EECS E6870 - Speech Recognition
Lecture 2

Stanley F. Chen, Michael A. Picheny and Bhuvana Ramabhadran

IBM T.J. Watson Research Center

Yorktown Heights, NY, USA

stanchen@us.ibm.com

picheny@us.ibm.com

bhuvana@us.ibm.com

15 September 2009

Outline of Today's Lecture

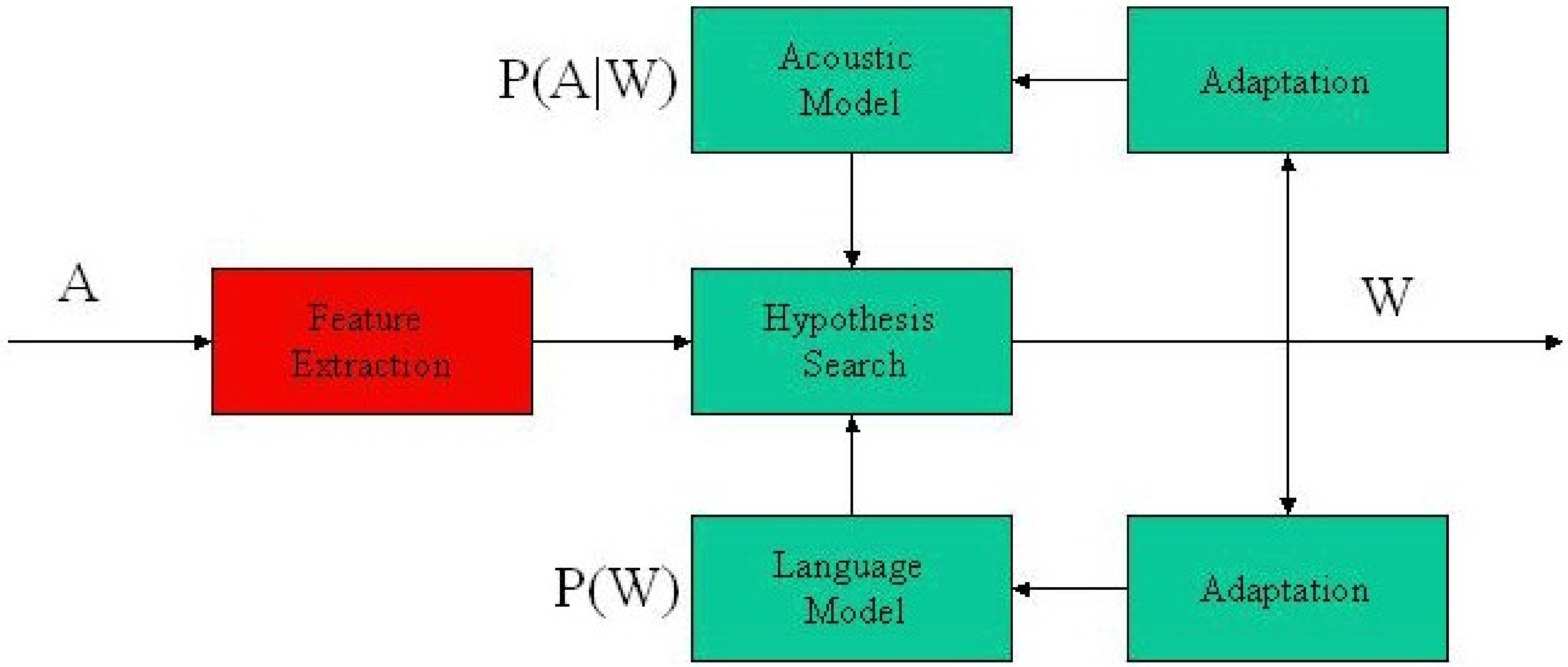
- Administrivia
- Feature Extraction
- Brief Break
- Dynamic Time Warping

Administrivia

- Feedback:
  - Get slides, readings beforehand
  - A little fast in some areas
  - More interactive, if possible
- Goals:
  - General understanding of ASR
  - State-of-the-art, current research trends
  - More theory, less programming
  - Build simple recognizer

Will make sure slides and readings provided in advance in the future, (slides should be available night before) change the pace, and try to engage more.

Feature Extraction



What will be “Featured”?

- Linear Prediction (LPC)
- Mel-Scale Cepstral Coefficients (MFCCs)
- Perceptual Linear Prediction (PLP)
- Deltas and Double-Deltas
- Recent developments: Tandem models

Figures from Holmes, HAH or R+J unless indicated otherwise.

Goals of Feature Extraction

- What do YOU think the goals of Feature Extraction should be?

Goals of Feature Extraction

- Capture essential information for sound and word identification
- Compress information into a manageable form
- Make it easy to factor out irrelevant information to recognition such as long-term channel transmission characteristics.

What are some possibilities?

- What sorts of features would you extract?

What are some possibilities?

- Model speech signal with a parsimonious set of parameters that best represent the signal.
- Use some type of function approximation such as Taylor or Fourier series
- Exploit correlations in the signal to reduce the the number of parameters
- Exploit knowledge of perceptual processing to eliminate irrelevant variation - for example, fine frequency structure at high frequencies.

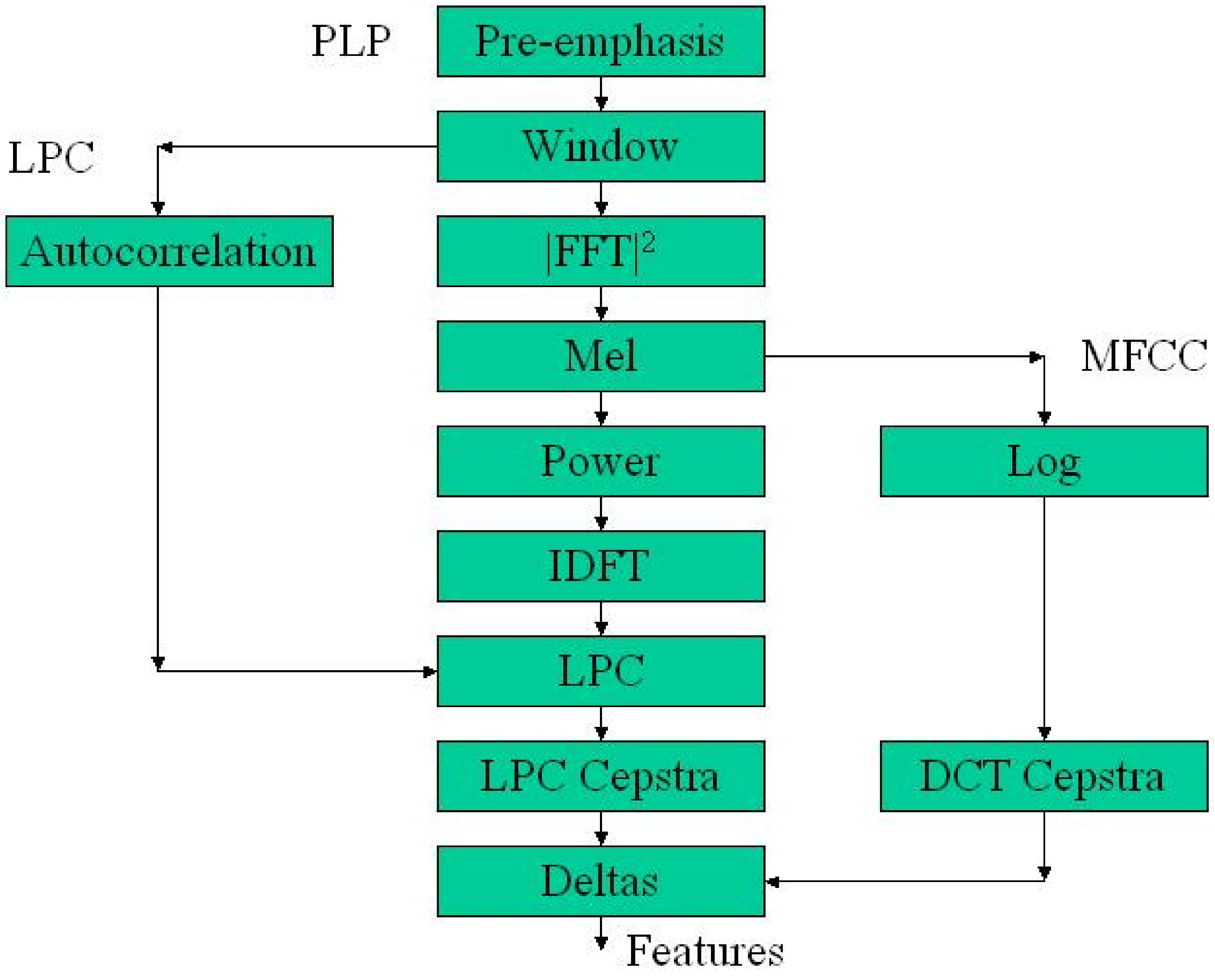
Historical Digression

- 1950s-1960s - Analog Filter Banks
- 1970s - LPC
- 1980s - LPC Cepstra
- 1990s - MFCC and PLP
- 2000s - Posteriors, and multistream combinations

Sounded good but never made it

- Articulatory features
- Neural Firing Rate Models
- Formant Frequencies
- Pitch (except for tonal languages such as Mandarin)

Three Main Schemes



Pre-Emphasis

Purpose: Compensate for 6dB/octave falloff due to glottal-source and lip-radiation combination.

Assume our input signal is $x[n]$. Pre-emphasis is implemented via very simple filter:

$$y[n] = x[n] + ax[n-1]$$

To analyze this, let's use the "Z-Transform" introduced in Lecture 1. Since $x[n-1] = z^{-1}x[n]$ we can write

$$Y(z) = X(z)H(z) = X(z)(1 + az^{-1})$$

If we substitute $z = e^{j\omega}$, we can write

$$\begin{aligned} |H(e^{j\omega})|^2 &= |1 + a(\cos\omega - j\sin\omega)|^2 \\ &= 1 + a^2 + 2a\cos\omega \end{aligned}$$

or in dB

$$10\log_{10}|H(e^{j\omega})|^2 = 10\log_{10}(1 + a^2 + 2a\cos\omega)$$

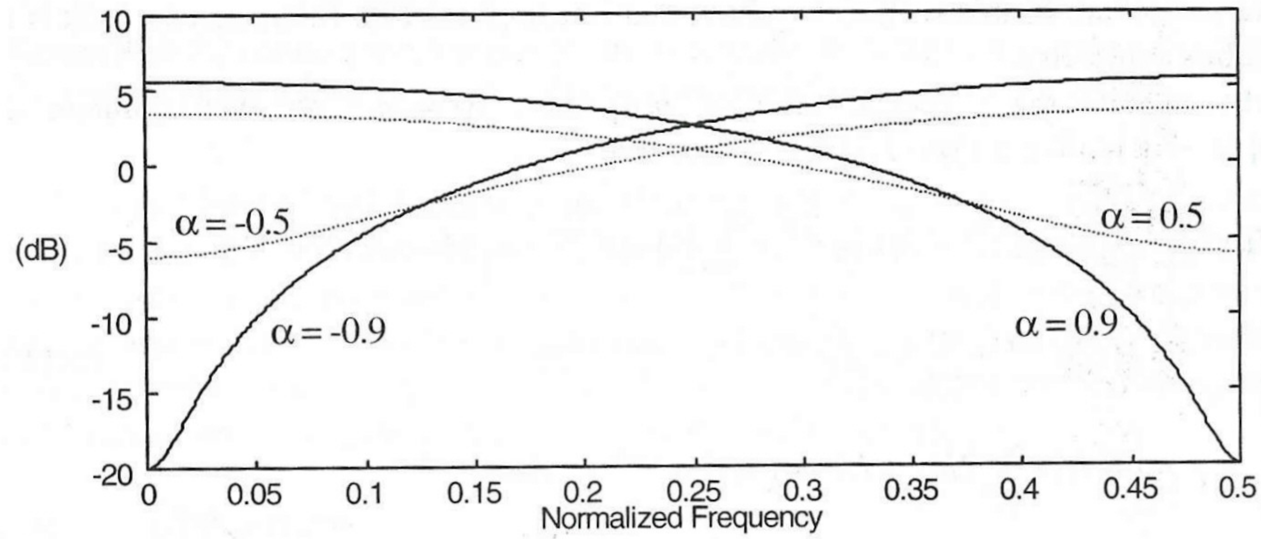


Figure 5.21 Frequency response of the first order FIR filter for various values of α.

For $a > 0$ we have a low-pass filter and for $a < 0$ we have a high-pass filter, also called a "pre-emphasis" filter because the frequency response rises smoothly from low to high frequencies.

Uses are:

- Improve LPC estimates (works better with “flatter” spectra)
- Reduce or eliminate DC offsets
- Mimic equal-loudness contours (higher frequency sounds appear “louder” than low frequency sounds for the same amplitude)

Basic Speech Processing Unit - the Frame

Block input into frames consisting of about 20 msec segments (200 samples at a 10 KHz sampling rate). More specifically, define

$$x^m[n] = x[n - mF]w[n]$$

as frame m to be processed where F is the spacing frames and $w[n]$ is our window of length N.

Let us also assume that $x[n] = 0$ for $n < 0$ and $n > L - 1$. For consistency with all the processing schemes, let us assume x has already been pre-emphasized.

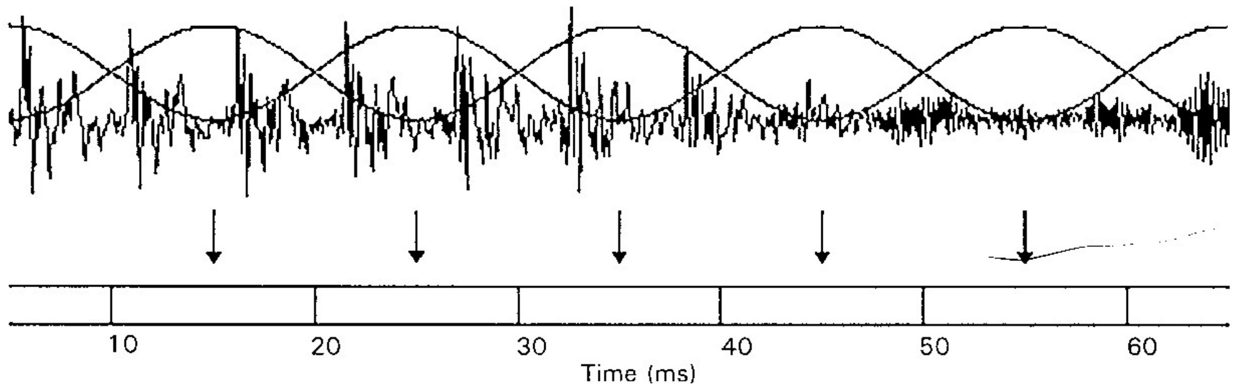


Figure 10.1 Analysis of a speech signal into a sequence of frames. This example shows a 20 ms Hanning window applied at 10 ms intervals to give a frame rate of 100 frames/s.

How do we choose the window $w[n]$, the frame spacing, F, and the window length, N?

- Experiments in speech coding intelligibility suggest that F should be around 10 msec. For F greater than 20 msec one starts hearing noticeable distortion. Less and things do not appreciably improve.
- From last week, we know that Hamming windows are good.

So what window length should we use?

- If too long, vocal tract will be non-stationary; smooth out transients like stops.
- If too short, spectral output will be too variable with respect to window placement.

Usually choose 20-25 msec window length as a compromise.

Effects of Windowing

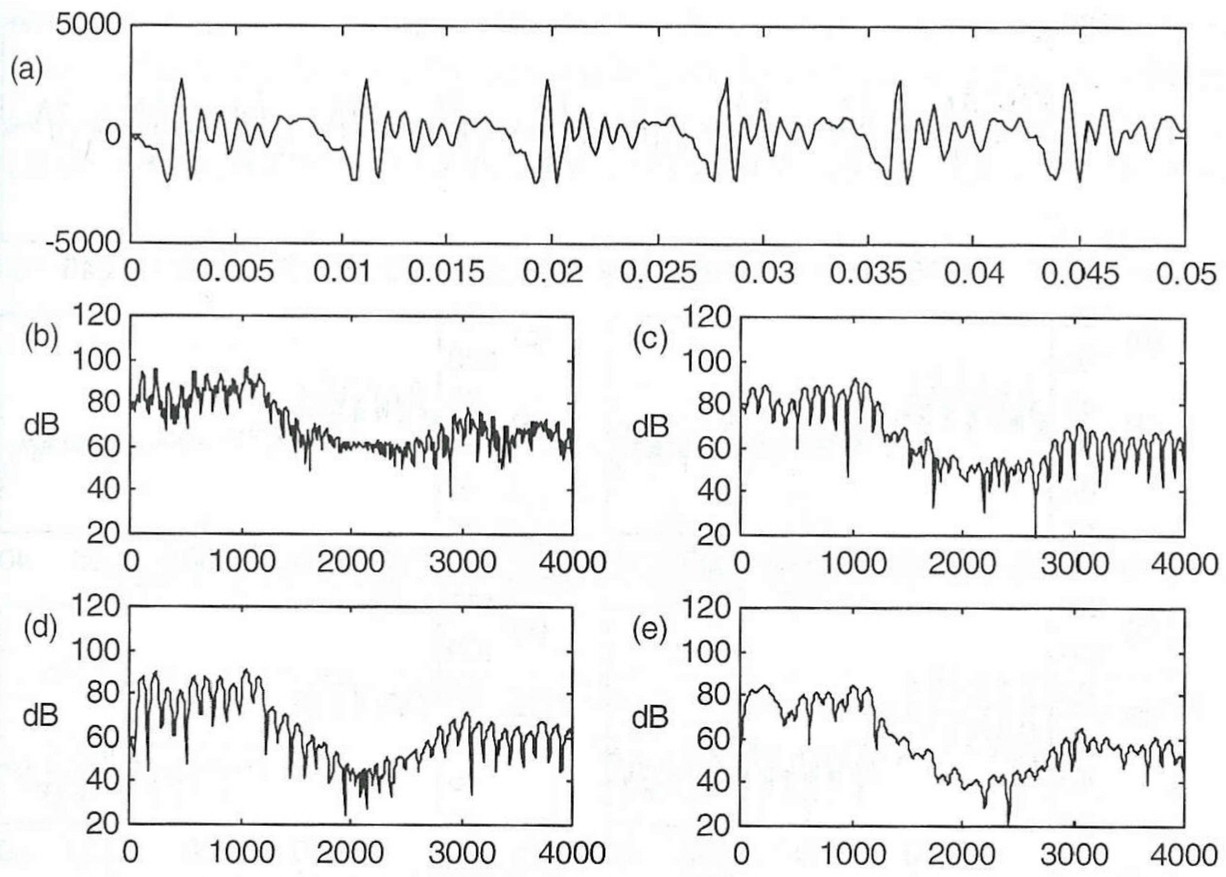


Figure 6.3 Short-time spectrum of male voiced speech (vowel /ah/ with local pitch of 110Hz): (a) time signal, spectra obtained with (b) 30 ms rectangular window and (c) 15 ms rectangular window, (d) 30 ms Hamming window, (e) 15 ms Hamming window. The window lobes are not visible in (e), since the window is shorter than 2 times the pitch period. Note the spectral leakage present in (b).

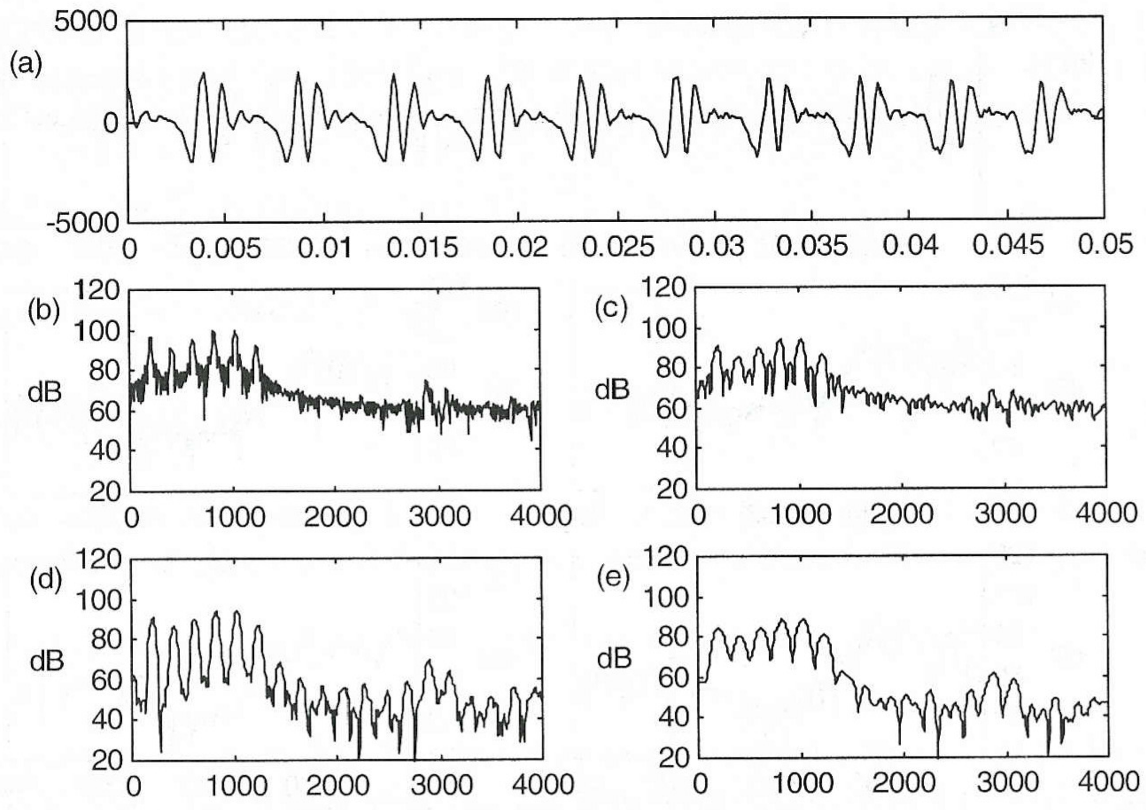


Figure 6.4 Short-time spectrum of female voiced speech (vowel /aa/ with local pitch of 200Hz): (a) time signal, spectra obtained with (b) 30 ms rectangular window and (c) 15 ms rectangular window, (d) 30 ms Hamming window, (e) 15 ms Hamming window. In all cases the window lobes are visible, since the window is longer than 2 times the pitch period. Note the spectral leakage present in (b) and (c).

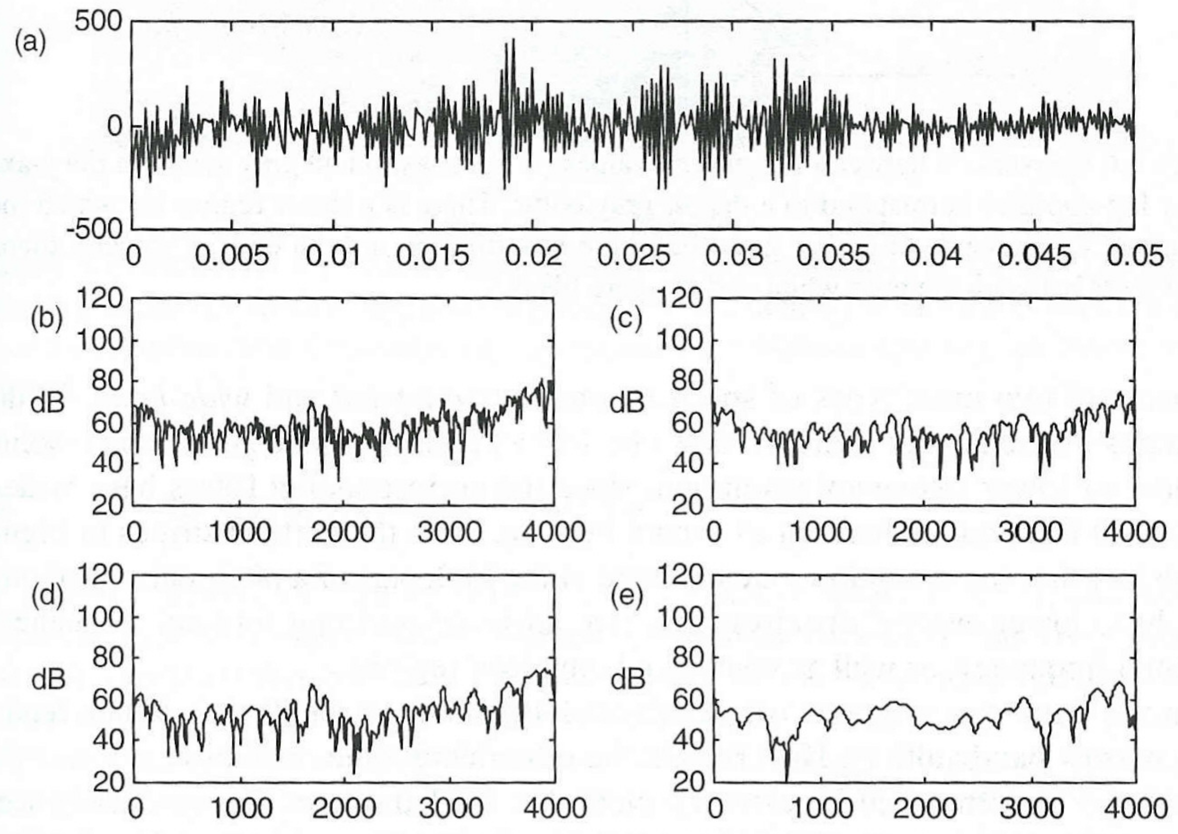
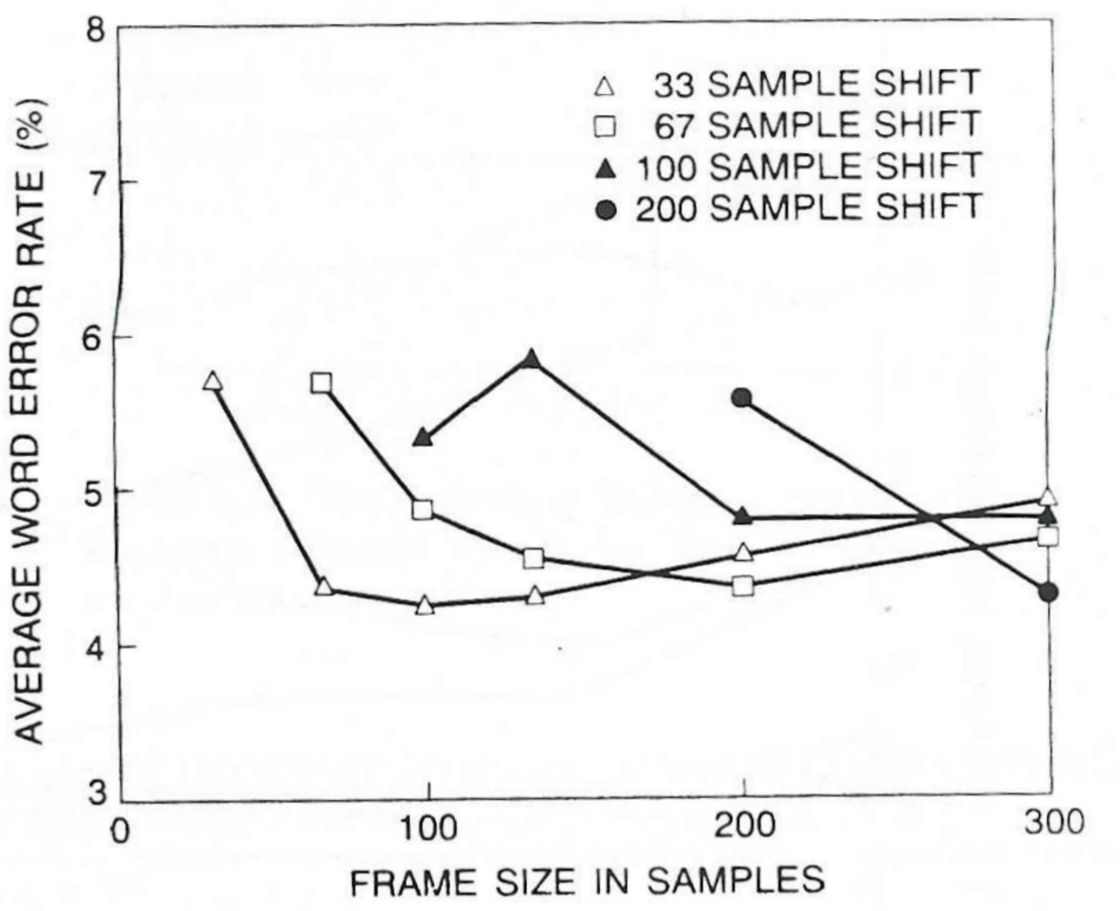


Figure 6.5 Short-time spectrum of unvoiced speech: (a) time signal, (b) 30 ms rectangular window, (c) 15 ms rectangular window, (d) 30 ms Hamming window, (e) 15 ms Hamming window.

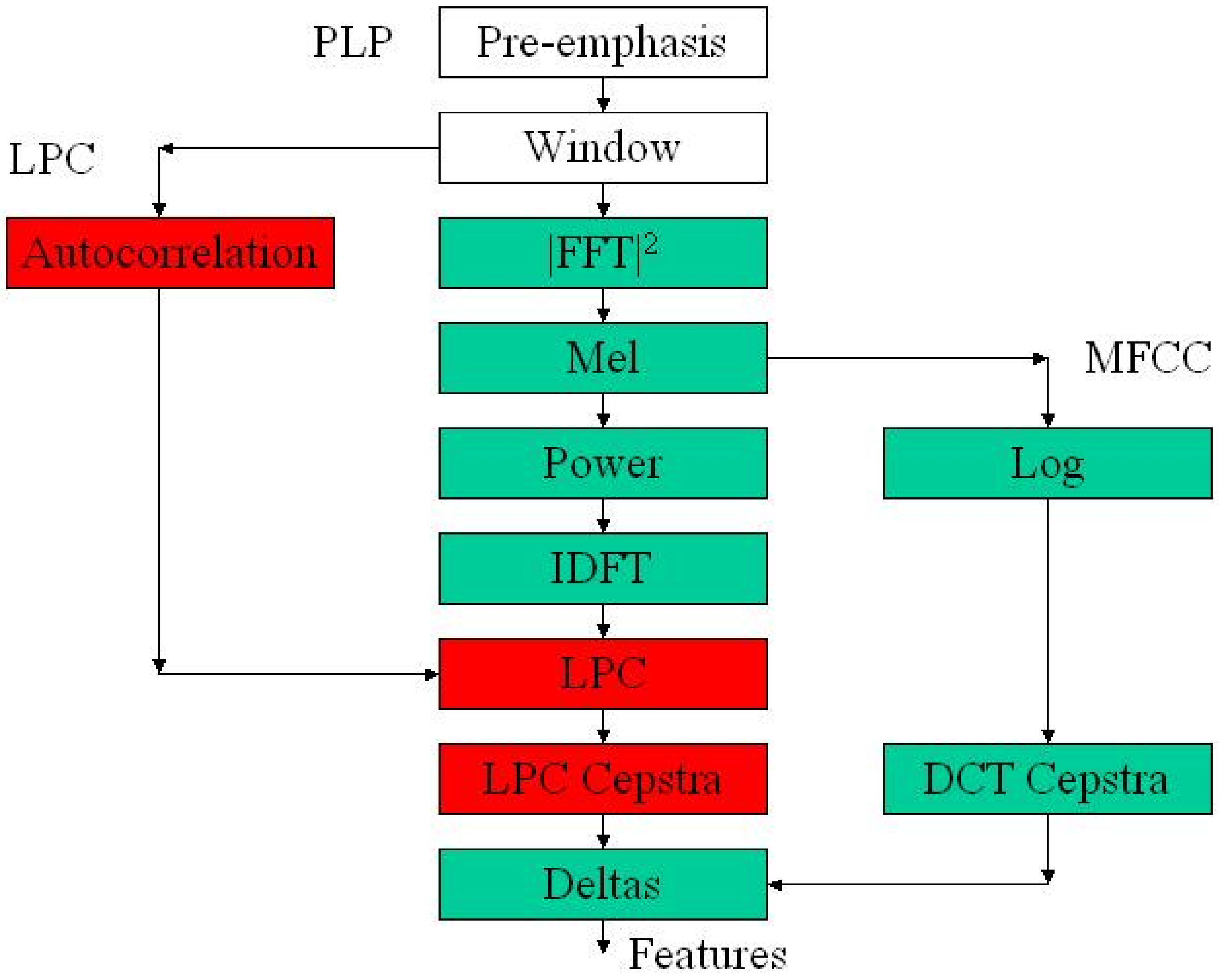
- What do you notice about all these spectra?

Optimal Frame Rate



- Few studies of frame rate vs. error rate
- Above curves suggest that the frame rate should be one-third of the frame size

Linear Prediction



Linear Prediction - Motivation

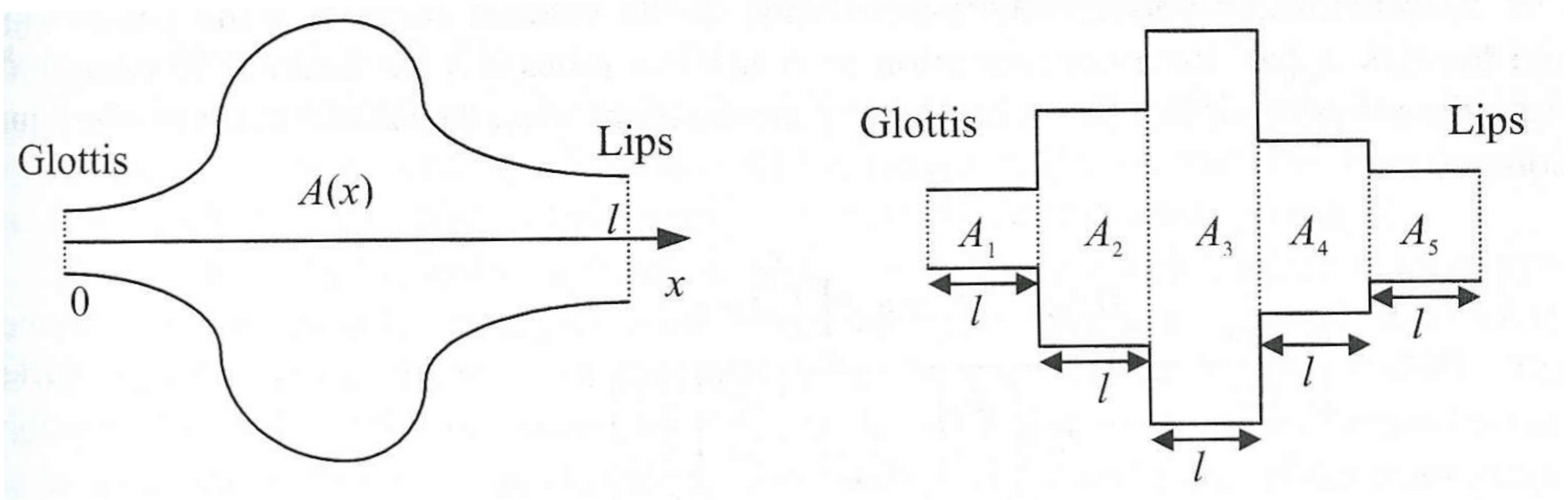
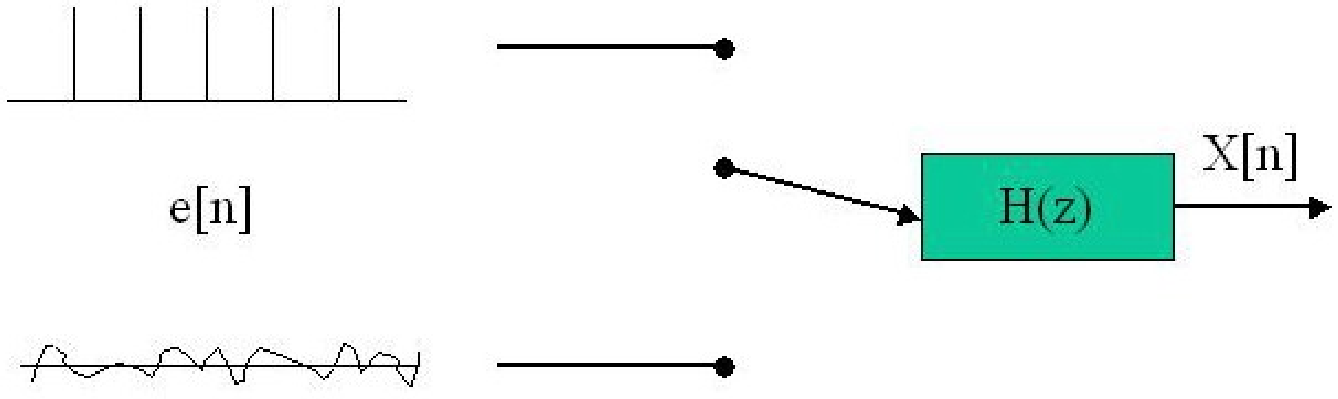


Figure 6.9 Approximation of a tube with continuously varying area $A(x)$ as a concatenation of 5 lossless acoustic tubes.

The above model of the vocal tract matches observed data quite well, at least for speech signals recorded in clean environments. It can be shown that associated the above vocal tract model can be associated with a filter $H(z)$ with a particularly simple time-domain interpretation.

Linear Prediction



The linear prediction model assumes that $x[n]$ is a linear combination of the p previous samples and an excitation $e[n]$

$$x[n] = \sum_{j=1}^{p} a[j]x[n-j] + Ge[n]$$

$e[n]$ is either a string of (unit) impulses spaced at the fundamental frequency (pitch) for voiced sounds such as vowels or (unit) white noise for unvoiced sounds such as fricatives.

Taking the Z-transform,

$$X(z) = E(z)H(z) = E(z)\frac{G}{1 - \sum_{j=1}^{p} a[j]z^{-j}}$$

where $H(z)$ can be associated with the (time-varying) filter associated with the vocal tract and an overall gain G.

Solving the Linear Prediction Equations

It seems reasonable to find the set of $a[j]$s that minimize the prediction error

$$\sum_{n=-\infty}^{\infty} (x[n] - \sum_{j=1}^{p} a[j]x[n-j])^2$$

If we take derivatives with respect to each $a[i]$ in the above equation and set the results equal to zero we get a set of p equations indexed by i:

$$\sum_{j=1}^{p} a[j]R(i,j) = R(i,0), 1 \leq i \leq p$$

where $R(i,j) = \sum_n x[n-i]x[n-j]$.

In practice, we would not use the potentially infinite signal $x[n]$ but the individual windowed frames $x^m[n]$. Since $x^m[n]$ is zero outside the window, $R(i,j) = R(j,i) = R(|i-j|)$ where $R(i)$ is just the *autocorrelation* sequence corresponding to $x^m(n)$. This allows us to write the previous equation as

$$\sum_{j=1}^{p} a[j]R(|i-j|) = R(i), 1 \leq i \leq p$$

a much simpler and regular form.

The Levinson-Durbin Recursion

The previous set of linear equations (actually, the matrix associated with the equations) is called *Toeplitz* and can easily be solved using the “Levinson-Durbin recursion” as follows:

Initialization $E^0 = R(0)$
Iteration. For $i = 1, \ldots, p$ do

$$\begin{aligned}
k[i] &= (R(i) - \sum_{j=1}^{i-1} a^{i-1}[j]R(|i-j|))/E^{i-1} \\
a^i[i] &= k[i] \\
a^i[j] &= a^{i-1}[j] - k[i]a^{i-1}[i-j], 1 \le j < i \\
E^i &= (1 - k[i]^2)E^{i-1}
\end{aligned}$$

End. $a[j] = a^p[j]$ and $G^2 = E^p$. Note this is an $O(n^2)$ algorithm rather than $O(n^3)$ and made possible by the Toeplitz structure of the matrix. One can show that the ratios of the successive vocal tract cross sectional areas, $A_{i+}/A_i = (1 - k_i)/(1 + k_i)$. The ks are called the *reflection coefficients* (inspired by transmission line theory).

LPC Examples

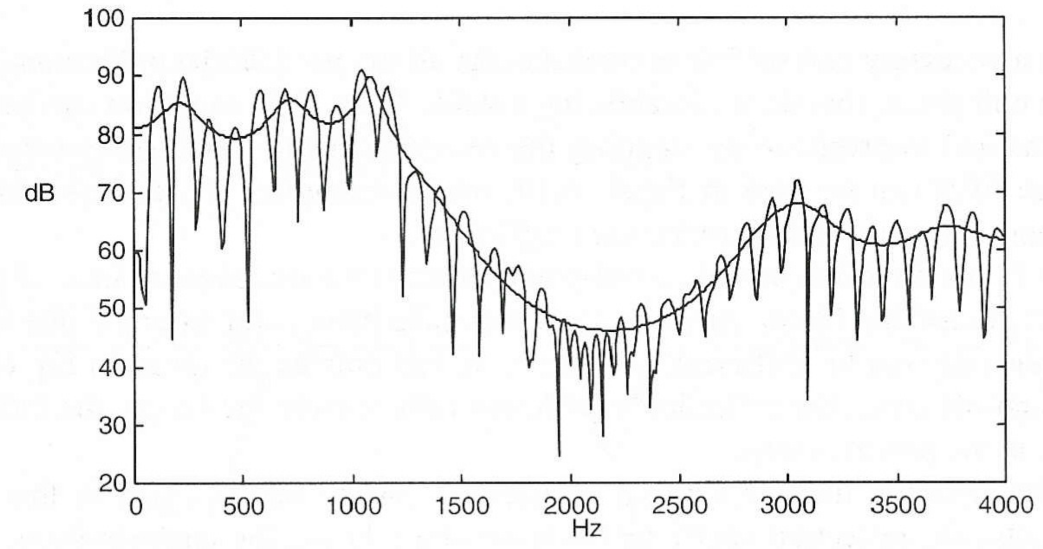


Figure 6.20 LPC spectrum of the /*ah*/ phoneme in the word *lives* of Figure 6.3. Used here are a 30-ms Hamming window and the autocorrelation method with $p = 14$. The short-time spectrum is also shown.

Here the spectra of the original sound and the LP model are compared. Note how the LP model follows the peaks and ignores the "dips" present in the actual spectrum of the signal as computed from the DFT. This is because the LPC error, $\int E(z) = X(z)/H(z)dz$ inherently forces a better match at the peaks in the spectrum than the valleys.

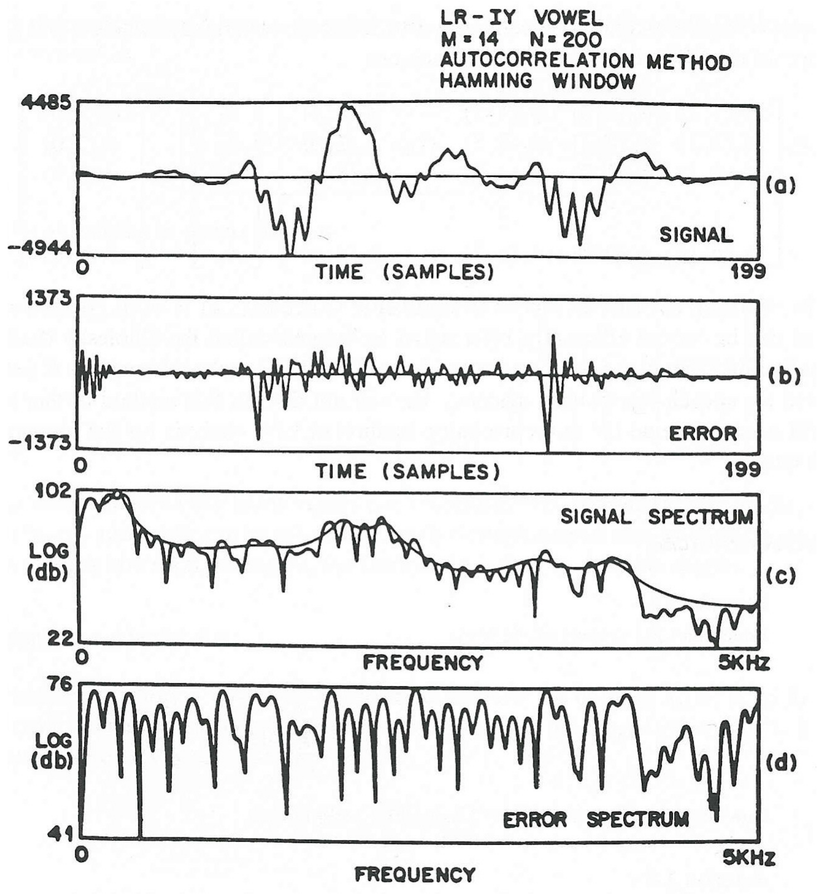


Figure 3.32 Typical signals and spectra for LPC autocorrelation method for a segment of speech spoken by a male speaker (after Rabiner et al. [8]).

Observe the prediction error. It clearly is NOT a single impulse. Also notice how the error spectrum is “whitened” relative to the original spectrum.

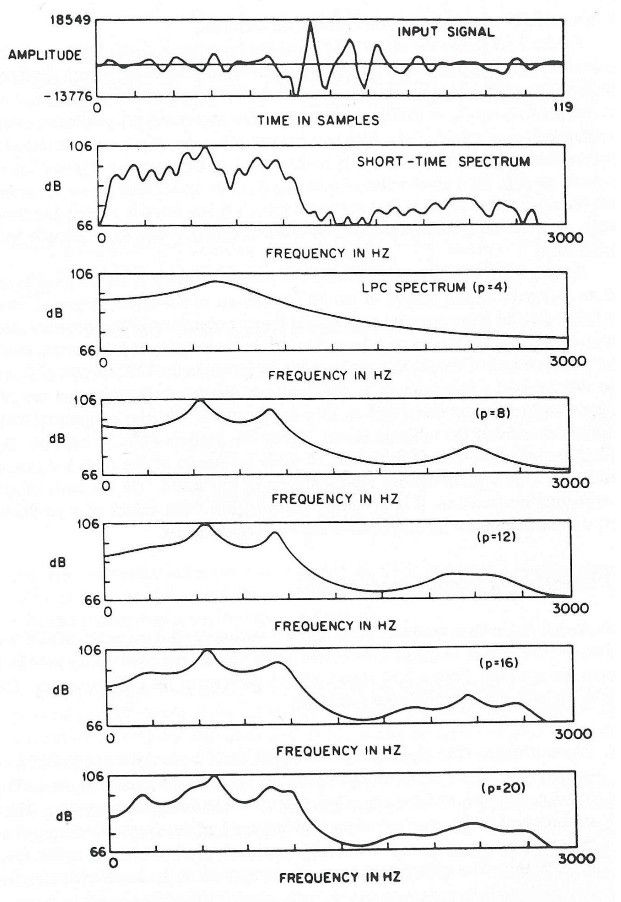


Figure 3.36 Spectra for a vowel sound for several values of predictor order, p.

As the model order p increases the LP model progressively approaches the original spectrum. (Why?) As a rule of thumb, one typically sets p to be the sampling rate (divided by 1 KHz) + 2-4, so for a 10 KHz sampling rate one would use $p = 12$ or

$p = 14$.

LPC and Speech Recognition

How should one use the LP coefficients in speech recognition?

- The $a[j]$s themselves have an enormous dynamic range, are highly intercorrelated in a nonlinear fashion, and vary substantially with small changes in the input signal frequencies.
- One can generate the spectrum from the LP coefficients but that is hardly a compact representation of the signal.
- Can use various transformations, such as the reflection coefficients $k[i]$ or the log area ratios $\log(1 - k[i])/(1 + k[i])$ or LSP parameters (yet another transformation related to the roots of the LP filter).
- The transformation that seems to work best is the LP *Cepstrum*.

LPC Cepstrum

The complex cepstrum is defined as the IDFT of the logarithm of the spectrum:

$$\tilde{h}[n] = \frac{1}{2\pi} \int \ln H(e^{j\omega}) e^{j\omega n} d\omega$$

Therefore,

$$\ln H(e^{j\omega}) = \sum \tilde{h}[n] e^{-j\omega n}$$

or equivalently

$$\ln H(z) = \sum \tilde{h}[n] z^{-n}$$

Let us assume correponding to our LPC filter is a cepstrum $\tilde{h}[n]$. If so we can write

$$\sum_{n=-\infty}^{\infty} \tilde{h}[n] z^{-n} = \ln G - \ln(1 - \sum_{j=1}^{p} a[j] z^{-j})$$

Taking the derivative of both sides with respect to z we get

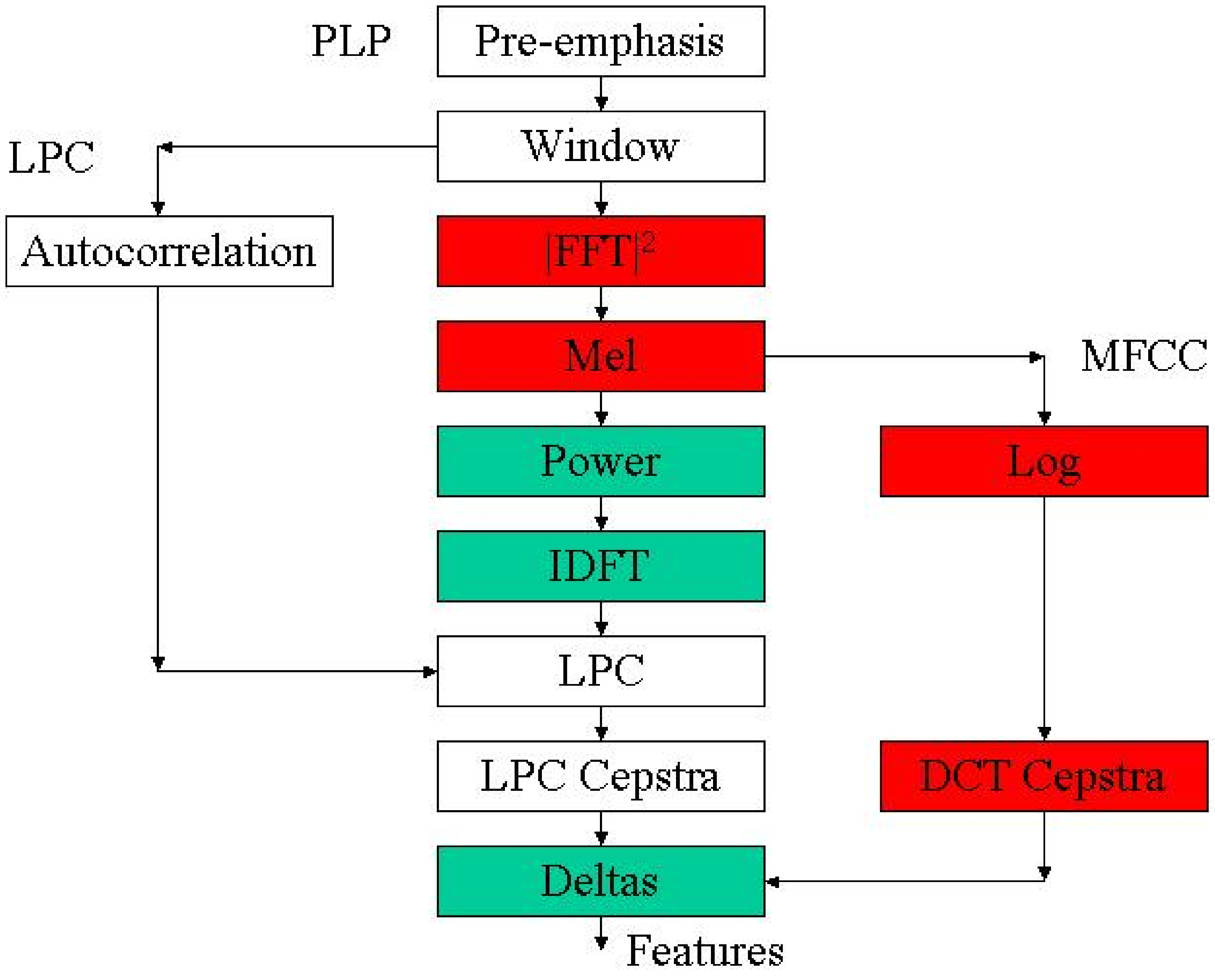
$$-\sum_{n=-\infty}^{\infty} n\tilde{h}[n]z^{-n-1} = \frac{-\sum_{l=1}^{p} la[l]z^{-l-1}}{1-\sum_{j=1}^{p} a[j]z^{-j}}$$

Multiplying both sides by $-z(1-\sum_{j=1}^{p} a[j]z^{-j})$ and equating coefficients of z we can show with some manipulations that $\tilde{h}[n]$ is

$$\begin{array}{cc} 0 & n<0 \\ \ln G & n=0 \\ a[n]+\sum_{j=1}^{n-1}\frac{j}{n}\tilde{h}[j]a[n-j] & 0<n\leq p \\ \sum_{j=n-p}^{n-1}\frac{j}{n}\tilde{h}[j]a[n-j] & n>p \end{array}$$

Notice the number of cepstrum coefficients is infinite but practically speaking 12-20 (depending upon the sampling rate and whether you are doing LPC or PLP) is adequate for speech recognition purposes.

Mel-Frequency Cepstral Coefficients



Simulating Filterbanks with the FFT

A common operation in speech recognition feature extraction is the implementation of filter banks.

The simplest technique is brute force convolution. Assuming i filters $h_i[n]$

$$x_i[n] = x[n] * h_i[n] = \sum_{m=0}^{L_i-1} h_i[m]x[n-m]$$

The computation is on the order of L_i for each filter for each output point n, which is large.

Say now $h_i[n] = h[n]e^{j\omega_i n}$, where $h(n)$ is a fixed length low pass filter heterodyned up (remember, multiplication in the time domain is the same as convolution in the frequency domain) to be centered at different frequencies. In such a case

$$\begin{aligned} x_i[n] &= \sum h[m] e^{j\omega_i m} x[n-m] \\ &= e^{j\omega_i n} \sum x[m] h[n-m] e^{-j\omega_i m} \end{aligned}$$

The last term on the right is just $X_n(e^{j\omega})$, the Fourier transform of a windowed signal evaluated at ω, where now the window is the same as the filter. So we can interpret the FFT as just the instantaneous filter outputs of a uniform filter bank whose bandwidths corresponding to each filter are the same as the main lobe width of the window.

Notice that by combining various filter bank channels we can create non-uniform filterbanks in frequency.

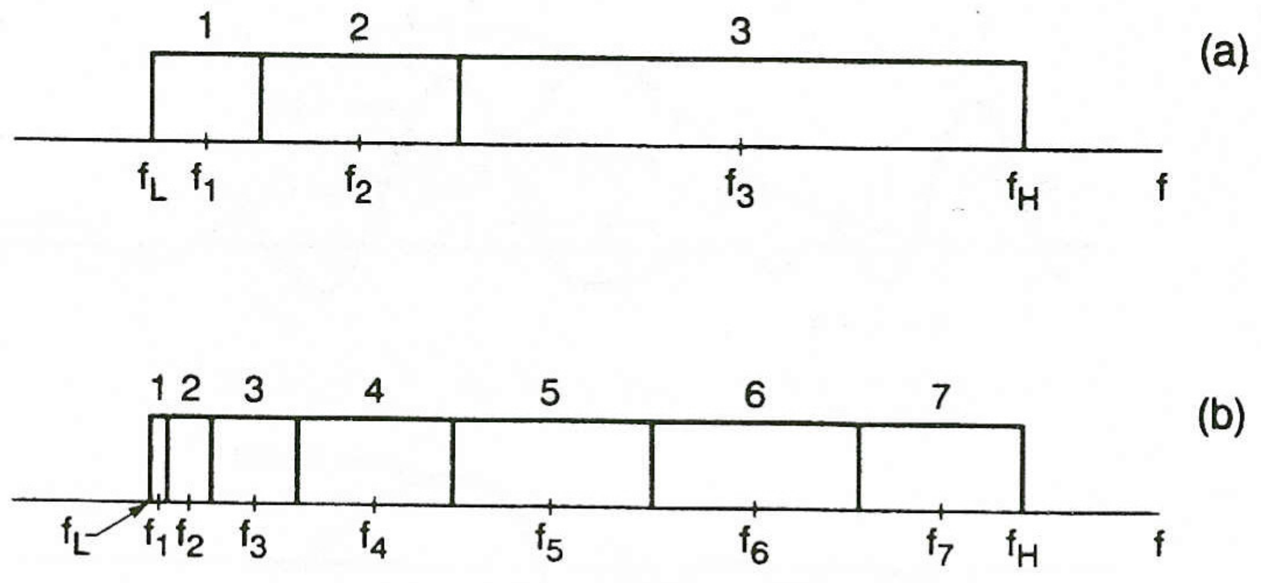


Figure 3.18 Two arbitrary nonuniform filter-bank ideal filter specifications consisting of either 3 bands (part a) or 7 bands (part b).

What is typically done in speech processing for recognition is to sum the magnitudes or energies of the FFT outputs rather than the raw FFT outputs themselves. This corresponds to a crude estimate of the magnitude/energy of the filter output over the time duration of the window and is not the filter output itself, but the terms are used interchangeably in the literature.

Mel-Frequency Cepstral Coefficients

Goal: Develop perceptually based set of features.

Divide frequency axis into m triangular filters spaced in equal perceptual increments. Each filter is defined in terms of the FFT bins k as

$$H_m(k) \begin{cases} 0 & k < f(m-1) \\ \frac{k-f(m-1)}{f(m)-f(m-1)} & f(m-1) \leq k \leq f(m) \\ \frac{f(m+1)-k}{f(m+1)-f(m)} & f(m) \leq k \leq f(m+1) \\ 0 & k > f(m+1) \end{cases}$$

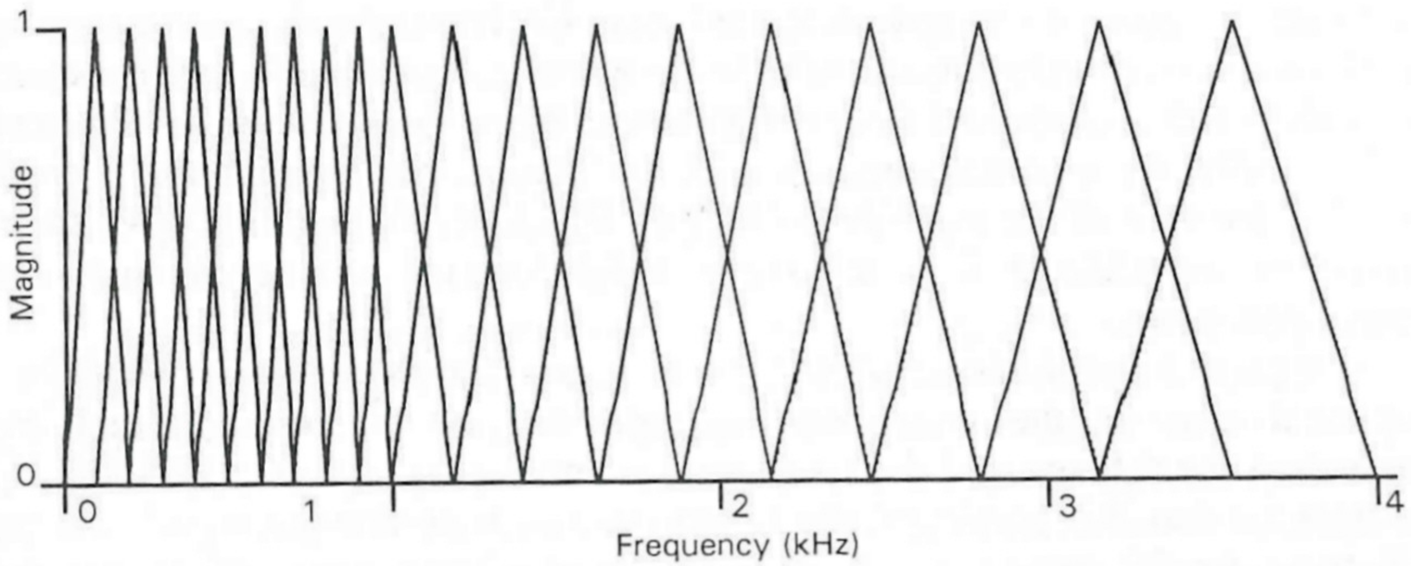


Figure 10.2 Triangular filters of the type suggested by Davis and Mermelstein (1980) for transforming the output of a Fourier transform onto a mel scale in both bandwidth and spacing.

Triangular filters are used as a very crude approximation to the shape of tuning curves of nerve fibers in the auditory system.

Define f_l and f_h to be lowest and highest frequencies of the filterbank, F_s the sampling frequency, M, the number of filters, and N the size of the FFT. The boundary points $f(m)$ are spaced in equal increments in the mel-scale:

$$f(m) = \frac{N}{F_S} B^{-1}(B(f_l) + m\frac{B(f_h) - B(f_l)}{M+1})$$

where the mel-scale, B, is given by

$$B(f) = 2595 \log_{10}(1 + f/700)$$

Some authors prefer to use $1127 \ln$ rather than $2595 \log_{10}$ but they are obviously the same thing. The filter outputs are computed as

$$S(m) = 20\, log_{10}(\sum_{k=0}^{N-1} |X_m(k)| H_m(k)), 0 < m < M$$

where $X_m(k)$ = N-Point FFT of $x^m[n]$, the mth window frame of the input signal, $x[n]$. N is chosen as the largest power of two greater than the window length; the rest of the input FFT is padded with zeros.

Mel-Cepstra

The mel-cepstrum can then be defined as the DCT of the M filter outputs

$$c[n] = \sum_{m=0}^{M-1} S(m) \cos(\pi n(m - 1/2)/M)$$

The DCT can be interpreted as the DFT of a symmetrized signal. There are many ways of creating this symmetry:

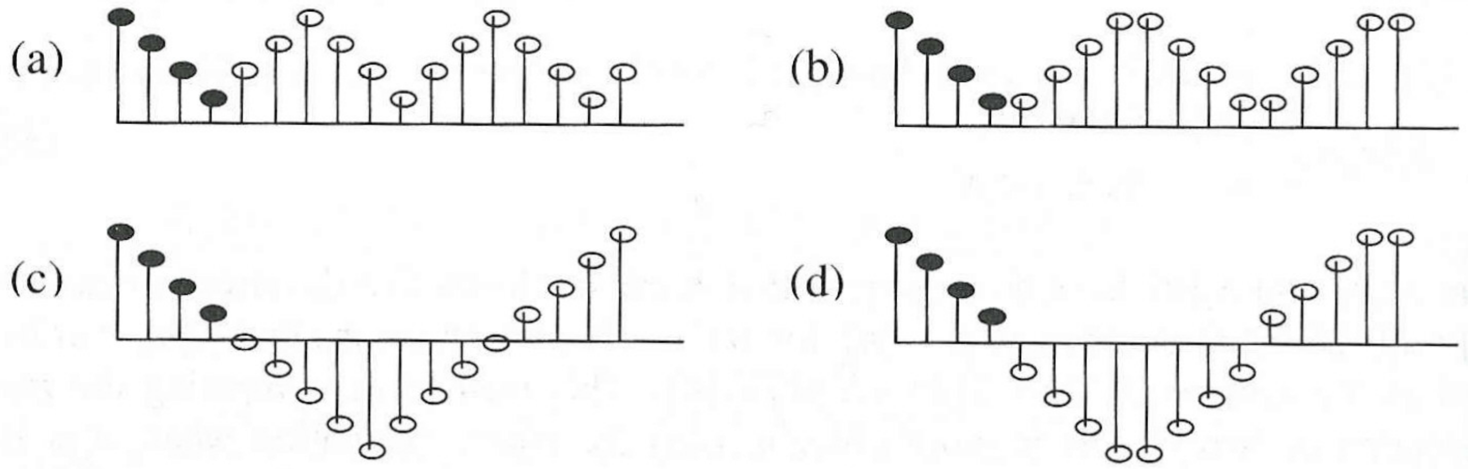
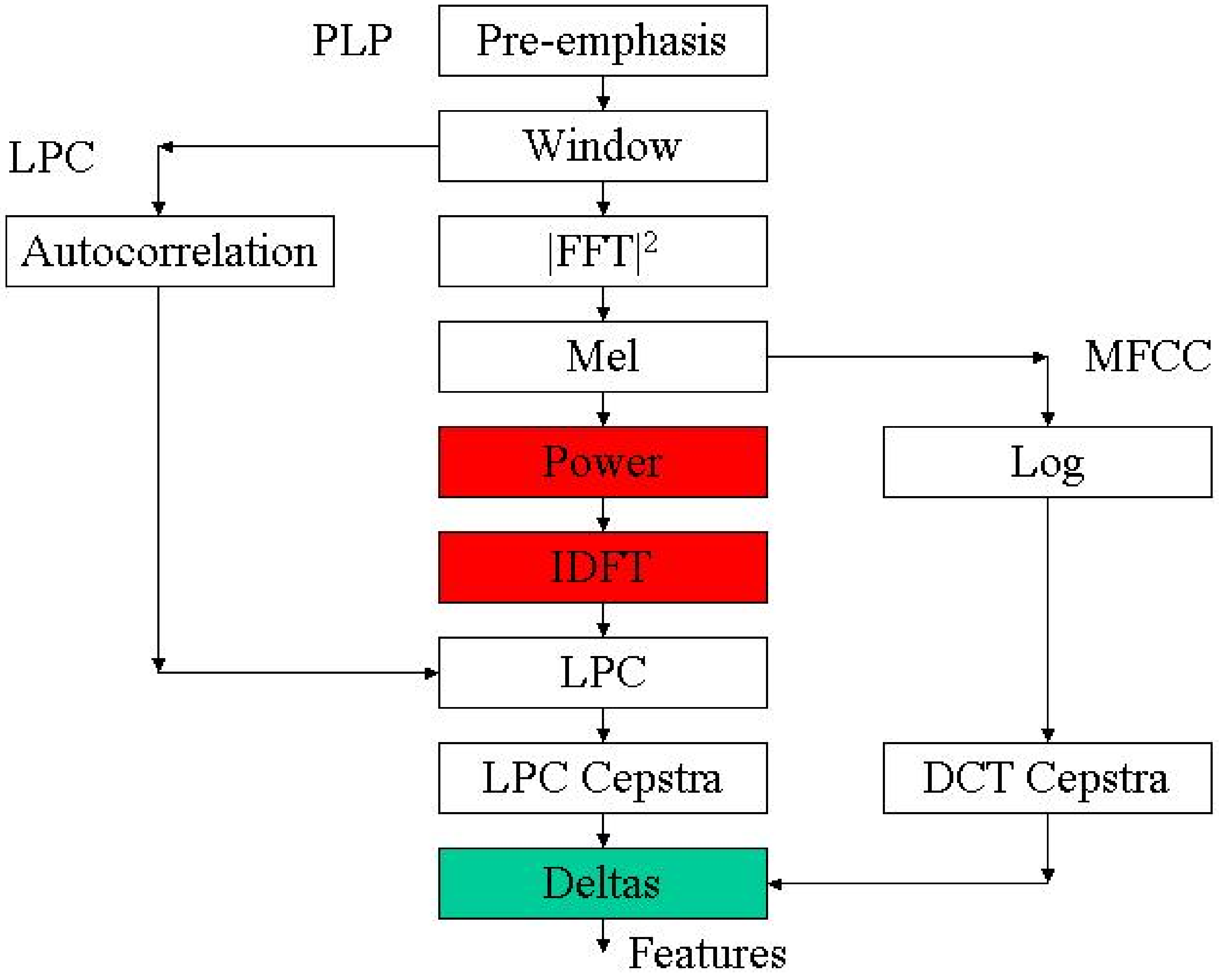


Figure 5.17 Four ways to extend a four-point sequence $x[n]$ to make it both periodic and have even symmetry. The figures in (a), (b), (c) and (d) correspond to the DCT-I, DCT-II, DCT-III and DCT-IV respectively.

The DCT-II scheme above has somewhat better *energy compaction* properties because there is less of a discontinuity at the boundary. This means energy is concentrated more at lower frequencies thus making it somewhat easier to represent the signal with fewer DCT coefficients.

Perceptual Linear Prediction



Practical Perceptual Linear Prediction [2]

Perceptual linear prediction tries to merge the best features of Linear Prediction and MFCCs.

- Smooth spectral fit that matches higher amplitude components better than lower amplitude components (LP)
- Perceptually based frequency scale (MFCCs)
- Perceptually based amplitude scale (neither)

First, the, cube root of power is taken rather than the logarithm:

$$S(m) = (\sum_{k=0}^{N-1} |X_m(k)|^2 H_m(k))^{.33}$$

Then, the IDFT of a symmetrized version of $S(m)$ is taken:

$$R(m) = \text{IDFT}([S(:), S(M-1:-1:2)])$$

This symmetrization ensures the result of the IDFT is real (the IDFT of a symmetric function is real).

We can now pretend that $R(m)$ are the autocorrelation coefficients of a genuine signal and compute LPC coefficients and cepstra as in “normal” LPC processing.

Deltas and Double Deltas

Dynamic characteristics of sounds often convey significant information

- Stop closures and releases
- Formant transitions

One approach is to directly model the trajectories of features. What is the problem with this?

Bright idea: augment normal "static" feature vector with dynamic features (first and second derivatives of the parameters). If y_t is the feature vector at time t, then compute

$$\Delta y_t = y_{t+D} - y_{t-D}$$

and create a new feature vector

$$y'_t = (y_t, \Delta y_t)$$

D is typically set to one or two frames. It is truly amazing that this relatively simple “hack” actually works quite well. Significant improvements in recognition performance accrue.

A more robust measure of the time course of the parameter can be computed using linear regression to estimate the derivatives. A good five point derivative estimate is given by:

$$\Delta y_t = \sum_{\tau=1}^{D} \tau \frac{(y_{t+\tau} - y_{t-\tau})}{2 \sum_{\tau=1}^{D} \tau^2}$$

The above process can be iterated to compute a set of second-order time derivatives, called “delta-delta” parameters., and augmented to the static and delta parameters, above.

What Feature Representation Works Best?

The literature on front ends, for reasons mentioned earlier in the talk, is weak. A good early paper by Davis and Mermelstein [1] is frequently cited.

Simple Framework:

- 52 different CVC words
- 2 (!) male speakers
- 169 tokens
- Excised from simple sentences
- 676 tokens in all

Compared following parameters:

- MFCC
- LFCC

- LPCC
- LPC+Itakura metric
- LPC Reflection coefficients

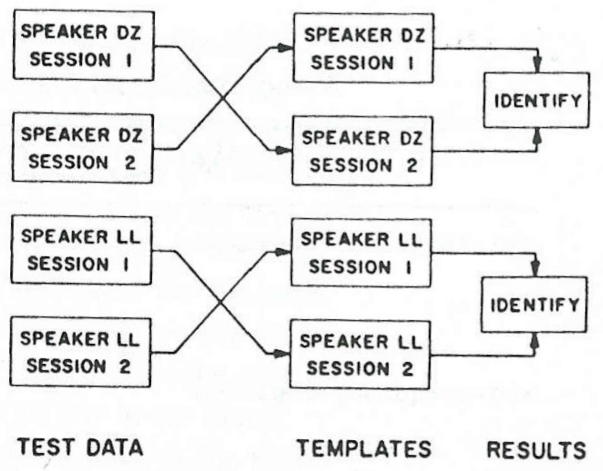


Fig. 7. Two-way speaker-dependent identification tests.

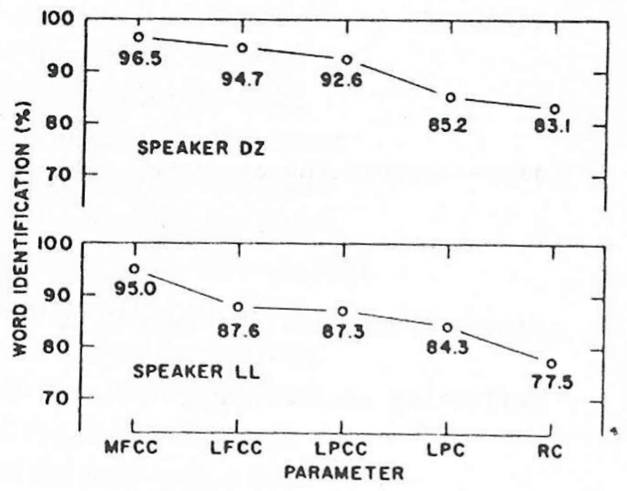


Fig. 8. Performance of parametric representations for recognition.

They also found that a frame rate of 6.4 msec works slightly better than a 12.8 msec rate, but the computation cost goes up substantially.

Other results tend to be anecdotal. For example, evidence for the value of adding delta and delta-delta parameters are buried in old DARPA proceedings, and many experiments comparing PLP and MFCC parameters are somewhat inconsistent - sometimes better, sometimes worse, depending on the task. The general consensus is PLP is slightly better, but it is always safe to stay with MFCC parameters.

Recent Developments: Tandem Models for Speech Recognition

- Idea: use Neural Network to compute features for standard speech recognition system [3]
- Train NN to classify frames into phonetic categories (e.g., phonemes)
- Derive features from NN outputs, e.g. log posteriors
- Append features to standard features (MFCC or PLP)
- Train system on extended feature vector

Some improvements (36% for new features vs 37.9% for PLP) over standard feature vector alone. May be covered in more detail in Special Topics lecture at end of semester.

What Feature Representation Works Best?

References

[1] S. Davis and P. Mermelstein, (1980) “Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences”, IEEE Trans. on Acoustics, Speech, and Signal Processing, 28(4) pp. 357-366

[2] H. Hermansky, (1990) “Perceptual Linear Predictive Analysis of Speech”, J. Acoust. Soc. Am., 87(4) pp. 1738-1752

[3] H. Hermansky, D. Ellis and S. Sharma (2000) “Tandem connectionist feature extraction for conventional HMM systems”, in Proc. ICASSP 2000, Istanbul, Turkey, June 2000.

Dynamic Time Warping - Introduction

- Simple, inexpensive way to build a recognizer
- Represent each word in vocabulary as a sequence of feature vectors, called a *template*
- Input feature vectors endpointed
- Compared against inventory of templates
- Best scoring template chosen

Two Speech Patterns

Say we have two speech patterns X and Y comprised of the feature vectors $(x_1, x_2, \ldots, x_{T_x})$ and $(y_1, y_2, \ldots, y_{T_y})$. How do we compare them? What are some of the problems and issues?

Linear Alignment

Let i_x be the time indices of X and i_y be the time indices of Y. Let $d(i_x, i_y)$ be the "distance" between frame i_x of pattern X and frame i_y of pattern Y.

In linear time normalization,

$$d(X, Y) = \sum_{i_x=1}^{T_x} d(i_x, i_y)$$

where i_x and i_y satisfy:

$$i_y = \frac{T_y}{T_x} i_x$$

One can also pre-segment the input and do linear alignment on the indvidual segments, allowing for a piecewise linear alignment.

Distances

L^p :	$\sum \lvert x-y \rvert^p$
Weighted L^p :	$\sum w \lvert x-y \rvert^p$
Itakura $d_I(X,Y)$:	$\log(a^T R_p a / G^2)$
Symmetrized Itakura :	$d_I(X,Y) + d_I(Y,X)$

Whatever you like. Note weighting can be done in advance to the feature vector components. Called “liftering” when applied to cepstra. Used for variance normalization. Also, note the L^2 metric is also called the *Euclidean* distance.

Time Warping Based Alignment

Define two warping functions:

$$\begin{aligned} i_x &= \phi_x(k) \quad k = 1, 2, \ldots, T \\ i_y &= \phi_y(k) \quad k = 1, 2, \ldots, T \end{aligned}$$

We can define a distance between X and Y as

$$d_\phi(X, Y) = \sum_{k=1}^{T} d(\phi_x(k), \phi_y(k)) m(k) / M_\phi$$

$m(k)$ is a non-negative weight and M_ϕ is a normalizing factor (Why might we need this?)

This can be seen in more detail in the following figure:

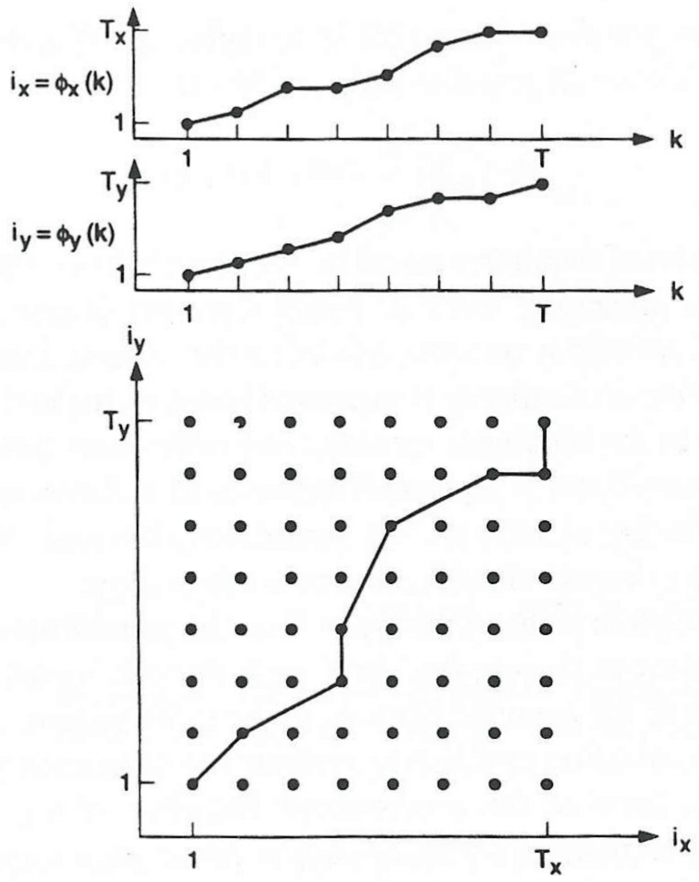


Figure 4.37 An example of time normalization of two sequential patterns to a common time index; the time warping functions ϕ_x and ϕ_y map the individual time index i_x and i_y, respectively, to the common time index k.

So the goal is to determine the two warping functions, which is basically the same as trying to determine the best path through the above grid, from the lower left corner to the top right corner.

Solution: Dynamic Programming

Definition: An algorithmic technique in which an optimization problem is solved by caching subproblem solutions (i.e., memorization) rather than recomputing them.

For example, take Fibonacci numbers.

$$\begin{aligned} f(i) &= f(i-1) + f(i-2) \text{ for } i > 1 \\ &= 1 \text{ otherwise} \end{aligned}$$

If we write a standard recursive function:

```
function fibonacci(n)
if n < 2 return 1
otherwise return (fibonacci(n-1) + fibonacci(n-2))
```

This repeats the same calculation over and over.

The alternative is:

```
fib(0,1) = 1
for i = 2 to n do
fib(n) = fib(n-1) + fib(n-2)
```

which is clearly much faster.

Why “Dynamic Programming?” [1]

“I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, Where did the name, dynamic programming, come from? The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research. Im not using the term lightly; Im using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word, “programming” I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying I thought, lets kill two birds with one stone. Lets take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is its impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. Its impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.”

Dynamic Programming: Basic Idea for Speech

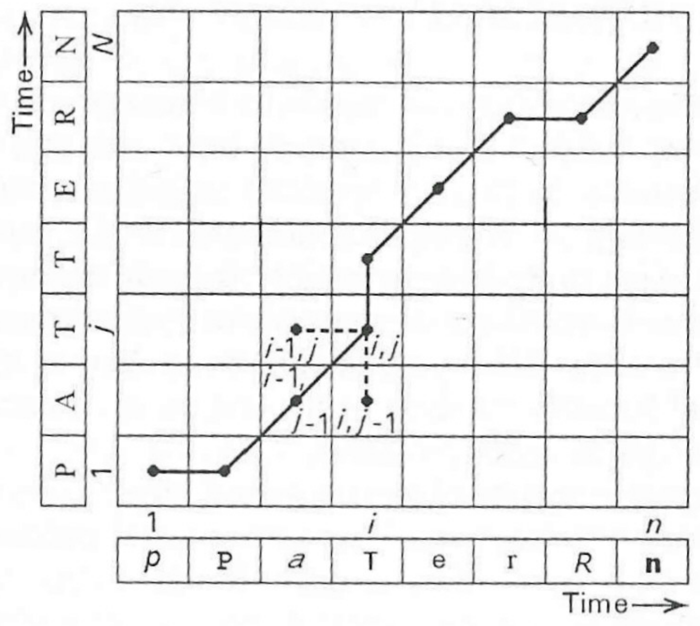


Figure 8.3 Illustration of a time-alignment path between two words that differ in their timescale. Any point i, j can have three predecessors as shown.

Let $D(i,j)$ be cumulative distance along the optimum path from the beginning of the word to the point (i,j) and let $d(i,j)$ be the distance between frame i of the input "speech" and frame j of the template. In the example, since there are only three possible ways to get to (i, j) we can write:

$$D(i,j) = \min[D(i-1,j), D(i,j-1), D(i-1,j-1)] + d(i,j)$$

All we have to do then is to proceed from column to column filling in the values of $D(i, j)$ according to the above formula until we get to the top right hand corner. The actual process for speech is only slightly more complicated.

Endpoint Constraints

Beginning Point:	$\phi_x(1) = 1$	$\phi_y(1) = 1$
Ending Point:	$\phi_x(T) = T_x$	$\phi_y(T) = T_y$

Sometimes we need to relax these conditions (Why?)

Monotonicity Constraints

$$\begin{aligned}\phi_x(k+1) &\geq \phi_x(k)\\ \phi_y(k+1) &\geq \phi_y(k)\end{aligned}$$

Why? What does equality imply?

Local Continuity Constraints

$$\begin{aligned} \phi_x(k+1) - \phi_x(k) &\leq 1 \\ \phi_y(k+1) - \phi_y(k) &\leq 1 \end{aligned}$$

Why? What does this mean?

Path Definition

One can define complex constraints on the warping paths by composing a path as a sequences of path components we will call *local paths*. One can define a local path as a sequence of incremental path changes. Define path P as a sequence of moves:

$$P \rightarrow (p_1, q_1)(p_2, q_2) \ldots (p_T, q_T)$$

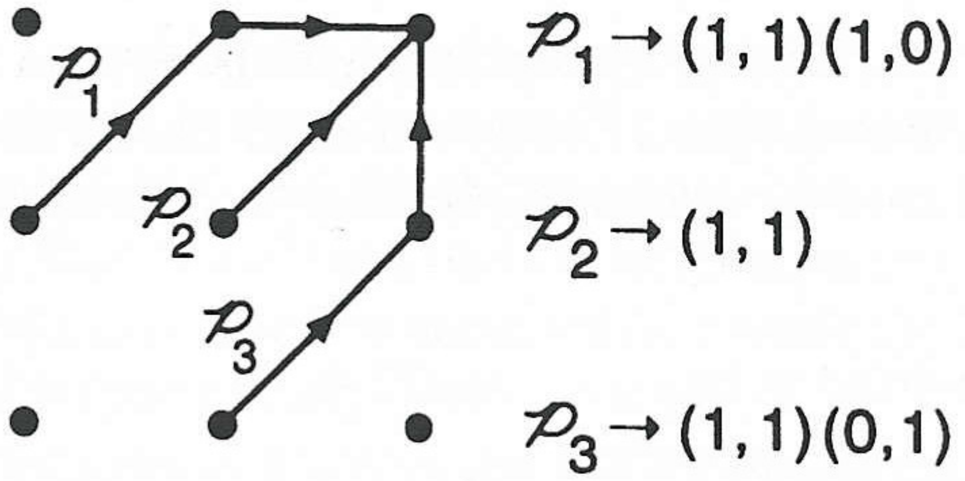

$\mathcal{P}_1$
$\mathcal{P}_2$
$\mathcal{P}_3$
$\mathcal{P}_1 \rightarrow (1,1)(1,0)$
$\mathcal{P}_2 \rightarrow (1,1)$
$\mathcal{P}_3 \rightarrow (1,1)(0,1)$

Figure 4.40 An example of local continuity constraints expressed in terms of coordinate increments (after Myers et al. [23]).

Note that

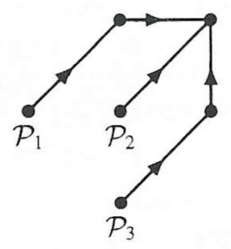
$$\phi_x(k) = \sum_{i=1}^{k} p_i \quad \phi_y(k) = \sum_{i=1}^{k} q_i$$

(with endpoint constraints)

$$T_x = \sum_{i=1}^{T} p_i \qquad T_y = \sum_{i=1}^{T} q_i$$

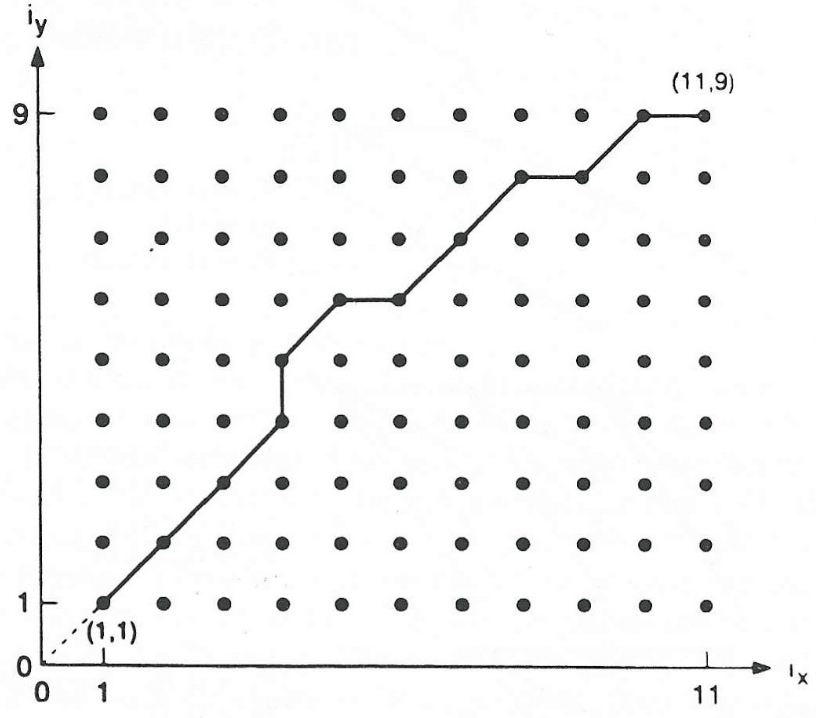
Exercise 4.5

Consider Type II local continuity constraints (see Figure 4.40 or Table 4.5).



$\mathcal{P}_1 \rightarrow (1,1)(1,0)$
$\mathcal{P}_2 \rightarrow (1,1)$
$\mathcal{P}_3 \rightarrow (1,1)(0,1)$

Find the sequence of path moves that match the sample path shown below:



Global Path Constraints

Because of local continuity constraints, certain portions of the i_x, i_y plane are excluded from the region the optimal warping path can traverse.

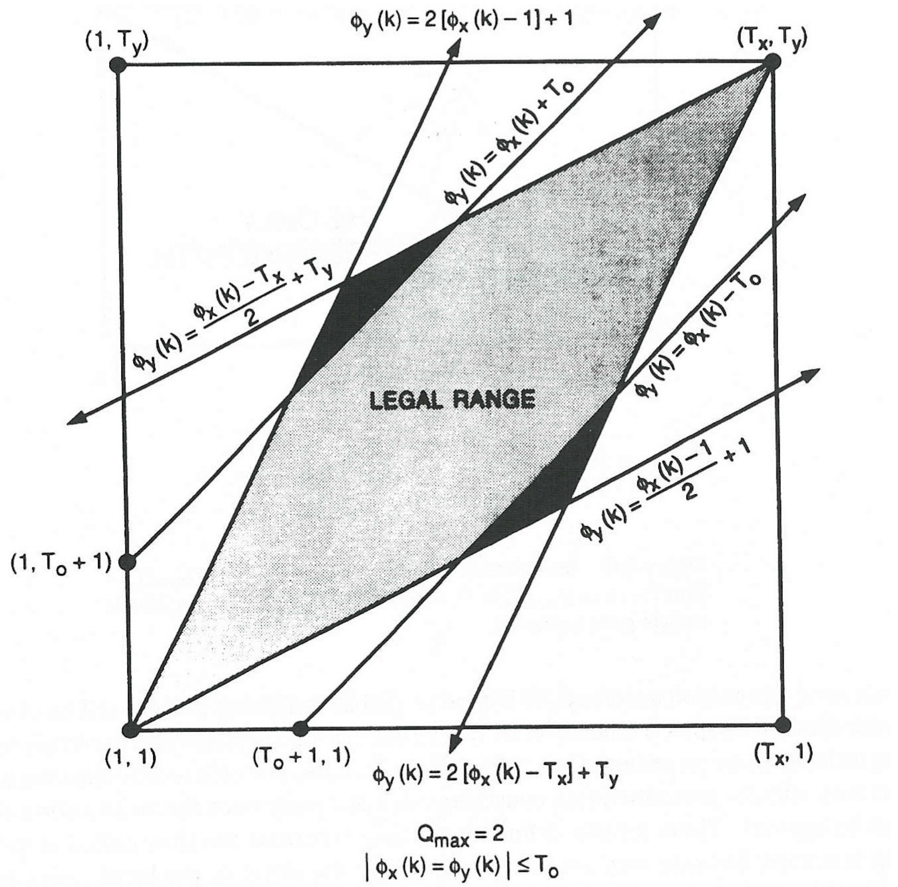

(1, T_y)
φ_y (k) = 2 [φ_x (k) − 1] + 1
(T_x, T_y)
φ_y (k) = φ_x (k) + T_o
φ_y (k) = (φ_x (k) − T_x)/2 + T_y
LEGAL RANGE
φ_y (k) = φ_x (k) − T_o
φ_y (k) = (φ_x (k) − 1)/2 + 1
(1, T_o + 1)
(1, 1)
(T_o + 1, 1)
φ_y (k) = 2 [φ_x (k) − T_x] + T_y
(T_x, 1)
Q_max = 2
| φ_x (k) = φ_y (k) | ≤ T_o

Figure 4.41 The effects of global path constraints and range limiting on the allowable regions for time warping functions.

Yet another constraint is to limit the maximum warping in time:

$$\phi_x(k) - \phi_y(k) \leq T_0$$

Note that aggressive pruning can effectively reduce a full-search $O(n^2)$ computation to a $O(n)$ computation.

Slope Weighting

The overall score of a path in dynamic programming depends on its length. To normalize for different path lengths, one can put *weights* on the individual path increments $(p_1, q_1)(p_2, q_2) \ldots (p_T, q_T)$ Many options have been suggested, such as

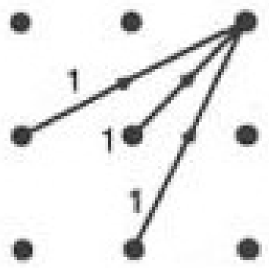
Type (a) $m(k) = \min[\phi_x(k) - \phi_x(k-1), \phi_y(k) - \phi_y(k-1)]$

Type (b) $m(k) = \max[\phi_x(k) - \phi_x(k-1), \phi_y(k) - \phi_y(k-1)]$

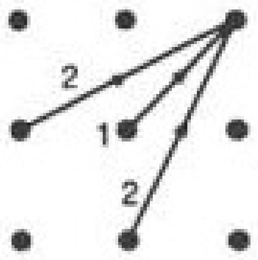
Type (c) $m(k) = \phi_x(k) - \phi_x(k-1)$

Type (d) $m(k) = \phi_y(k) - \phi_y(k-1) + \phi_x(k) - \phi_x(k-1)$

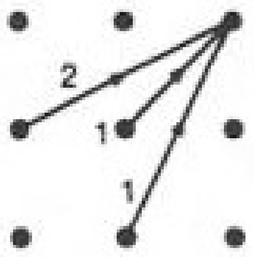
(a) $m(k) = \min[\phi_x(k) - \phi_x(k-1), \phi_y(k) - \phi_y(k-1)]$



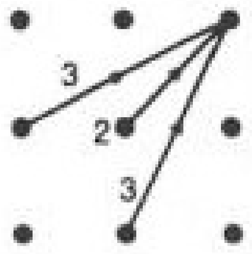
(b) $m(k) = \max[\phi_x(k) - \phi_x(k-1), \phi_y(k) - \phi_y(k-1)]$



(c) $m(k) = \phi_x(k) - \phi_x(k-1)$



(d) $m(k) = \phi_x(k) - \phi_x(k-1) + \phi_y(k) - \phi_y(k-1)$



Overall Normalization

Overall normalization is needed when one wants to have an average path distortion independent of the two patterns being compared (for example, if you wanted to compare how far apart two utterances of the word “no” are relative to how far apart two utterances of the word “antidisestablishmentarianism”). The overall normalization is computed as

$$M_\phi = \sum_{k=1}^{T} m(k)$$

Note that for type (c) constraints, M_ϕ is T_x and for type (d) constraints, M_ϕ is $T_x + T_y$. However, for types (a) and (b), the normalizing factor is a function of the actual path, a bit of a hassle. To simplify matters, for type (a) and (b) constraints, we set the normalization factor to T_x.

DTW Solution

Since we will use an M_ϕ independent of the path, we now can write the minimum cost path as

$$D(T_x, T_y) = \min_{\phi_x, \phi_y} \sum_{k=1}^{T} d(\phi_x(k), \phi_y(k)) m(k)$$

Similarly, for any intermediate point, the minimum partial accumulated cost at (i_x, i_y) is

$$D(i_x, i_y) = \min_{\phi_x, \phi_y, T'} \sum_{k=1}^{T'} d(\phi_x(k), \phi_y(k)) m(k)$$

where $\phi_x(T') = i_x$ and $\phi_y(T') = i_y$.

The dynamic programming recursion with constraints then becomes

$$D(i_x, i_y) = \min_{i'_x, i'_y} [D(i'_x, i'_y) + \zeta((i'_x, i'_y), (i_x, i_y))]$$

where ζ is the weighted accumulated cost between point (i'_x, i'_y) and point (i_x, i_y):

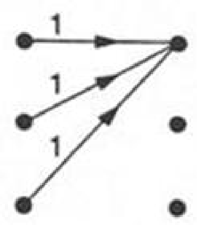
$$\zeta((i'_x, i'_y), (i_x, i_y)) = \sum_{l=0}^{L_s} d(\phi_x(T' - l), \phi_y(T' - l))m(T' - l)$$

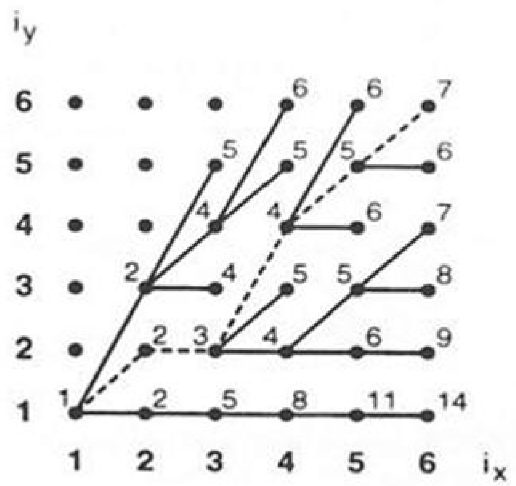
where L_s is the number of moves in the path from (i'_x, i'_y) to (i_x, i_y) according to ϕ_x and ϕ_y.

So ζ is only evaluated over the allowable paths as defined by the chosen continuity constraints for efficient implementation of the dynamic programming algorithm.

DTW Example

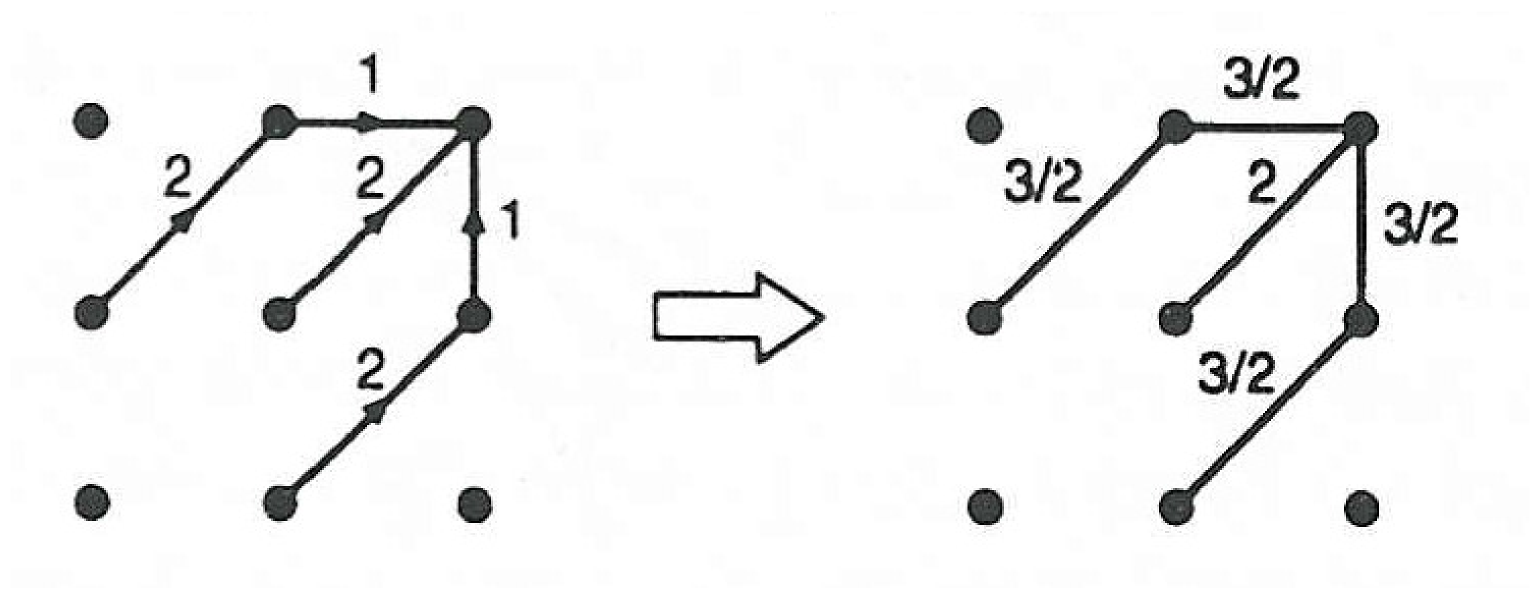
i_y						
6	●3	●2	●3	●2	●2	●2
5	●3	●2	●3	●1	●1	●1
4	●2	●2	●2	●1	●2	●2
3	●2	●1	●2	●2	●1	●3
2	●1	●1	●1	●1	●2	●3
1	●1	●1	●3	●3	●3	●3
	1	2	3	4	5	6 i_x





Additional DTW Comments

Although there are many continuity constraints and slope weightings, the following seems to produce the best performance:



The version on the left was evaluated originally by Sakoe and Chiba [2] but R+J claim that distributing the weights in a smooth fashion produces better performance (right).

Other Comments

Multiple utterances may be employed for additional robustness. Speaker dependently, this is done as follows:

- Speak one utterance
- Speak second utterance
- Align second vs. first utterance. If close, average samples along best path.
- If not close, ask for third utterance, compare to first two, and find best pair.

Multiple templates may be employed when performing speaker independent recognition. Samples from multiple speakers can be clustered to a small number of templates using a variation of the pervious algorithm.

It is also possible to extend this algorithm to connected speech,

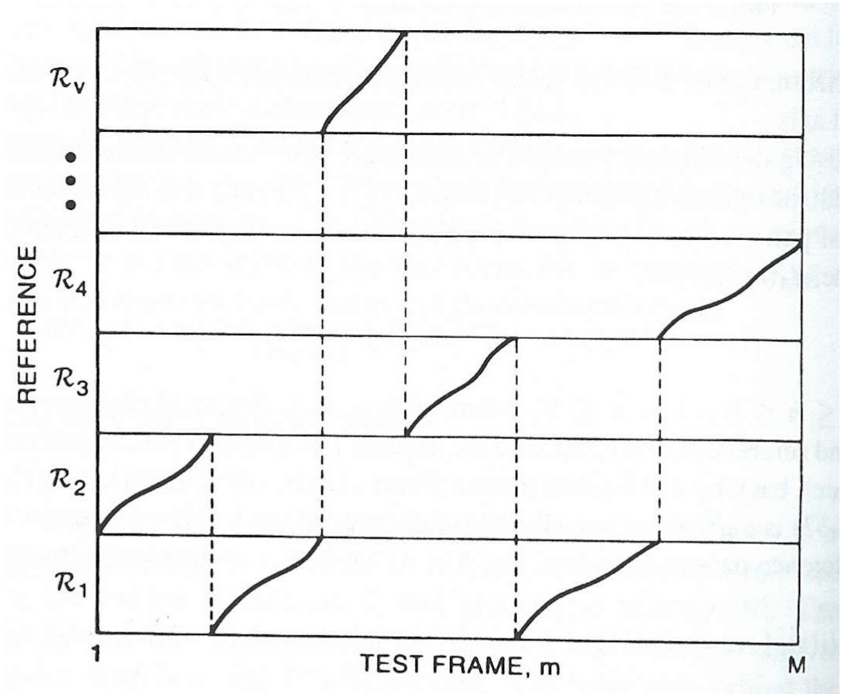


Figure 7.20 The one-pass connected word recognition algorithm (after Bridle et al. [6]).

References

[1] R. Bellman (1984)“Eye of the Hurricane”. World Scientific Publishing Company, Singapore.

[2] H. Sakoe and S. Chiba (1978)”Dynamic Programming

Algorithm Optimization for Spoken Word Recognition", IEEE Trans. on Acoustics Speech and Signal Processing vol. ASSP-26 pp 43-45, Feb.

COURSE FEEDBACK

- Was this lecture mostly clear or unclear? What was the muddiest topic?
- Other feedback (pace, content, atmosphere)?