and directories and what you should hope to find therein. There are many links within these directories. Although this can make things appear a little complicated, it was necessary in order to spread the volume of data.

6.1 The SDB

The SDB is found in,

```
/homes/dsann/chatr-dsann/    ; root dir
 src/      ; the source is in here
 obj/      ; directory for object files
 main/     ; source files containing the main function
 bin/      ; resulting executable files
 Makefile ; gnu type makefile
```

It is also available in the CVSROOT directory `/DB/PI/CVSROOT` and can be checked out using `cvs co chatr-dsann`.

6.2 Databases and compiled SDBs

Database data	`/homes/dsann/chatr.lib/DBs/db_name/`
SDB specifications	`/homes/dsann/chatr.lib/SDBs/speaker_name`
Binary SDBs	`/homes/dsann/chatr.lib/SDBs/bin/speaker_name`

6.3 HMM Distance Measures

The code for calculating HMM distance measures is found in `/homes/dsann/Work62/HMM/`.

6.4 Additional Data

Additional data is found in the directories named `chatr.data*`. This data consists of various things from HMM training information to database spectra. Please see the `README` files in these directories for futher details.

6.5 Utilities

- `/homes/dsann/chatr-dsann/src/bin/`
  - `SDBcompile dbspec binfile`
    compile ASCII database specified `dbspec` to binary db in `binfile`.
  - `sdb_batch batch_file` execute the commands given in the `batch_file`
  - `htk2wspec window htkspecfile`
    convert HTK file to wspec file. Adding the specified window size to the header. Not every HTK format is handled.
- `/homes/dsann/bin/all/perl5/`
  - `esps-processf0s.pl [options] getF0files...` Process and extract f0 and power data from ESPS `get_f0` generated files. use the `-help` option to find out all the options.
  - `filebase.pl files` Remove the last extension from all the files and print the result to stdout.
- `/homes/dsann/bin/all/perl5/HTK/filelist.pl` generate HTK style file lists (scripts). use the -help option to find out more.
- `/homes/dsann/Work62/HMM/bin`
  - `hmm_phdist htkModels htkModelList` generate ASCII phoneme distance data for bi- or tri- phone models. result is written on stdout.
  - `hmmCompile htkModels htkModelList binaryfile` Write HTK models in Mike Schuster's HMM binary format.
- `/homes/dsann/Work62/patrick/chatrizer/Chatrizer.java` The chatrizer interface. Use `java Chatrizer`.
- `/homes/dsann/Work62/patrick/chatrizer/bin/`
  - `sdb_server.pl` Server maintaining a continuously running selection. This saves time on loading of spectral information which can be very time consuming.
  - `sdb_client.pl` Client to talk to the `sdb_server`. Simply 'cat' the input commands to the client's stdin.

Chapter 7

Conclusions

7.1 Segment Concatenation

The ability to freely chose segments, from our speech databases, that match well to requested input segments depends crucially on how well we can join different segments together. If the signals must be very very similar at the joint points to avoid audible errors then our choice is extremely constrained and it becomes almost impossible to select that which is desired. Currently within the CHATR approach, segment context is so important that it places constraints which cannot be met while also matching input prosody.

Prosody is very important, clearly, to the intelligibility of synthesized speech. Listeners expect certain acoustic events to occur at certain times. In English, and many European languages, stress timing is crucial. In French and many other languages syllable timing is equally important. If durations are incorrect and unexpected events occur, the listener easily becomes confused, and the speech unintelligible. Incorrect f0 can be forgiven in many cases in isolated sentences. That is because they have no context and the reading can be deamed 'plausible'. However, within a discourse, incorrect prosody will render the meaning at best ambiguous and at worst, wrong.

If the constraints on segment concatenation are not lifted or relaxed, it will be extremely difficult to improve on prosody produced by CHATR. No matter how good the prediction the selection cannot match what is asked for.

In order to relax these constraints, much work must be conducted specifically on the problem of segment concatenation. The solution to this problem will certainly necessitate breaking the 'CHATR tradition', resorting to the use of signal processing.

7.2 Spectral Distance for Joint Quality Estimation

It was found that the use of cepstral frames for estimation of joint signal mismatched at phoneme boundaries was ineffective. There are several reasons which explain why this might be expected.

1. The phoneme boundaries represent the point in the signal at which articulatory movement is maximal in most cases. This means that the signal is very rapidly changing its characteristics. Frame based spectral evaluation assumes that the signal is *stationary*, a fact that is clearly not true at the points we are currently concerned with. This makes the spectral information inaccurate.

2. The use of 'Dumb' concatenation splices the signals without any further processing. The use of 10 milli-second spaced frames with 25 milli-second analysis windows cannot really be expected to be specific enough to help concatenation at the specific point we are interested in, in the general case.

3. The use of 'Dumb' concatenation (splicing) means that many segments that are present in the database can simply not be joined because they will not join without damaging spectral discontinuities.

7.3 Suggestions

I have a number of suggestions which may help to improve synthesis.

I suggest moving the concatenation points away from phoneme boundaries in the general case an into points where the signal is (more) stationary. This will enable accurate signal analysis and so allow better joining points to be identified. It will also allow the use of signal processing techniques at the boundary points where necessary. To achieve this, one might move to half phoneme sized segments, this would allow concatenation points similar to those found in diphone concatenation while still allowing the possibility of phoneme boundary joints where this was necessary.

Since (short term) stationary points in the signal are likely to offer the best analysis potential and the best amenity to smooth concatenation. An analysis of the stationarity of the signal might be conducted to determine regions for good concatenation or a measure of concatenation preferability at different points.

Including the use of flexible concatenation points into the segment selection scheme will not be a trivial task, and will need careful planning.

Other signal properties may also lend themselves well to better concatenation points. For example, masking of errors in fricative sounds is much higher than that non-noisy resonant sounds. An investigation of which of these properties can be exploited to our advantage would greatly help future progress.

In conclusion,

- The choice of concatenation point within the signal should be closely examined.

- The method of concatenation should be improved, this solution will almost certainly have to involve some signal processing.

Appendix A

Comments on the UDB

The UDB and the SDB are intended to serve the same purpose. The SDB is intended as a replacement for the UDB in two ways. Firstly it represents the database data more accurately than the UDB, ensuring correct time alignment of f0, power and spectral information. Secondly it draws a clear line separating the search process from the prediction and concatenation phases [1]. This separation avoids a problem with the UDB which is that information assumed to be present may in fact be missing if certain input methods are used. It also makes the code clearer and easier to understand and prevents mixing of unrelated code.

There are several problems, in my opinion with the UDB.

1. Rounding and smoothing of database data leads to an inability to distinguish segments accurately.
2. Integer rounded values used for features will lead to accumulated rounding errors which could be simply avoided.
3. The extensive use of global variables in both C and the 'lisp-environment' lead to a number of problems. Values changing unexpectedly as the side effect of some function. Persistent values after a speaker change. This makes the functioning of the code extremely difficult to understand and therefore leads to many hard to find (and sometimes never found or even noticed) bugs.
4. Trained Distance functions. There is a fundamental problem with the generic distance functions.

A.1 Generic distance functions

The generic distance functions are based upon the following theory. Given a function, $f(a, b, c, d, \ldots)$ estimate that function with another function $g(p, q, r, s \ldots)$. *The variables of f must be distinct from the variables of g.* If this is not the case, the f will simply be an imperfect identity mapping, and it would be correct simply to use g as defined. In this particular case g is a *hand specified* function of three variables,

[1] Although the search and concatenation may have strong constraining links, as processes they are not necessarily directly connected.

1. cepstral distance
2. f0
3. power

and f a function of various features found in the input and target segments to the udb. These include,

1. phoneme features (rounding voicing ... etc)
2. f0
3. power
4. duration

The use of f0 and power violates the condition specified above. Namely identity mapping of variables. And therefore in these aspects can do *no better than* the originally defined function. In which case it is much more sensible to specify them directly.

The use of cepstral distance is acceptable. Although, the linear alignment cepstral distance employed in the distance function training is a poor definition and is very unsuitable for phonemes such as plosives.

The effect of the these factors is that the main correlation produced by the learning algorithms is in the dimension of f0 (which would be expected). All other features correlate very poorly, if at all.

The lack of correlation is easily demonstrated by plotting the true cepstral distance against the feature based distance. That is f against g. If the estimation were perfect then we would expect a straight line. A widely scattered plot would indicate no correlation.

Figure A.1: cepstral - feature distance : correlation for a voiced phoneme

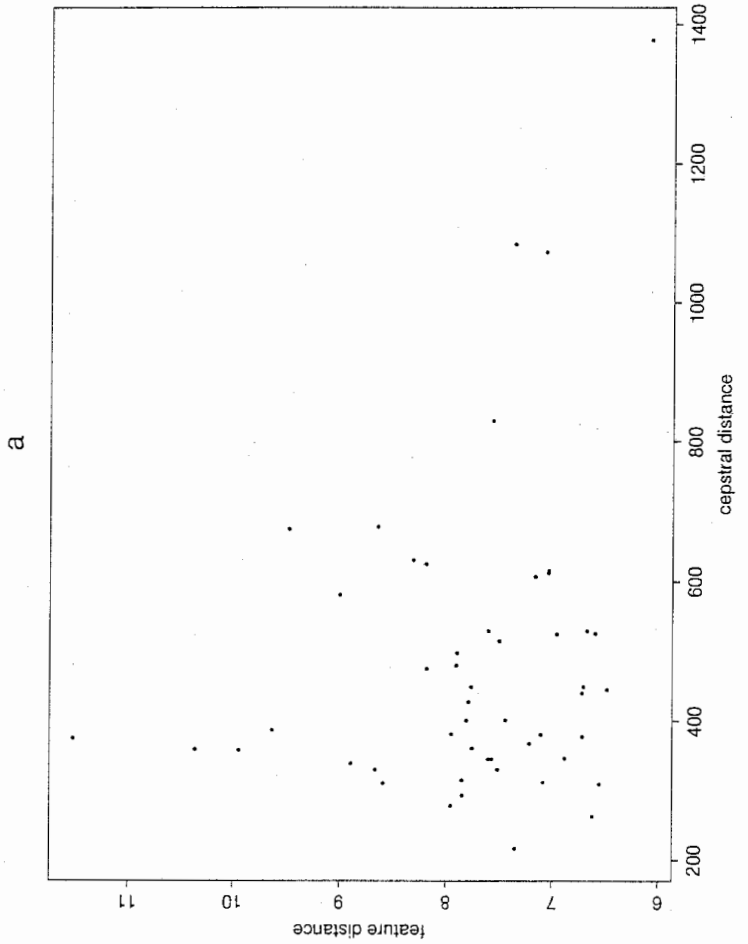
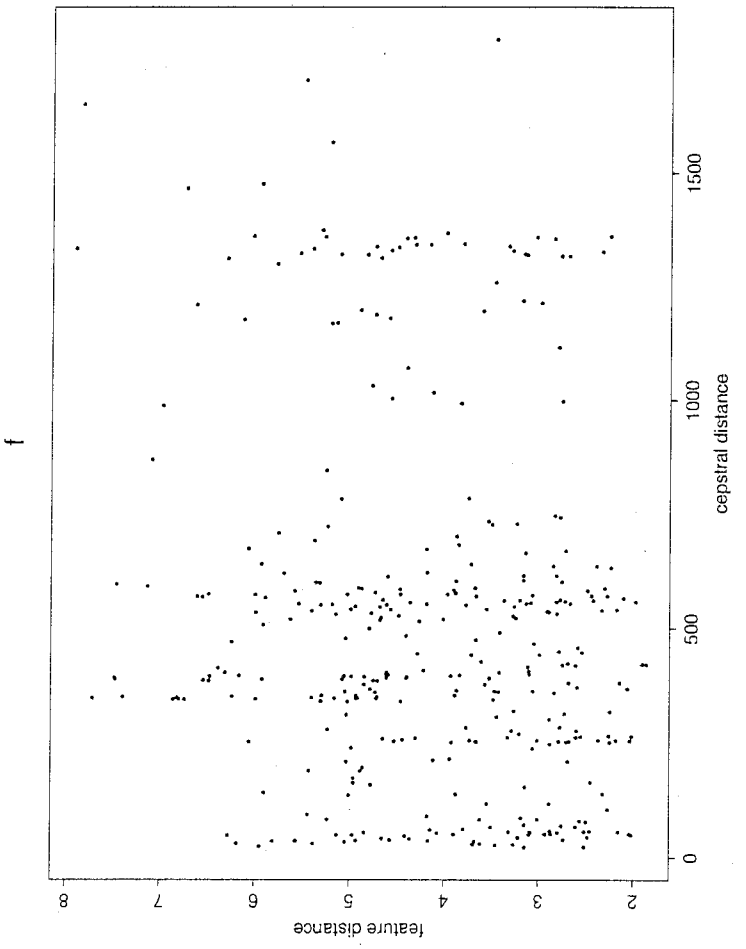


Figure A.2: cepstral - feature distance : correlation for an unvoiced phoneme



Bibliography

[htk] Entropic Research Labs. *HTK - Hidden Markov Model Toolkit.*.

[kullbark] Christopher M. Bishop. *Neural Networks for Pattern Recognition pp 59, 244.*. Clarendon press - Oxford 1995. ISBN 019 853864 2.