

Introduction to Speech Synthesis

Raul Fernandez

`fernandra@us.ibm.com`


IBM Research, Yorktown Heights

Outline

➤ Introduction and Motivation

- General Speech Synthesis Architectures
- Front End: Text Analysis
- Back End Architecture (I): Concatenative Unit-Selection Synthesis
- Back End Architectures (II): Parametric Synthesis

Motivation

- First impressions are hard to overcome: 
- Quality is text-dependent and speaker-dependent
- When is a good idea to use TTS?
 - Content is dynamic (changes all the time – impossible to pre-record).
 - ✓ E.g., Account information; dialog systems; Watson.
 - Content is fixed but very large (unfeasible to pre-record).
 - ✓ E.g., Navigation assistance (large database of US street names)

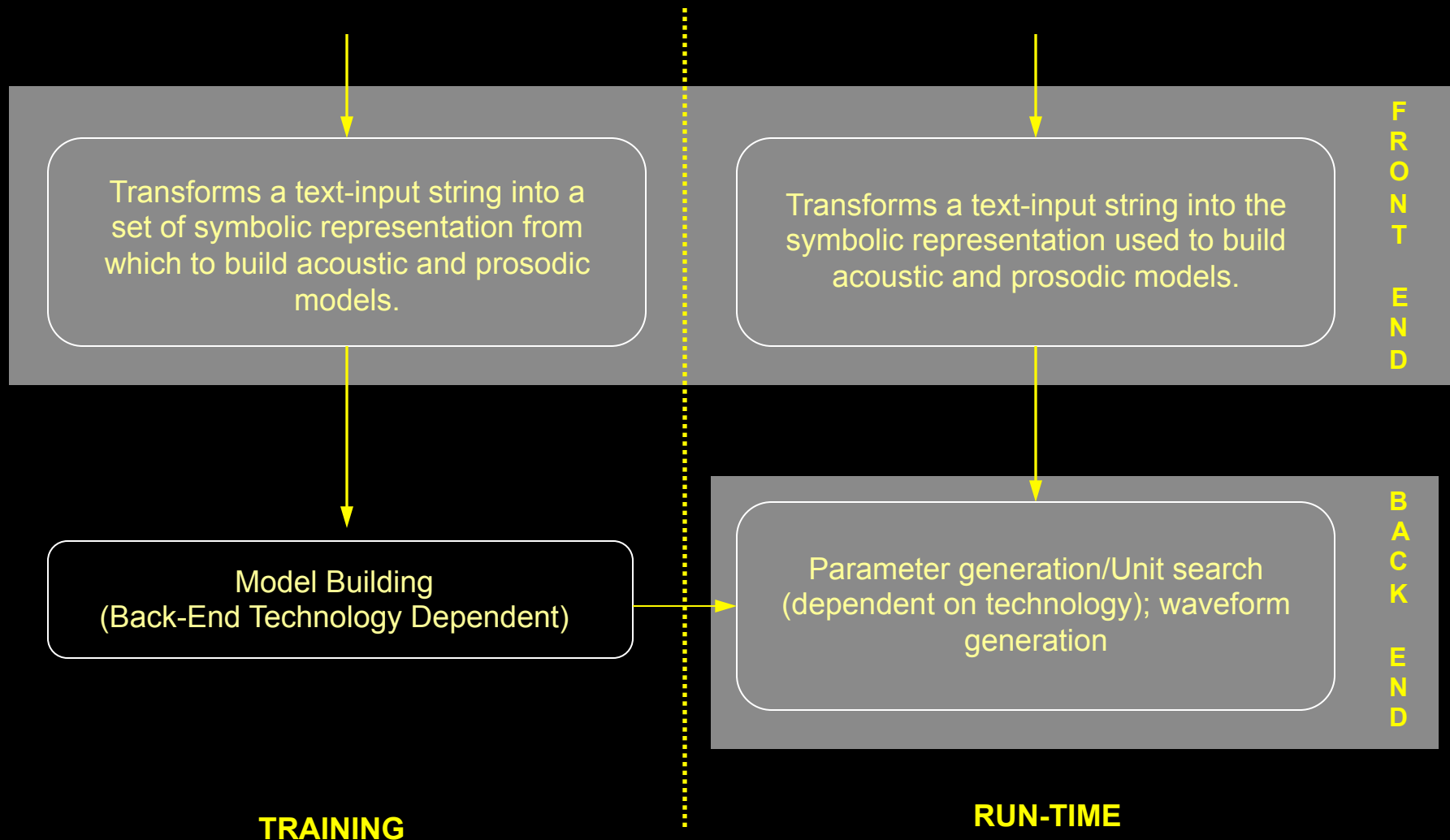
Outline

✓ Introduction and Motivation

➤ General Speech Synthesis Architectures

- Front End: Text Analysis
- Back End Architecture (I): Concatenative Unit-Selection Synthesis
- Back End Architectures (II): Parametric Synthesis

High-Level TTS Architecture



Modern Approaches to TTS Back-Ends: Concatenative

- Record actor reading large script
- Divide resulting recorded speech database into very small units
- At runtime, choose best sequence of units and splice them together to create new words
 - + High naturalness and intelligibility since units are actual speech
 - Large footprint
 - Can suffer from audible discontinuities when two units don't join well.
 - Inflexible: Hard to extend without increasing size of unit-selection database.

Modern Approaches to TTS Back-Ends: Parametric

- Record actor reading a script
 - Select a (high-quality) parametric analysis/synthesis model
 - Train a generative model (e.g., HMM) to learn the distributions of the parameters (acoustic, prosodic).
 - At runtime, generate a sequence of parameters using the model, and reproduce the speech from these parameters.
-
- + Small footprint
 - + Flexible: One can use adaptation techniques to change the “quality” of the training corpus (to alter its expression, give it an accent, etc.)
 - + Oversmoothing: Statistical averaging tends to oversmooth the spectral and prosodic characteristics of speech → Jarring discontinuities are rare.
 - Oversmoothing: Statistical averaging tends to oversmooth the spectral and prosodic characteristics of speech → Sounds flat, monotone.
 - Doesn't quite sound like actual natural speech (vocoded quality of the parametrization)

Outline

- ✓ Introduction and Motivation
- ✓ General Speech Synthesis Architectures
- Front End: Text Analysis
 - Back End Architecture (I): Concatenative Unit-Selection Synthesis
 - Back End Architectures (II): Parametric Synthesis

Front-End

- Goal is to take as input a string of text and transform it into a symbolic description (i.e., contextual features) that can be exploited at model-building time and run-time.
- Language dependent (though some sophisticated front-ends accommodate multi-lingual processing for handling mixed-language inputs).
- Typically independent of back-en technology used.

Front-End Functionality (I)

- Text cleaning: Get rid of items (HTML mark-up, etc.) that are not to be synthesized. It's often language-independent.
- Text normalization: transforms items such as *dates, time, numbers, currency, phone numbers, addresses, and abbreviations* into normal orthographic form.

Examples { *Dr. King Dr.* becomes *Doctor King Drive*
1 oz. becomes *one ounce*
2 oz. becomes *two ounces*

- Grapheme-to-Phoneme Conversion (Phonetization): Transforms a (normalized) orthographic string into the “phones” of the language:

The brown fox → DH AX B R AW N F AO K S

Front-End Functionality (II)

- Syllabification and lexical stress prediction: Divides a word's phonetic representation into syllables, and marks lexical stress:

Speech Synthesis → S P IY (1) CH II S IH (1) N I TH AX (0) I S IX (0) S

- Part-of-Speech Tagging:

She came to record the record. → She (PRN) came (VB) to record (VB) the (DET) record (NOUN).

- Syntactical Analysis

[NP The brown fox] [VP jumped] [PP over] [NP the lazy dog.]

- Semantic Analysis such as named-entity recognition (is it a person? a place? an organization? a quantity? etc.)

Jim bought 300 shares of Acme Corp. in 2006. → <ENAMEX TYPE="PERSON">Jim</ENAMEX> bought <NUMEX TYPE="QUANTITY">300</NUMEX> shares of <ENAMEX TYPE="ORGANIZATION">Acme Corp.</ENAMEX> in <TIMEX TYPE="DATE">2006</TIMEX>.

Front-End Implementations

- Usually modular architectures. Implementation depends on the nature of the module (phonetizer, parser, etc.) and language
 - Rules-based:
 - Hand-crafted rules
 - Amenable to encoding exceptions that aren't captured by patterns (e.g., English letter-to-sound rules).
 - Hard to maintain
 - Brittle (rules may interact in unpredictable ways)
 - Statistical / Data-driven:
 - Flexible (easy to retrain given newly available training data)
 - Requires no expert linguistic knowledge
 - Capture statistical tendencies and therefore tend to perform poorly when handling exceptions.
 - Look-Up:
 - Most front ends provide a mechanism for bypassing the processing (rules-based or statistical) and resort to direct look-up.
 - E.g., Exception dictionaries to reflect hard-to-predict pronunciations of names in an ever-growing customer's database.

Outline

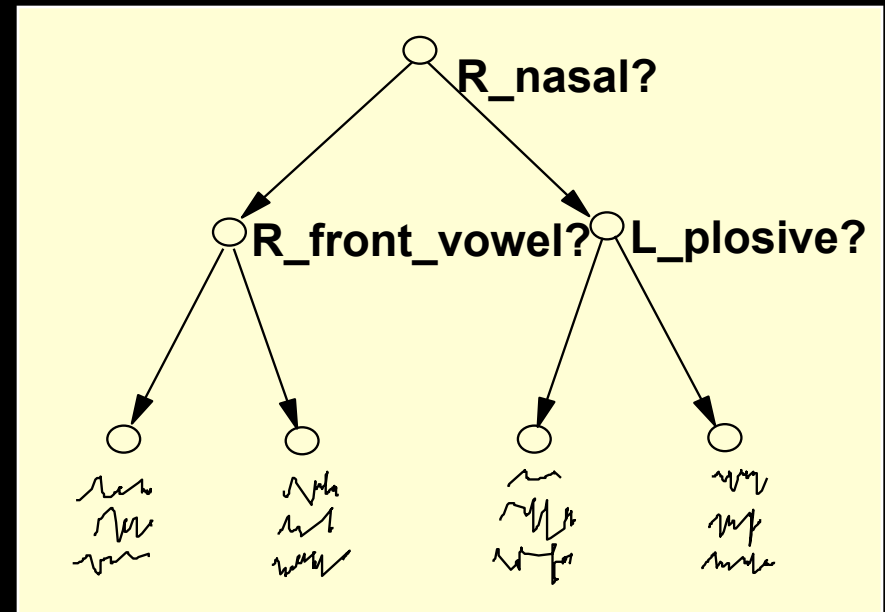
- ✓ Introduction and Motivation
- ✓ General Speech Synthesis Architectures
- ✓ Front End: Text Analysis
- Back End Architecture (I): Concatenative Unit-Selection Synthesis
- Back End Architectures (II): Parametric Synthesis

Typical Concatenative Back-End Architecture

- Define a unit size (e.g., $\sim 1/3$ phones). Automatically align and pre-segment a recorded voice database into the basic building units.
- Acoustic Candidate Generation: Build a model that predicts which acoustic waveform units are needed/relevant to synthesize a given context.
- Prosody Target Generation: Build a model that predicts which prosody (pitch, duration, energy) the units should carry in a given context.
- Search: Given candidate units, and prosody targets, extract the optimal sequence of units according to some criterion.
- Waveform generation/modification: Join the selected units (via, e.g., PSOLA), taking care to reduce perceptual acoustic artifacts.
 - Optionally: Ensure that the resulting output waveform reflects the pitch, duration and energy values of the prosody targets.

Acoustic Model: Decision Tree for Mapping Context to Units.

- ✓ Associate every acoustic unit in the database with its symbolic context (based on the output of the front end).
- ✓ Build a tree by asking questions (i.e., making splits) based on the context of the current and neighboring units (e.g., central phone plus 2 on either side).
- ✓ Acoustic observations gathered at the resulting leaves (i.e., context-dependent units) are the basic synthesis units for an input context.
- ✓ At run-time, the symbolic context is extracted for the input text, the tree is traversed, and once a leaf is reached, we extract its members as candidates.



Brief Digression: A Word About Context

- Besides providing a mechanism for retrieving a set of relevant units for a given context, a tree inherently provides a way to handle **DATA SPARSITY!!**
- We never get to observe instances for every given context. That's a **PROBLEM** if we want to synthesize at run-time a context we didn't observe in the training data (which happens all the time, actually!).
- Consider this simple context feature vector:

<u>Feature Type</u>	<u>Cardinality</u>	<u>Total # of Combinations</u>
(1) phone identity	44	44
(2) POS	25	$44 \times 25 = 1100$
(3) # of phones from start	50	$44 \times 25 \times 50 = 55,000$
(4) Question/Statement/Excl	3	$44 \times 25 \times 50 \times 3 = 165,000$
- This simple 4-dimensional feature vector requires we observe 165K combinations. In practice we work with hundreds of symbolic features, generating hundreds of thousands of context combinations.
- Trees effectively exhaustively partition the input space into N context combinations (the N leaves of the tree). At run-time, you will reach one leaf. Even if the full run-time context vector was not observed in the training set, there will at least be a partial match.

Prosody Target Generation

- Similar idea: Build a model that generates prosody targets for a particular context combination.
- It may make sense to model different prosodic dimensions at different levels:
 - sonorant-region F0 prediction
 - phone-level log-duration prediction
 - phone-level energy prediction
- Build the tree as before. Use it at run time to generate a target. Propagate this target down to the basic unit of the back-end architecture (e.g., sub-phone).

Prosody Target Generation Example: F0

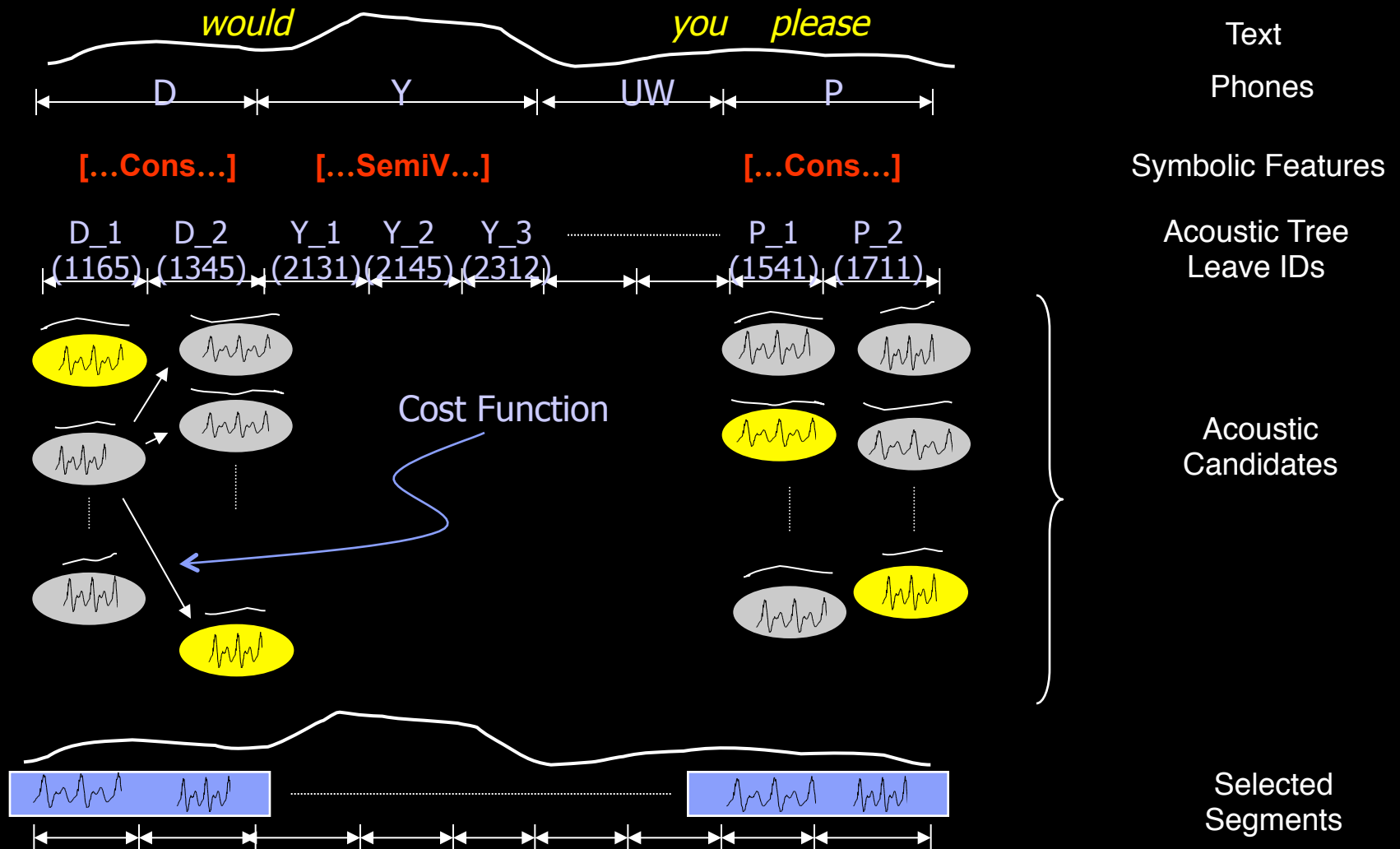
■ Syllable-based statistical decision tree predictor

- Observations: Three pitch values are predicted over the syllable's sonorant region (nucleus plus adjacent liquids and glides).
- Predictor feature set: Syllabic context of 5 syllables (2 preceding and 2 following), including
 - Position of syllable in phrase
 - Syllable lexical stress
 - Word stress (prominence)
 - Position of word in phrase
 - Phrase type (question, statement, exclamation)
 - Word part-of-sentence
 - Number of syllables in word
 - Phone identity of 5 phones (current plus previous and following 2)

**Context
Features**

Mean of observations gathered at a leaf becomes predicted F0 target for a sonorant region. A F0 target at the unit level is obtained by interpolating across these

Unit Selection: Viterbi Search



Search: Typical Cost Function

$$\begin{aligned}
 \text{Cost Function} = & \underbrace{\text{Target Cost}}_{\beta \times \text{Prosody Cost}} + \underbrace{\text{Concatenation Cost}}_{\delta \times \text{Spectral Transition} + \eta \times \text{F0 Transition}} \\
 & \underbrace{\gamma \times \text{Pitch Cost} + \zeta \times \text{Duration Cost} + \phi \times \text{Energy Cost}}
 \end{aligned}$$

More formally: $u_1^* \dots u_M^* = \arg \max C(u_1^{k_1} \dots u_M^{k_M})$

$$\text{where } C(u_1^{k_1} \dots u_M^{k_M}) = \sum_{m=1}^M C_{\text{target}}(u_m^{k_m}, \text{PsdyTg}_m) + \sum_{m=2}^M C_{\text{concat}}(u_{m-1}^{k_{m-1}}, u_m^{k_m})$$

$u_1^* \dots u_M^*$ can be found through dynamic programming

Waveform Generation

- Pitch-Synchronous Overlap-Add (PSOLA) is used to modify the pitch and duration of the units selected in the search to desired values. For instance:
 - The pitch/duration targets requested by the prosody trees
 - A smoothed version of the units' natural pitch contour
 - Etc.
- When selected synthesis units appear contiguously in the database, signal processing through these units is disabled everywhere except at the boundaries (*contiguous bypass*) to further reduce artifacts introduced by the modification and preserve the natural prosody.

Outline

- ✓ Introduction and Motivation
- ✓ General Speech Synthesis Architectures
- ✓ Front End: Text Analysis
- ✓ Back End Architecture (I): Concatenative Unit-Selection Synthesis
- Back End Architectures (II): Parametric Synthesis

Typical Parametric (HMM) Back-End Architecture

- Main Idea Recap: Instead of using waveform pieces, adopt a parametric representation of speech, learn the distribution of the parameters, generate parameters at run-time, and then reconstruct speech from them. That is, we need an “invertible” transformation (i.e., an analysis-resynthesis method).
- Use a left-to-right N-state (3 or 5 per phone, typically) topology. Each HMM state models a context-dependent distribution of the spectral parameters and F0 jointly, augmented by their delta and delta-delta features (we’ ll see why shortly!).
- Given a string of text at run time, extract the context, form a sentence-level context-dependent HMM, and use the Parameter Generation Algorithm to synthesize the parameters.
- Generate speech from synthesized parameters.

Which Underlying Parametric Representation?

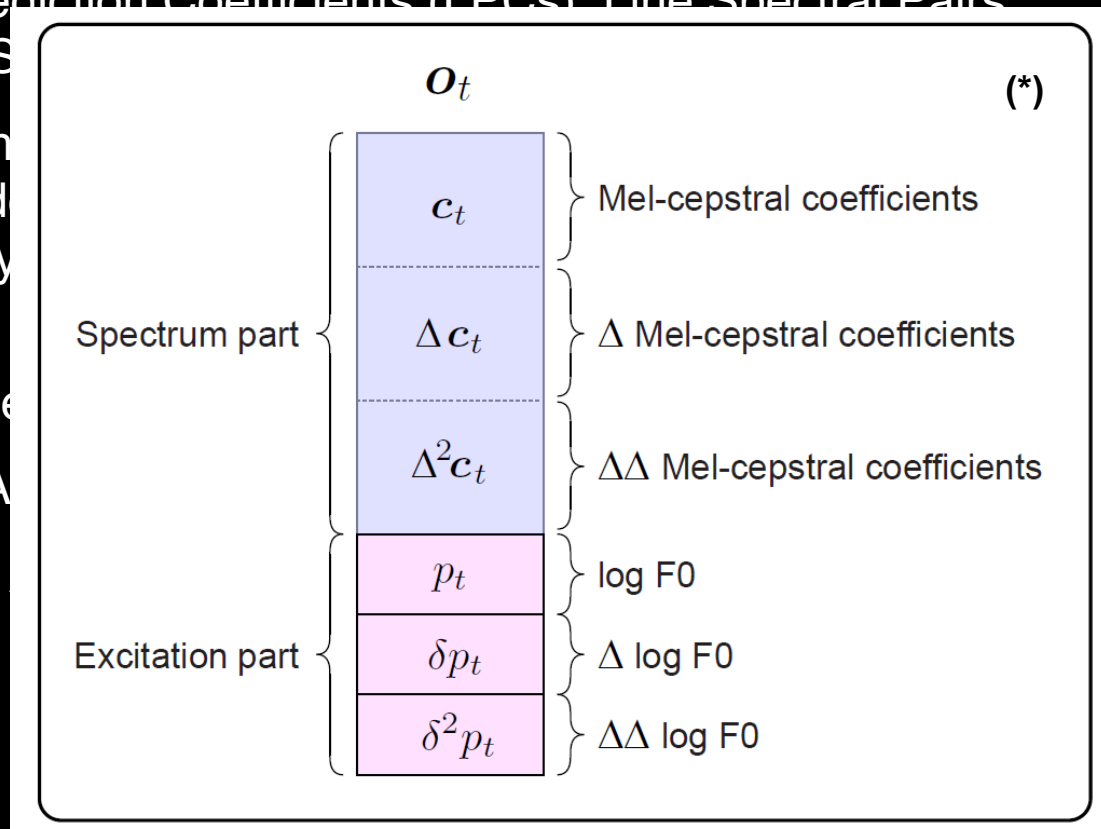
➤ There are many! Linear Prediction Coefficients (LPCs), Line Spectral Pairs (LSPs), Mel-Cepstral (MCEPS)

➤ We won't go into details (though we can represent a short-time window of relatively low dimensionality values).

➤ Let's assume we have access to

➤ Algorithm 1 (Analysis and Synthesis)

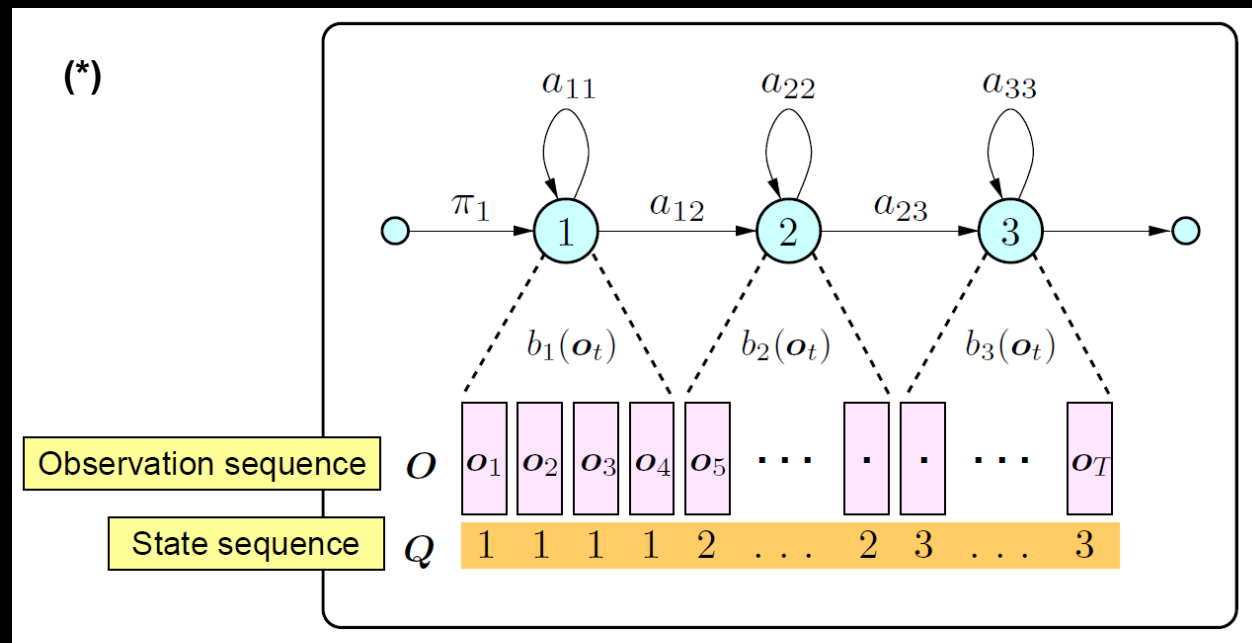
➤ Algorithm 2 (Synthesis)



(*) Tokuda & Zen (2009)

HMM Topology: 3-State Left-to-Right

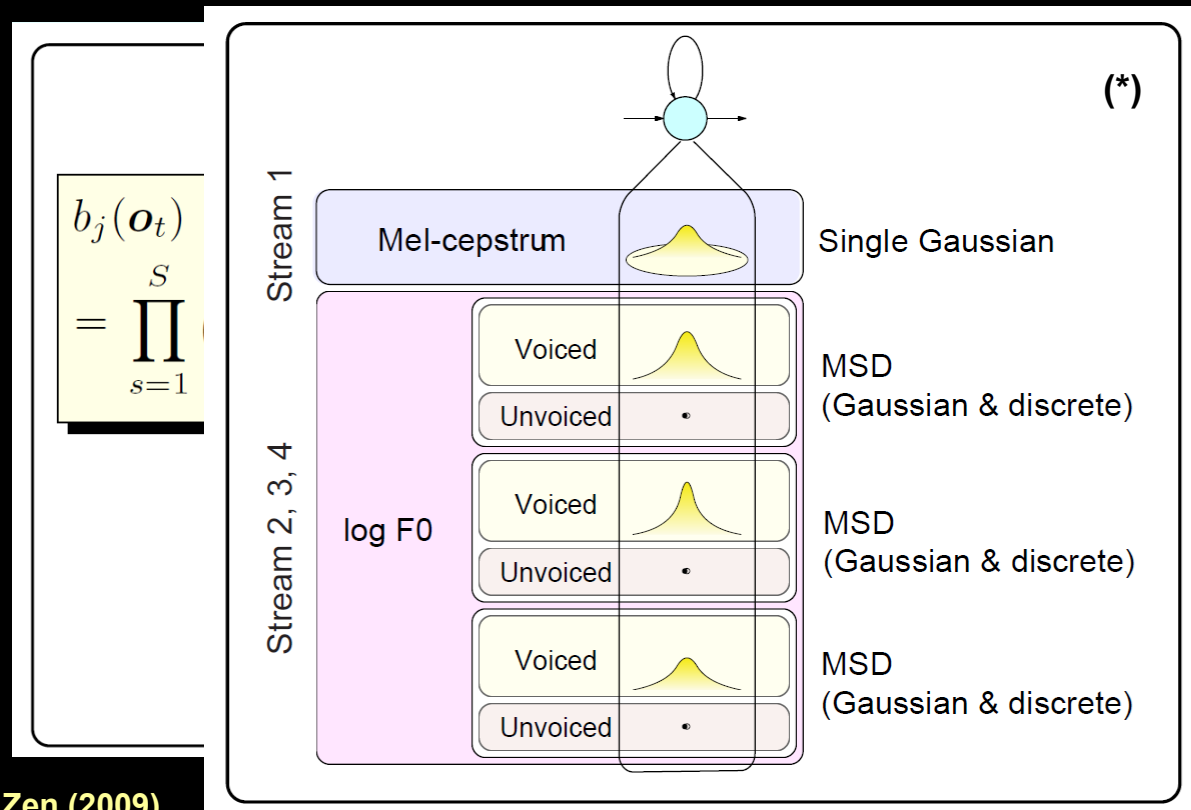
- Why left-to-right? Answer coming up in a few more slides!



(*) Tokuda & Zen (2009)

Multi-Stream HMM and Output Observations

- Represent different streams of information using different distributions.
- Global output distribution for an observation is a weighted product of each stream distribution.
- Collapse spectral parameters and their deltas into a single stream (a vector of real-valued numbers).
- Use separate stream for F0 and each of their deltas.



(*) Tokuda & Zen (2009)

HMM Training

- Distribution of spectral and F0 parameters depend not just on the phone, but on its context.
→ Intuition: Phone /AW/ looks spectrally different at the end of a word than, say, when followed by a liquid. To collapse all observations together is a rather poor modeling decision.

- Each state, therefore, is context-dependent
variant thereof.

→ No problem
possible.

- Not so fast: Same
possible contexts.

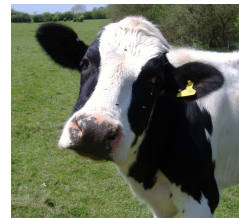
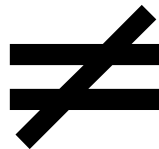
→ We know how

→ We may wish to build separate trees for the spectral and F0 streams since they're probably influenced differently by context. (Think, for instance, of how the pitch can rise at the end of a question. F0 may be more contextually sensitive to a *distance-from-end-of-sentence* feature than the spectrum is.)

- Use Baum-Welch (forward/backward) techniques to learn the HMM parameters.



/ AW L /



/ K AW /

text-dependent

ble context

data to cover all

Parameter Generation Approach

Write down the maximization problem →

$$\hat{o} = \arg \max_o p(o \mid w, \hat{\lambda})$$

$$= \arg \max_o \sum_{\forall q} p(o, q \mid w, \hat{\lambda})$$

$$\approx \arg \max_o \max_q p(o \mid q, \hat{\lambda}) p(q \mid w, \hat{\lambda})$$

Introduce a Viterbi approximation →

Decompose into TWO maximization sub-problems:

1. Find the best state sequence
2. Find the best parameter sequence given the best state sequence

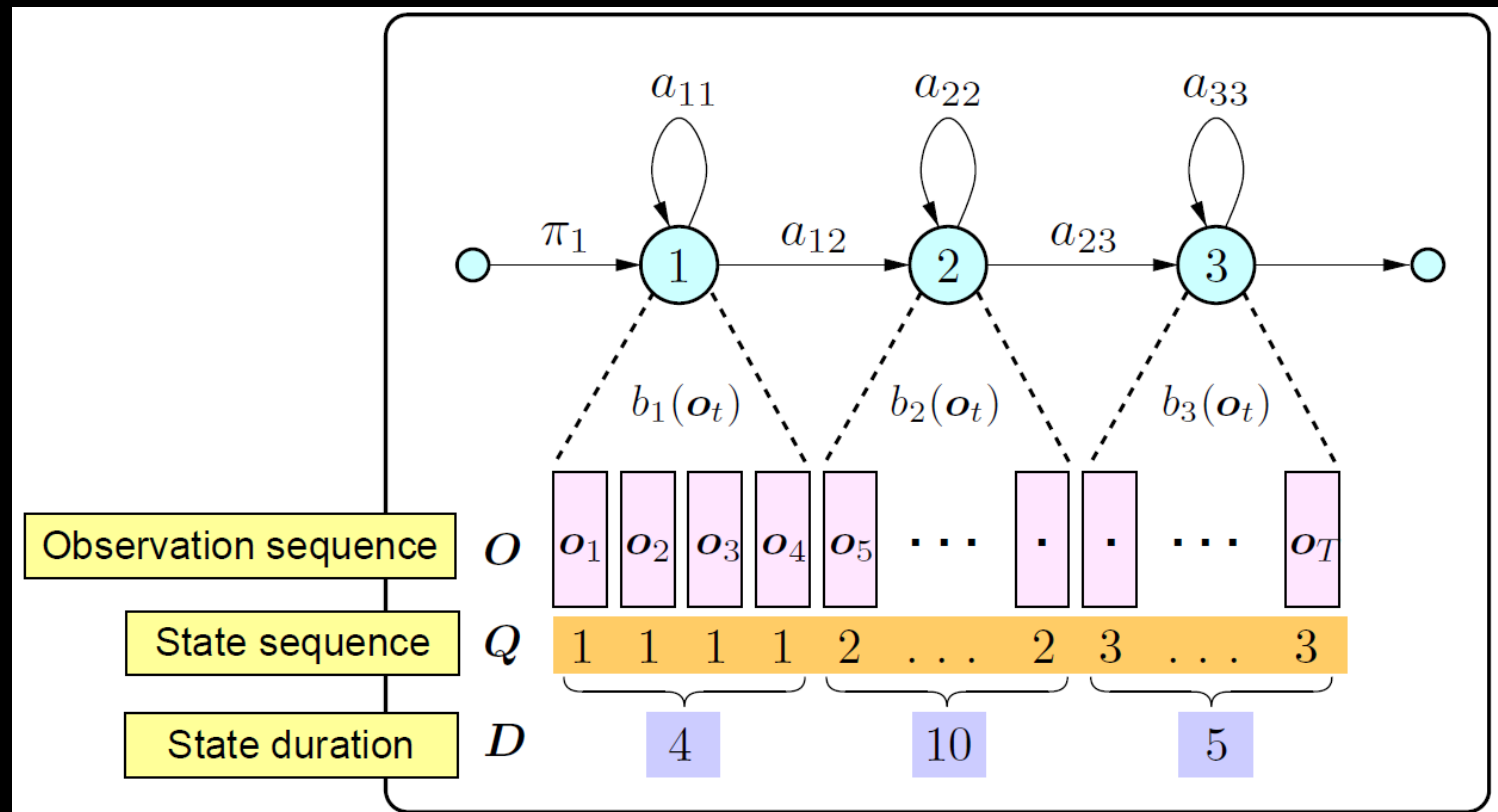


$$\hat{q} = \arg \max_q P(q \mid w, \hat{\lambda})$$

$$\hat{o} = \arg \max_o p(o \mid \hat{q}, \hat{\lambda})$$

(*) Tokuda & Zen (2009)

Problem #1: Estimating State-Sequence (I)



(*) Tokuda & Zen (2009)

- If HMM is left-to-right, there's a unique way to traverse it.
- We just don't know how long we've spent in each state (how many frames a state should emit)
- If we can estimate each state's duration, then the state sequence is uniquely defined.

Problem #1: Estimating State-Sequence (II)

Example: The probability of arbitrary state sequence 1 1 2 2 2 3 3 is:
the probability of staying in state 1 for 2 frames AND in state 2 for 3 frames AND in state 3 for 2 frames

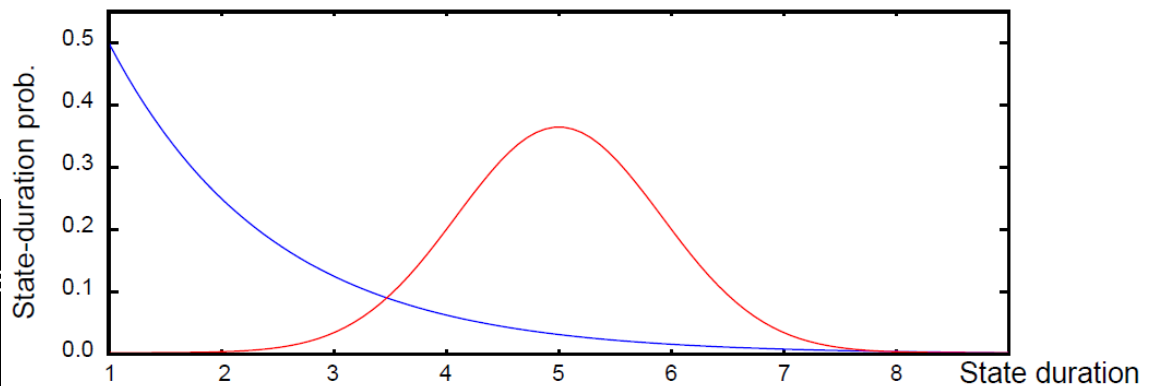
More formally:

Geometric

$$p_i(d_i) = a_{ii}^{d_i-1} (1 - a_{ii}) \Rightarrow \hat{d}_i = 1$$

Gaussian

$$p_i(d_i) = \mathcal{N}(d_i ; m_i, \sigma_i^2) \Rightarrow \hat{d}_i = m_i$$

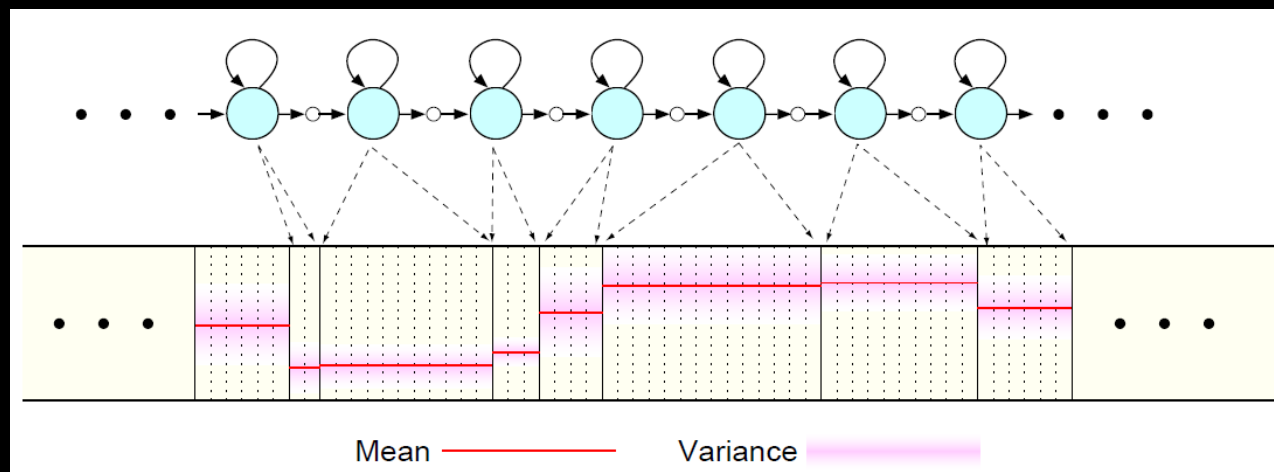


So, maximize the LHS of the above e

(*) Tokuda & Zen (2009)

Problem #2: Sequence Generation (I)

- If we model the observations directly without any constraints (such as dynamics), then, given a state sequence, the maximum-likelihood estimate of the observations is that state's mean throughout the entire state duration.
 - The ML estimates of the trajectories would be flat. Natural speech does not behave that way (or would sound good if we tried to synthesize with these parameters!).



(*) Tokuda & Zen (2009)

- Solution: Add a dynamic constraint → The delta features we added to our observation vector earlier

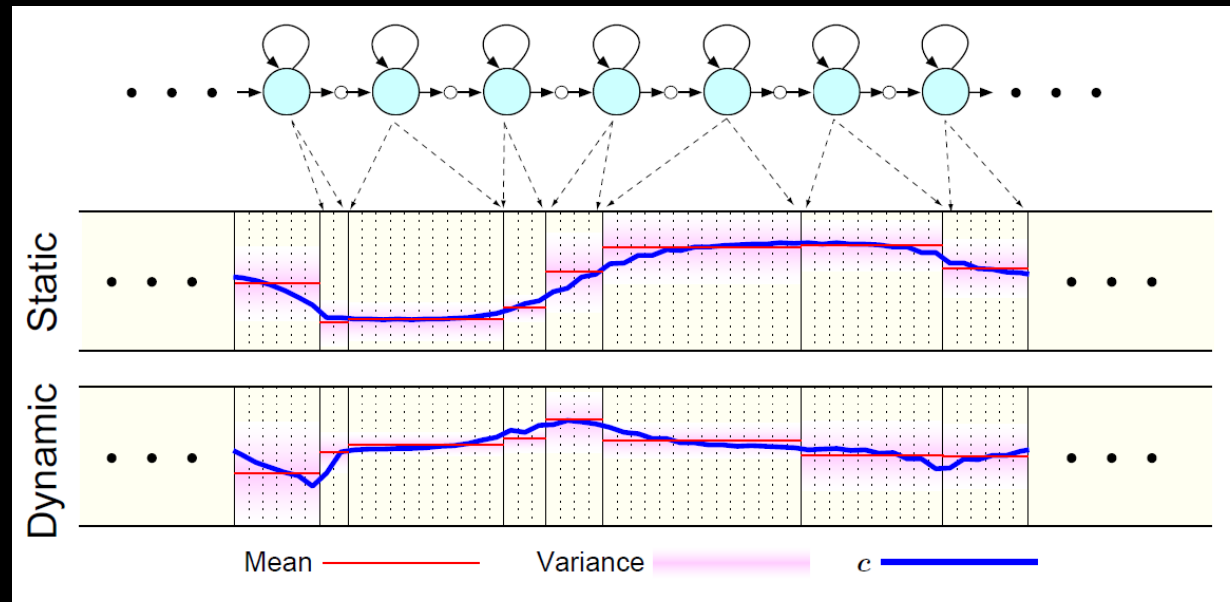
Problem #2: Sequence Generation (II)

$$\begin{array}{c}
 \begin{array}{c} \mathbf{W}^\top \end{array} \quad \begin{array}{c} \Sigma_{\hat{q}}^{-1} \end{array} \quad \begin{array}{c} \mathbf{W} \end{array} \quad \begin{array}{c} \mathbf{c} \end{array} \\
 \begin{array}{|c|c|c|c|c|} \hline 1 & 0 & 0 & \dots & \\ \hline 0 & 1 & 0 & \dots & \\ \hline 0 & 0 & 1 & 0 & \dots \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|c|} \hline \Sigma_{q_1} & & & & \\ \hline & \Sigma_{q_2} & & & \\ \hline & & \ddots & & \\ \hline & & & \Sigma_{q_T} & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|c|} \hline 1 & 0 & 0 & \dots & \\ \hline 0 & 0 & 0 & \dots & \\ \hline 0 & 1 & 0 & \dots & \\ \hline -1 & 1 & 0 & \dots & \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline \dots & 0 & 1 & 0 & \\ \hline \dots & -1 & 1 & 0 & \\ \hline \dots & 0 & 0 & 1 & \\ \hline \dots & 0 & -1 & 1 & \\ \hline \end{array} \quad \begin{array}{|c|} \hline c_1 \\ \hline c_2 \\ \hline \vdots \\ \hline c_T \\ \hline \end{array} \\
 \\
 = \quad \begin{array}{c} \mathbf{W}^\top \end{array} \quad \begin{array}{c} \Sigma_{\hat{q}}^{-1} \end{array} \quad \begin{array}{c} \mu_{\hat{q}} \end{array} \\
 \begin{array}{|c|c|c|c|c|} \hline 1 & 0 & 0 & \dots & \\ \hline 0 & 1 & 0 & \dots & \\ \hline 0 & 0 & 1 & 0 & \dots \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|c|} \hline \Sigma_{q_1} & & & & \\ \hline & \Sigma_{q_2} & & & \\ \hline & & \ddots & & \\ \hline & & & \Sigma_{q_T} & \\ \hline \end{array} \quad \begin{array}{|c|} \hline \mu_{q_1} \\ \hline \mu_{q_2} \\ \hline \vdots \\ \hline \mu_{q_T} \\ \hline \end{array}
 \end{array}$$

Tokuda & Zen (2009)

Problem #2: Sequence Generation (III)

- Things look much better now:



(*) Tokuda & Zen (2009)

- We now have the generated parameters, which we can put through our resynthesis algorithm to convert back to speech!!

To find out more:

- ✓ *The IBM Expressive Text-to-Speech Synthesis System for American English* -- Pitrelli et al., 2006.
(Overview of a concatenative architecture and approach to synthesizing expressive speech.)
- ✓ *Fundamentals and Recent Advances in HMM-Based Speech Synthesis* -- Tokuda & Zen, 2009.
(Interspeech 2009 Tutorial on HMM synthesis, from which several plots in this presentation are taken).