Speech Processing

Undergraduate course code: LASC10061
Postgraduate course code: LASC11065

Speech Synthesis

- The next section of the course is on speech synthesis
  - Defining the problem
  - Applications
  - Text-to-speech (TTS)
  - Synthesis methods
  - Diphones

http://www.cstr.ed.ac.uk/projects/festival/morevoices.html

Examples: diphones, unit selection, HMMs

Assessed speech synthesis practical

- The main objective is to “learn by doing”
- This practical is in three parts
    - Part I – Festival: running the pipeline step-by-step
    - Part II – Festival: finding and explaining errors
    - Part III – mini literature review
- **Due**: see Learn for due date and submission instructions
- **Suggestion**: keep a detailed lab book in which you record every thing you do

What is speech synthesis?

- **We will define speech synthesis by what it is not:**
  - Playback of whole sentences (this is called ‘canned speech’)
  - Only being able to say a set of fixed sentences
- **and by what it is:**
  - Synthesising new sentences
  - Usually including new words

- First, lets look at some applications

Applications

- **The application may determine the method we choose**
    - Automated services, e.g., reading your email by telephone
    - Assistive technologies, e.g., reading machines (screen readers), voice output communication aids
    - Interactive dialogue systems, e.g., flight booking systems
    - Entertainment, e.g., taking characters in a computer game, ebook reader
    - Speech-to-speech translation

- Can you think of any other applications?

Input

- **The form of input to the system might be:**
  - Plain text
  - Marked-up text
  - Semantic (concept)
- **On this course we will limit ourselves to**
  - text-to-speech systems where the input is plain text
  - the diphone method for waveform generation
- **The second semester course “Speech synthesis” covers more advanced material, including**
  - the unit selection method for waveform generation,
  - statistical parametric speech synthesis

Text-to-speech (TTS)

- **Definition: a text-to-speech system must be**
  - Able to read any text
  - Intelligible
  - Natural sounding

- The first of these puts a constraint on the method we can choose:
  - *playback of whole words or phrases in not a solution*

- The second is actually closer to being a '*solved problem*' than the third

Methods

- The methods available for speech synthesis fall into two categories:
  - Model-based, or parametric
  - **Concatenative** - *we will only cover this type of system*
- In the past, model-based ***used*** to only mean
  - some sort of simplified model of speech production
  - which requires values for a set of parameters (e.g. formant frequencies)
  - which are in turn generated from hand-crafted rules
- Concatenative systems use
  - recorded examples of real speech

Concatenative systems (“cut and paste”)

- Most common method in state-of-the-art commercial and research systems.
- Example systems
  - CHATR, Ximera – ATR, Japan
  - Festival – University of Edinburgh, UK (*Open Source*)
  - rVoice – Rhetorical, UK (now Nuance)
  - Natural Voices – AT&T, USA
  - RealSpeak – ScanSoft (now Nuance)
  - Vocalizer – Nuance
  - Loquendo TTS – Loquendo, Italy (now Nuance)
  - InterPhonic – iFlyTek, China
  - IVONA – IVO software, Poland (now Amazon)
  - SVOX, Switzerland (now Nuance)
  - Cepstral, USA
  - Phonetic Arts, UK (now Google)
  - CereVoice - Cereproc, UK
- **Concatenative speech synthesis** = joining together pre-recorded units of speech

Pros of concatenative synthesis

- Can change the voice relatively easily (but not necessarily cheaply) without changing any software
    - Just record a new speaker
- Can sound very much like a particular individual
- On a good day – very natural sounding indeed

Cons of concatenative systems

- On a bad day – just plain awful!
- Large database of speech required for best quality
  - Expensive to collect; large memory/disk requirements; problems in maintaining consistent voice quality during recording
- Can sometimes hear the joins between units
- Control over most aspects of the speech is limited
  - F0, duration control is possible (some signal degradation); voice quality control is not possible
- Concatenative systems sound much better than rule-driven models
  - but statistical parametric synthesis is nearly as good, and more flexible

Components of a concatenative system

- In this course we will examine a typical concatenative TTS system
- We’ll look at the pipeline of processes that takes us from input text to output waveform; the pipeline can be broken into two main parts
  - the ‘front end’
  - waveform generation
- We’ll see how the front end infers additional information like pronunciation, intonation and phrasing to produce a ‘**linguistic specification**’
- The pitch and duration are manipulated during waveform generation, in order to convey this information

Examples: diphones vs. unit selection

Techniques required

- A variety of techniques will be required in the various components of our synthesiser.
- Natural language processing
  - text analysis, morphology, syntactic parsing
- Phonetics and phonology
  - generating pronunciations, syllabification of unknown words
- Prosody
  - determining durations and F0 (e.g. pitch accents)
- Signal processing
  - generating the waveform

A text-to-speech system: Festival

- The practicals will use Festival version 1.96 which is a complete toolkit for speech synthesis research, widely used around the world. The principal stages in the pipeline are:
- Text processing
  - Tokenisation; rules (e.g. for dates and numbers)
  - Part of speech tagging
  - Phrase break prediction
- Pronunciation
  - Lexicon
  - Letter-to-sound rules or decision tree (CART) trained on data
- Duration prediction
  - CART trained on data

A text-to-speech system: Festival

- *Intonation (not covered in this course)*
    - *TOBI accents predicted using CART models*
- Waveform generation
    - Diphone units
    - Various signal processing methods
      (PSOLA, LPC, MBROLA)

The ‘front end’

From input text to linguistic specification

Text processing

- Text processing breaks the original input text into units suitable for further processing; this involves tasks such as
  - expanding abbreviations
  - part-of-speech (POS) tagging
  - letter-to-sound rules
  - prosody prediction
- We end up with a '**linguistic specification**' - in other words, all the information required to generate a speech waveform, such as
  - phone sequence
  - phone durations
  - pitch contour

Tokenisation

- The input to a TTS system can be any text, for example:

  In 1871, Stanley famously said "Dr. Livingston, I presume"

- Punctuation is generally preserved, so this might be tokenised as:

  (In) (1871) (,) (Stanley) (famously) (said) (") (Dr.) (Livingston) (,) (I) (presume) (")

- In some systems, the punctuation is stored as a feature of the preceding or following token

Abbreviations

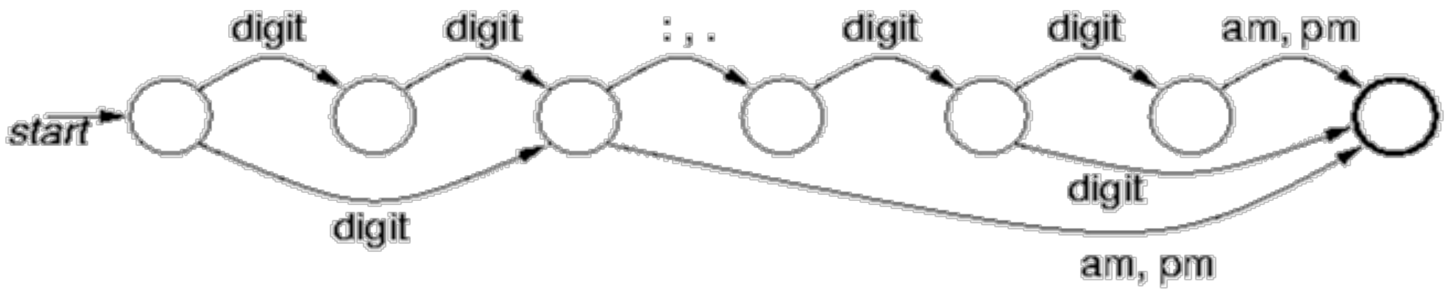
- In text, abbreviations are often used, but conventionally they are read out fully:
- Even simple abbreviations can be ambiguous, e.g.:
    - Dr. Livingston vs. Livingston Dr.
    - St. James vs. James St.
    - V can be a roman numeral or Volts
    - 100m could be “100 million” or “100 metres” or “100 miles”, …
- The system must
    - recognise abbreviations
    - then expand them

Numbers

- The interpretation of numbers is context sensitive
  - 2.16pm
  - 15:22
  - 2.1
  - 20/11/05
  - The 2nd
  - $100bn
  - 99p
  - 0131 651 3174
- Simple rules can be used to expand most of these into words, although writing such rules is pretty tedious, and often language dependent

Finite state methods

- A common implementation of rules used to recognise abbreviations, for example, is as regular expressions (or their equivalent finite state machine)
- Here is a machine which will accept time expressions like 2.16pm, 15:22, 4am, 11.05am. The labels on the arcs are the input symbols.



The arcs could have probabilities, and by adding the output symbols, the machine becomes a finite state transducer, which can simultaneously recognise and expand abbreviations (see Jurafsky and Martin)
Finite state machines are computationally efficient (fast, low memory)

From letters to sounds

- Once we have a sequence of fully spelled-out words, we next need to work towards a sequence of phonemes
- Morphology (optional - not very helpful for English)
- Part-of-speech (POS) tagging
- The lexicon
- Post-lexical rules
- Letter-to-sound (LTS) rules
    - Which will involve a very useful type of model called a “Classification and Regression Tree (CART)”

Part-of-Speech (POS)

- Some words have multiple possible POS categories
- We must **disambiguate** the POS:
  - without POS information, pronunciation might be ambiguous e.g. “lives”
  - POS will also be used to predict the prosody later on
- POS tagging is the process of determining a single POS tag for each word in the input; the method can be
  - deterministic, or
  - probabilistic

Penn treebank POS tag set

- CC Coordinating conjunction
- CD Cardinal number
- DT Determiner
- EX Existential there
- FW Foreign word
- IN Preposition or subordinating conjunction
- JJ Adjective
- JJR Adjective, comparative
- JJS Adjective, superlative
- LS List item marker
- MD Modal
- NN Noun, singular or mass
- NNS Noun, plural
- NNP Proper noun, singular
- NNPS Proper noun, plural
- PDT Predeterminer
- POS Possessive ending
- PRP Personal pronoun
- PRP$ Possessive pronoun
- RB Adverb
- RBR Adverb, comparative
- RBS Adverb, superlative
- RP Particle
- SYM Symbol
- TO to
- UH Interjection
- VB Verb, base form
- VBD Verb, past tense
- VBG Verb, gerund or present participle
- VBN Verb, past participle
- VBP Verb, non-3rd person singular present
- VBZ Verb, 3rd person singular present
- WDT Wh-determiner
- WP Wh-pronoun
- WP$ Possessive wh-pronoun
- WRB Wh-adverb

plus 9 tags for punctuation

Probabilistic POS tagging

- One of the simplest and most popular methods is to train models on labelled data (i.e., already tagged, by hand), combining
  - HMMs (Hidden Markov Models):
    - where the observations are words and the models are the POS classes (This will make more sense after the speech recognition part of the course)
  - N-grams
- The latest state-of-the-art taggers are extremely accurate. Festival's tagger is now somewhat dated, but performs well enough

Progress check

- Our TTS system is a pipeline, taking words and gradually transforming them into speech. How far have we got?

text	Dogs	like	to	bark.	
token	(Dogs)	(like)	(to)	(bark)	(.)
POS	NNS	VBP	TO	VB	.

The lexicon

- The lexicon entries have three parts:
  - Head word
  - POS
  - Pronunciation (in terms of phonemes)
- The POS is sometimes necessary to distinguish homographs, e.g.:

head	POS	phonemes
lives	NNS	l ai v z
lives	VBZ	l I v z

Syllables and lexical stress

- The lexicon will usually also mark syllable structure and lexical stress
  - present n (((p r eh z) 1) ((ax n t) 0))
  - present v (((p r iy z) 0) ((eh n t) 1))
- In Festival, there are three steps to find the pronunciation of a word:
  - Look up in main lexicon
  - If not found, look up in addenda (e.g. domain specific additional lexicon)
  - If not found, use letter-to-sound model
- The main lexicon is large and ordered to allow fast searching, the addenda contains a small number of words added by hand, and the letter-to-sound model will deal with the rest

Letter-to-sound

- If lexical lookup fails, we fall back on letter-to-sound rules
- Example:
  - The letter c can be realised as /k/, /ch/, /s/, /sh/, /ts/ or /ɛ/ [deleted]
  - We might write rules like:
    - If the “c” is word-initial and followed by “i” then map to /s/
    - If the “c” is word-initial and followed by “h” then map to /ch/
    - If ...
- This approach works well for Spanish, but performs very poorly for English
- In general, we want an automatic method for constructing these “rules”
  - The most popular form of model: a **classification tree**

Post-lexical rules

- The lexicon and letter-to-sound rules arrive at a pronunciation for each word *as it would be spoken in isolation*, known as the "**citation form**"
- Now we need to apply cross word and phrasal effects such as:
  - Vowel reduction
  - Phrase-final devoicing
  - r-insertion
- Since these effects are small in number, hand written rules work OK
- Festival has a mixture of
  - hard-wired rules (compiled into the C++ code), and
  - voice specific rules (implemented in Scheme which can be changed at run-time)

Progress

text	Dogs	like	to	bark.	
token	(Dogs)	(like)	(to)	(bark)	(.)
POS	NNS	VBP	TO	VB	.
Lex/lts	/d aa g z/)	/l ay k/	/t uw/	/b aa r k/	
postlex	/d aa g z/	/l ay k/	/t ax/	/b aa r k/	

CART – classification and regression trees

- These are decision trees for predicting the value of either a
  - Categorical variable (classification tree)
  - Continuous variable (regression tree)
- We’ll consider only the categorical case, but the principles are the same for continuous variables
- The nodes in the tree are questions about features which describe the environment
- The tree is learned automatically from data
- Trees are human readable and editable (mostly)
- Concise and fast
- Automatically select predictors that are useful, ignores those that are not

Learning from data: the two main stages

- It's very important to make a clear distinction between:

- **Learning the model from data ("training")**
    - we obtain some labelled training data
    - we choose some form of model (e.g., classification tree)
    - we fit the model to the training data (e.g., grow the tree)

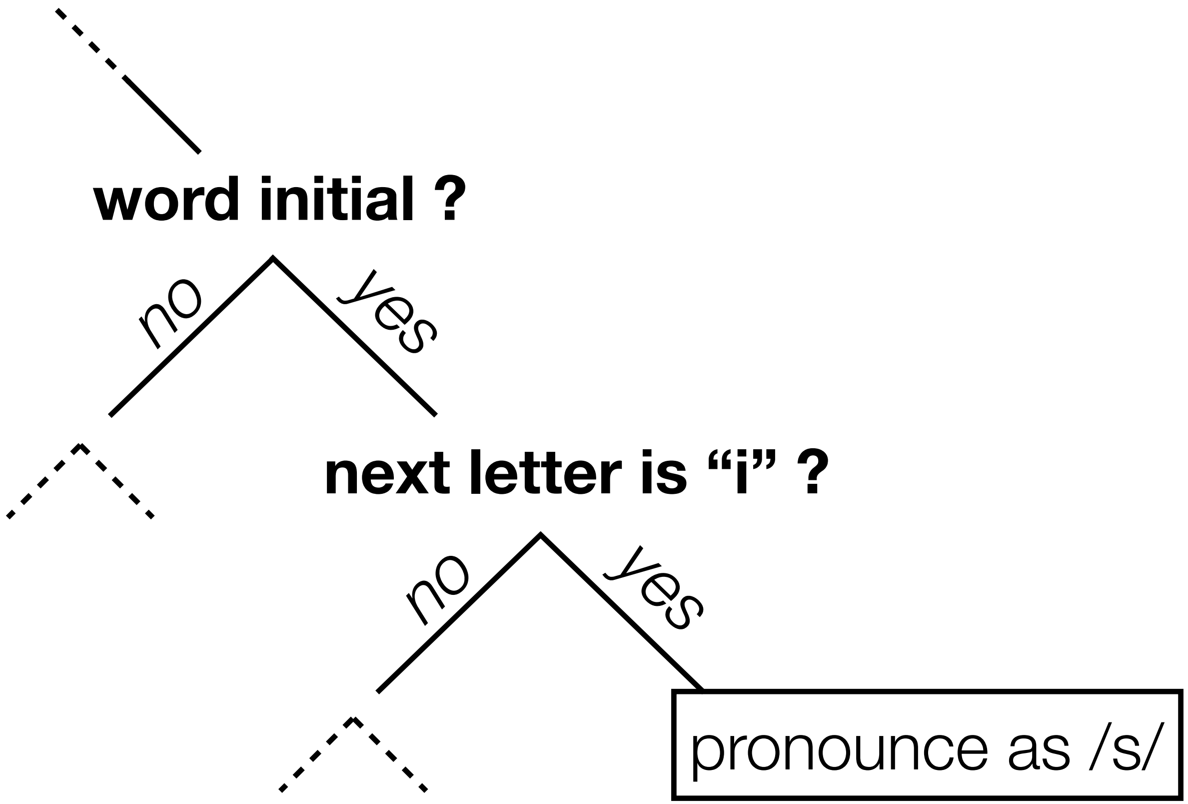
- **Using the model to make classifications, predictions, etc. ("testing")**
    - we have some unlabelled test data
    - we use the model to label the test data

Predictors and predictees

- Predictors: things whose value we know (think independent variables)
- Can be just about anything
  - Continuously valued
  - Discrete (categorical)
- Predictee: the thing whose value you want to predict (think dependant variable)
- Letter-to-sound “rules” can be written as a classification tree
  - The predictors used for letter to sound rules might include: the surrounding context letters, position in the word, word boundary information

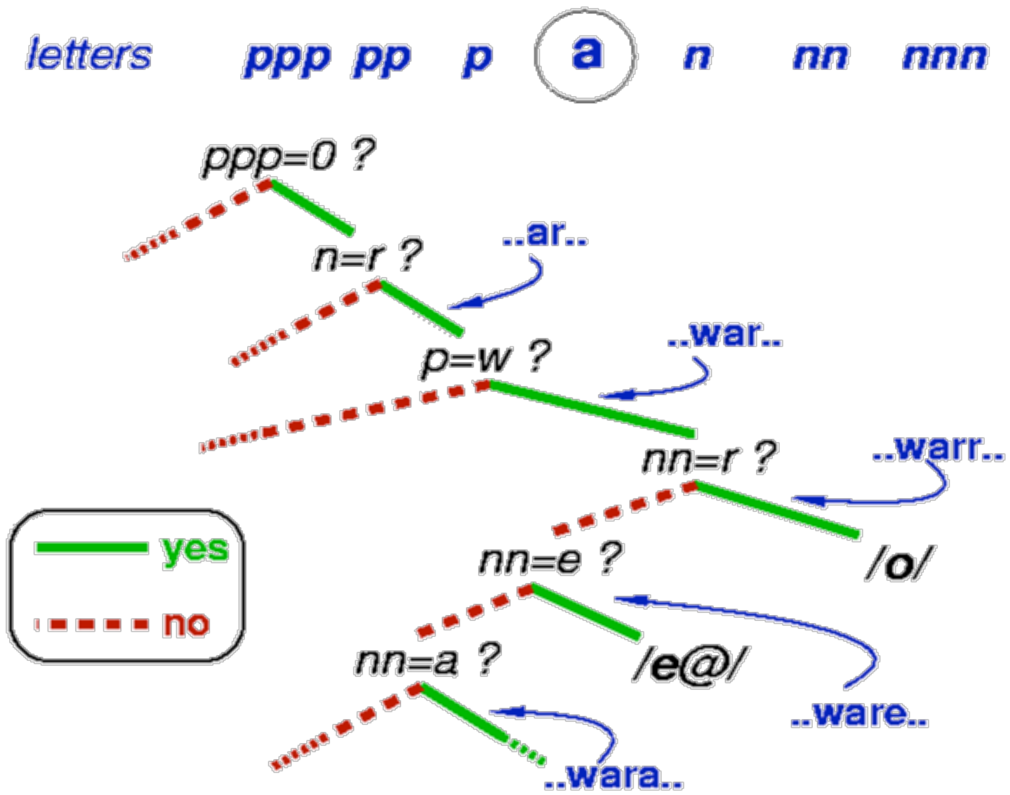
Classification trees are equivalent to ordered rules

- Here's a fragment of a tree - we've already decided the letter is "c" :



Part of Festival’s LTS tree

- Here is a fragment of the LTS tree from Festival: letter “a” for British English



Learning a CART from data: prerequisites

- Before learning a CART model, we need to specify:
  - The predictors (sometimes called features)
  - The predictee
  - All the possible questions we can ask about the predictors

- The list of possible questions can be determined automatically (e.g., ask whether a categorical predictor is equal to each possible value it can take)
- The training algorithm will choose which questions to use, and where to put then in the decision tree

Questions

- For discrete predictors, question are simply of the form:
  - Is value of predictor equal to v ?
  - Is value of predictor in the set {u,v,w}?
- The number of possible questions of the first type is much smaller than for the second type
- For continuous predictors, questions are simply of the form:
  - Is the value of predictor greater than v
- To reduce the space of possible questions, can try only a fixed number of v values (e.g. 10). This is in effect **quantising** the continuous variable and then treating it as discrete

Learning a CART from data: algorithm

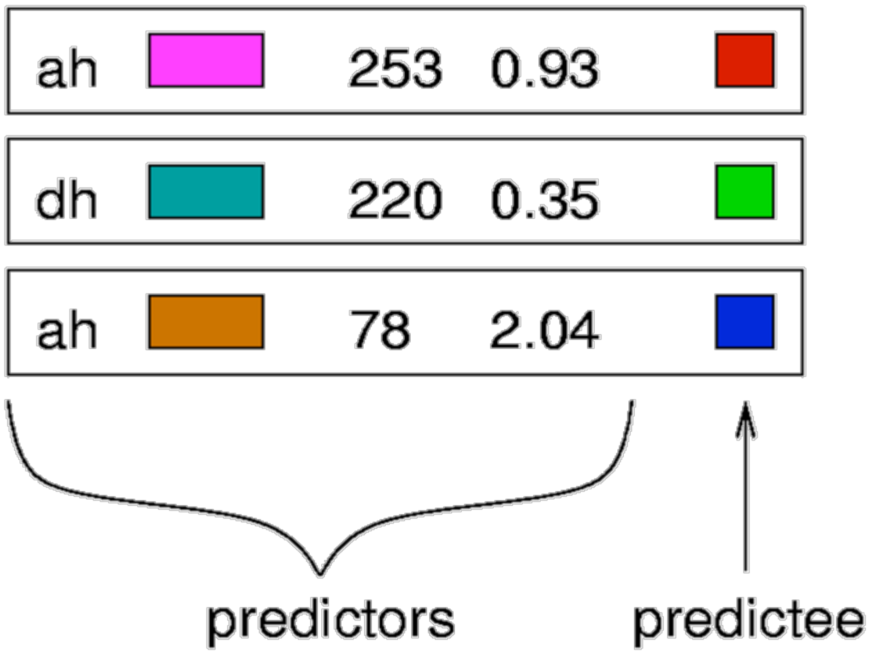
- At the start, all data is placed in a single set at the root node of the tree
- A question is placed in the tree, and the data is split according to it: the data is partitioned into two subsets, which descend the branches of the tree. This procedure is then recursively applied
- At each iteration, we need to decide:
    - <u>Which question to put into the tree next?</u>
        - need to measure how well each question splits the data, i.e., how coherent the resulting subsets are (e.g., measure variance for continuous data or entropy for discrete data)
    - <u>Whether to stop growing the tree?</u>
        - some stopping criterion is required, such as a minimum number of data points in a subset)

Learning a CART from data: pseudo code

- `Function: partition()`
  - `Consider each possible question in turn`
    - `Choose the question that splits the data into the most consistent two subsets`
  - `Place this question in the tree`
  - `Partition the data using question`
  - `Send the resulting subsets of data down the branches of the tree`
  - `Recurse: for each subset, call partition()`
- To start the algorithm, we make a tree with only one node (the root), place all of the data there, and call partition() on it
- This type of algorithm is called a **greedy algorithm** – at a given point during the training procedure, a decision is made which gives the best outcome at that point, with no regard to how it will affect future outcomes. There is no backtracking

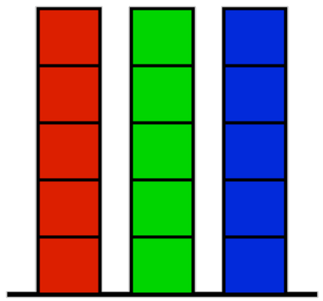
Discrete predictee example

- The predictee is discrete and can take one of three values: red, green or blue
- Each training example has particular values for the predictors and a known value for the predictee
- Here are three training examples:



Training data

- The training data set has a total of 5 examples for each of these classes
- Here is the distribution of the predictee values in the training set:

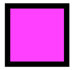


They are all equally likely, in other words:
The probability of each predictee value occurring in the complete training set is 1/3 or about 0.33

Probability

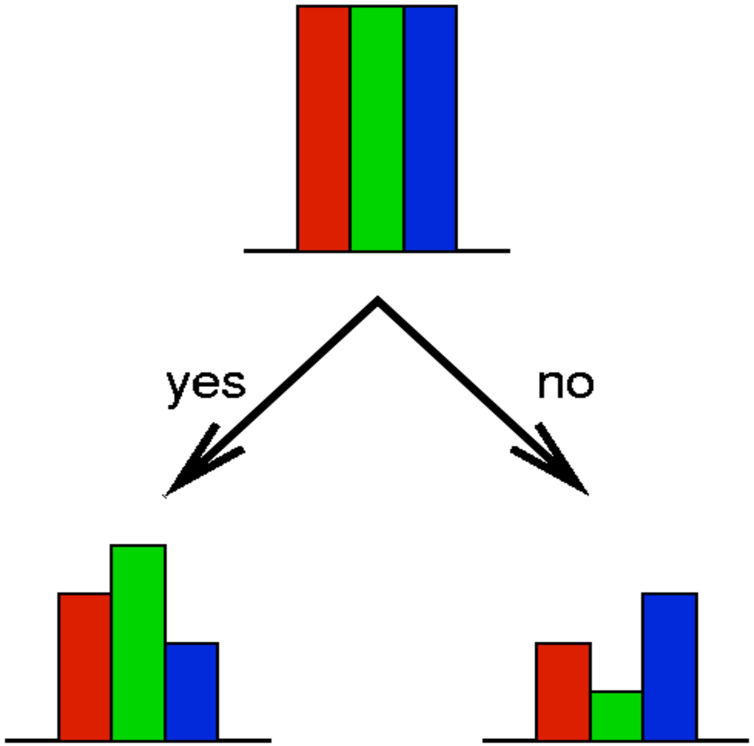
- In this simple example, the probability distribution of the training data predictee value is:
  - P(red) = 1/3
  - P(green) = 1/3
  - P(blue) = 1/3
- We will be coming back to probability in more depth in the second half of this course.

Possible questions

- Does feature1 = “ah”?
- Does feature1 = “dh”?
- Does feature2 = ?
- Does feature2 = ?
- Does feature2 = ?
- Is feature3 > 0?
- Is feature3 > 100?
- Is feature3 > 200?
- Is feature4 > 1?
- etc.

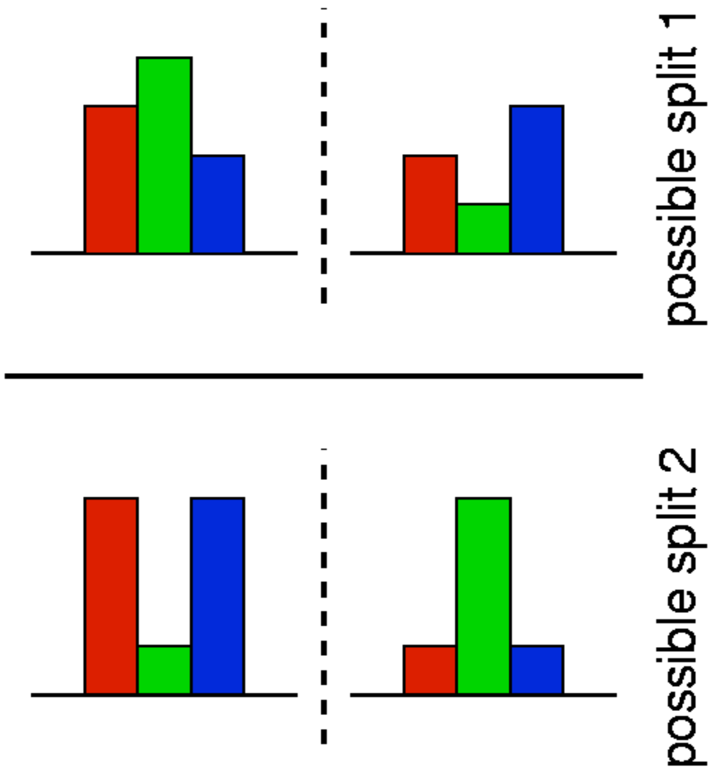
Partitioning

- Try each yes/no question in turn.
- Here is what happens for one question we are trying:



Entropy measures 'purity'

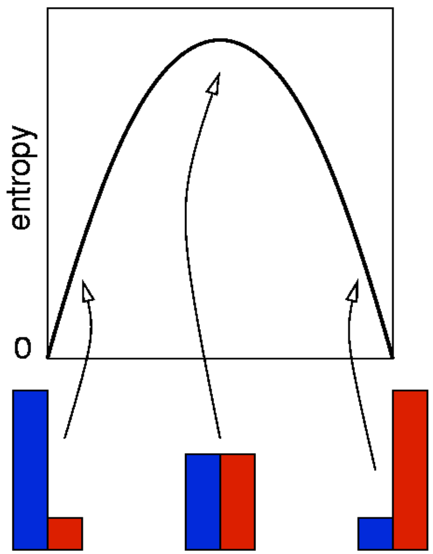
- Low entropy means highly predictable.
- Here is what happens for two different questions we are trying, which each give a different split of the data:



In the second question (lower figure), the total entropy is lower, so this is a better split of the data

Entropy more formally:

- Entropy is: $$H = -\sum_x p(x) log(p(x))$$



Entropy is zero when things are 100% predictable,
e.g., everything is blue

How big should the tree grow?

- We want to stop the tree-building algorithm at some point
- Need a criterion for when to stop
  - When none of the remaining questions usefully split the data
  - Limit the depth of the tree
  - When the number of data points in a partition is too small
- Don’t simply want to continue until we run out of questions because:
  - Not all questions usefully split the data (perhaps because not all predictees are informative)
  - Can’t reliably measure goodness of split for small data partitions

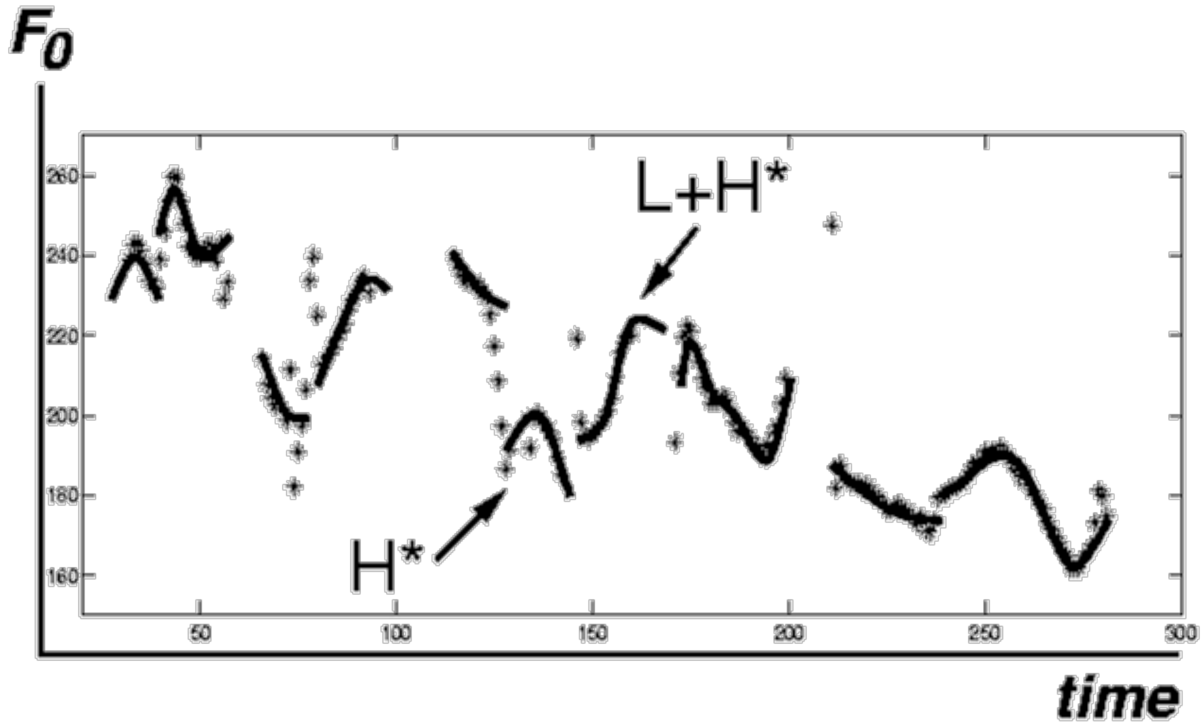
When can CART be used

- When there are a number of predictors, possibly of mixed types
- When we don't know which are the most important ones
- When some predictors might not be useful
- When we can ask yes/no questions about the predictors

Prosody and intonation - in brief!

- Recap:
  - We have processed the text into tokens and then into words
  - We have determined a sequence of phonemes
- We now turn to **suprasegmental** aspects of synthesis.
- Not covering this in detail in this course, just need to mention
  - phrase boundaries
  - intonation events (e.g., pitch accents & breaks)
  - realisation of intonation via the F0 contour

Describing intonation using ToBI



ToBI provides a stylised symbolic representation suitable for hand-annotation of data, and for computation

Automatic phrase boundary prediction

- Task: predict boundary position and strength
- Equivalent task: predict a boundary strength after every word (some are zero)
- **Predictee**
  - *break strength*
    - Festival uses just 3 boundary strengths (instead of ToBI's 5): Major (BB [big break]), Minor (B [break]), No break (NB)
- **Predictors**
  - *contextual features* of current and neighbouring syllables (similar to intonational event prediction - see next slide)
- Models
  - CART
  - Markov model with N-gram

Automatic intonation event prediction: placement

- Step 1: placement
  - **Predictee**
    - *placement* (whether a syllable receives an accent)
  - **Predictors**
    - Syllable position in phrase
    - Syllable context
    - Lexical stress
    - Lexical stress of neighbours
    - Break strength of this word and neighbouring words
    - POS tags

Automatic intonation event prediction: type

- Step 2: type
  - **Predictee**
    - *accent type*
      - For ToBI, one of: L*, H*, L*+H, L+H*, H+L* or a boundary tone L% or H% (using a CART as a **classification tree**)
      - In parametric models, a parameterised representation of accent height, duration, etc. (using a CART as a **regression tree**)
  - **Predictors**
    - again, a number of factors relating to the syllable in question and its context

Automatic intonation event prediction: realisation

- Step 3: realisation
    - The ToBI symbol must now be realised as actual F0 values.
    - Typically predict F0 at 3 points per syllable
    - It will not come as surprise that this prediction too can be done using a model trained on data
        - We’re now predicting continuous values
        - Use a CART : this time as a **regression** tree

Waveform generation

Waveform generation

- Now we have got
  - sequence of phonemes
  - F0 and duration for all phonemes
    - we didn't discuss duration prediction in detail, but you can work out for yourself the type of model and the predictees we could use
- All that remains is to
  - concatenate the recorded speech units
  - impose the required F0 and duration using signal processing
- This stage of the pipeline is called **waveform generation**
  - the techniques will generally be language-independent

Concatenative synthesis

- What size are the units (pieces of pre-recorded speech) that we are going to concatenate?
- Large
  - Fewer joins per utterance
  - but, more unit **types** means a larger inventory is needed
- Small
  - Fewer unit types means a smaller inventory is needed
  - but, more joins per utterance

Why are joins bad?

- We will hear them!
- Why?
  - Mismatch between units
    - Pitch
    - Amplitude
    - Natural variation in segment quality
    - Co-articulation / assimilation effects
  - Signal processing artefacts
    - properties of the signal not present in the original speech

Why is a smaller inventory good?

- Easier and quick to construct and record
- Easier to store at run-time
- Quicker to access units
- Smaller set of possible unit type sequences for any given utterance to be synthesised (possibly a unique sequence; e.g., phonemes, diphones)

Possible choices of unit size

- Sub-phone sized units (e.g. half phone)
- Phones
- Diphones (same size as phones)
- Demi-syllables
- Syllables
- Words
- Phrases

- We need to trade off: the number of joins, how noticeable they will be and the inventory size

Phones?

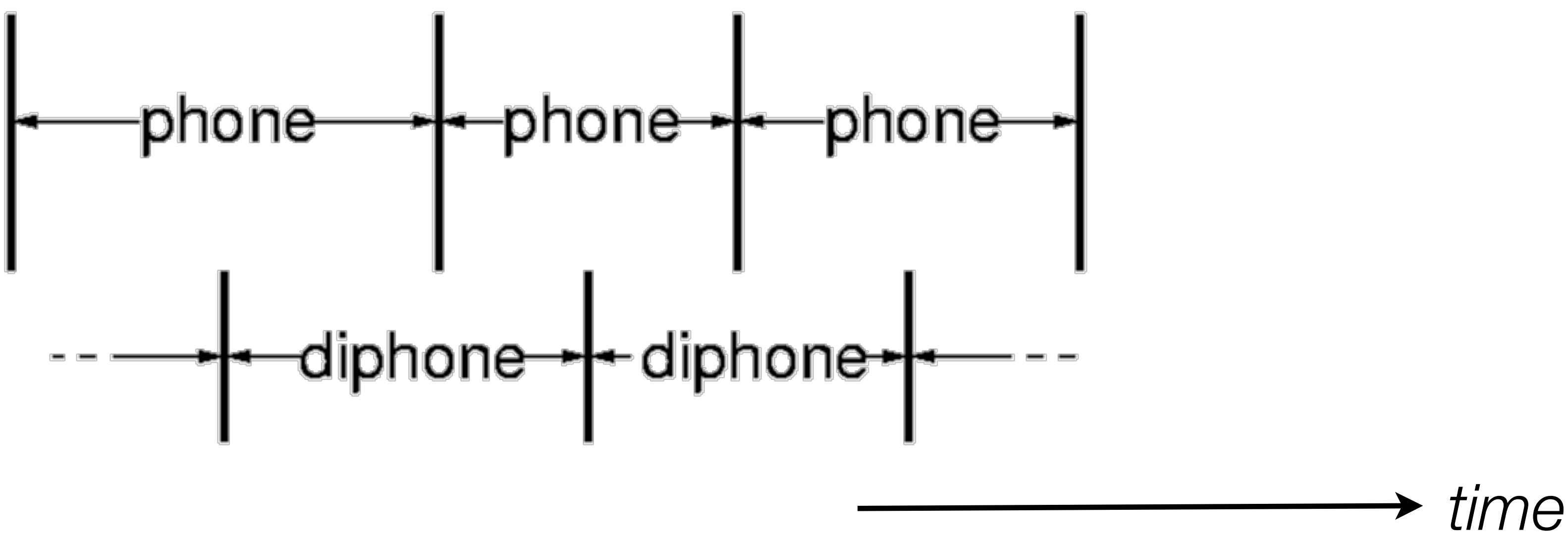
- Phones (i.e. recorded instances of phonemes) are an obvious choice: but are they a good unit for concatenative synthesis?
    - Small inventory
    - Unique unit sequence
    - But
        - Lots of joins
        - Very context dependent

Joining phones

- Joins will be a phone boundaries
- Where there is a maximum amount of coarticulation
    - The articulators are on the move from the configuration of the previous phone to the next phone
    - Articulator position depends on both the left and right phone
        - Acoustic signal is determined by articulator positions
        - Therefore acoustic signal is highly context dependent
- Phones are not suitable

Diphones

- Why are diphones a good idea?
- We have moved the concatenation points (joins) to the mid-phone position
- Diphones are the second half of one phone plus the first half of the following phone
- There will still be one join per phone



Advantages of diphones

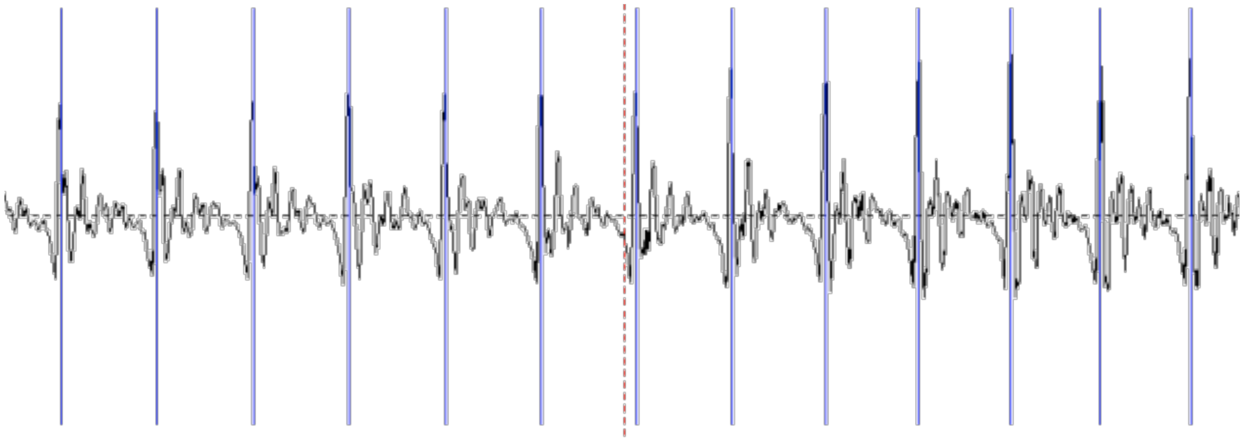
- Joins will be mid-phone
    - The mid-point of a phones is relatively acoustically stable
    - Further from phone edges means less context sensitive
- Still fairly small inventory
    - approximately (number of phones)^2
- For any given phone sequence: there is a unique diphone sequence
- Less context dependent than phones
- But still lots of joins, although in better positions than with phone units

Alternatives to diphones

- Diphones tend to be the standard unit for concatenative synthesis, but there are alternatives:
  - Smaller units
    - e.g., AT&T's Next Gen uses half phones
  - Larger units
    - syllables, half syllables
  - Mixed inventory
    - syllables, half syllables, diphones, affixes

Time-domain

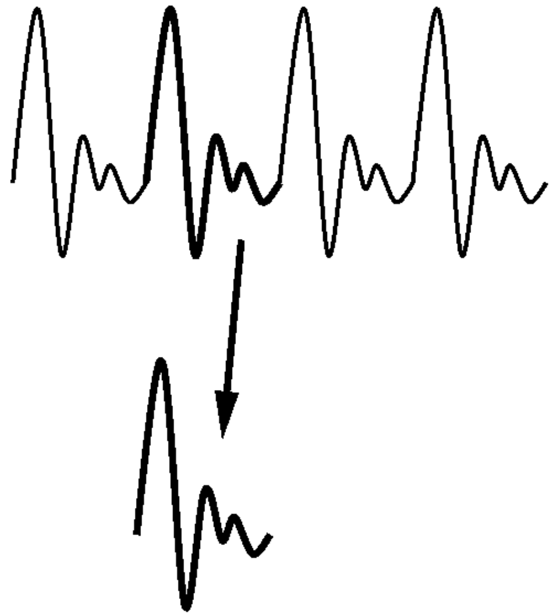
- The inventory contains the waveform plus pitch-marks for each speech unit (i.e., diphone)



Units have their original duration and F0, which will get modified during waveform generation
The pitch-marks are needed by PSOLA-type algorithms

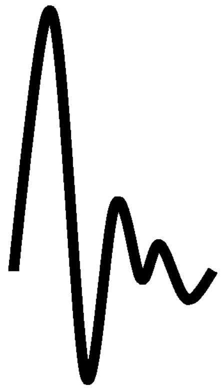
PSOLA (Pitch Synchronous OverLap and Add)

- The first method we consider for modifying F0 and duration is a time domain version of PSOLA called TD-PSOLA.
- It operates directly on waveforms



How TD-PSOLA works

- Deal with individual pitch periods (each of which is essentially the impulse response of the vocal tract)



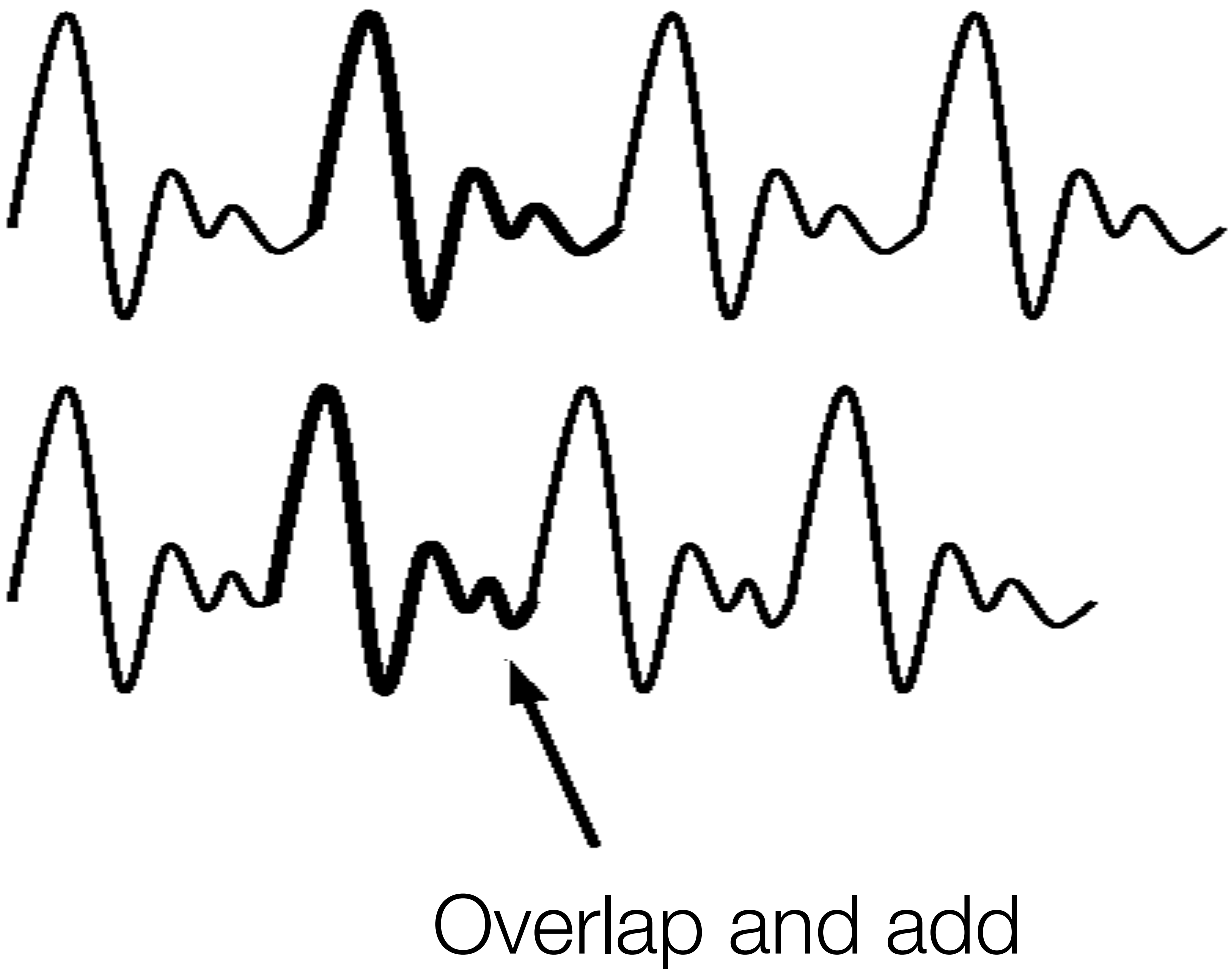
- The pitch periods themselves are not modified
- To increase F0, periods are moved closer together; where they overlap, we add the waveforms
- To decrease F0, periods are moved further apart

TD-PSOLA

- Decreasing F0:

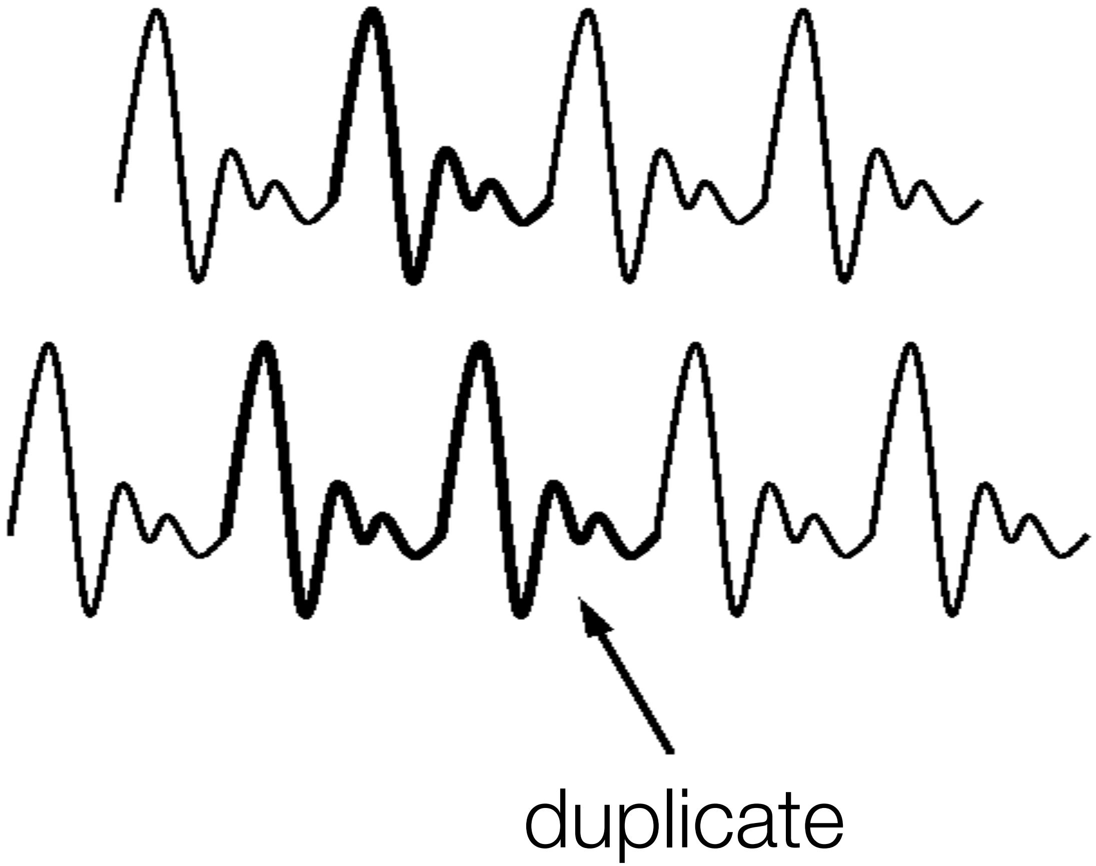
TD-PSOLA

- Increasing F0:



TD-PSOLA

- Increasing duration:



TD-PSOLA for duration and F0 modification

- Modify duration by duplicating or deleting pitch periods
- Modify F0 by changing the spacing between pitch periods
- In practice, the pitch periods are windowed to give smooth joins
    - we actually deal with two pitch periods, windowed
- We also have to compensate by adding or deleting pitch periods when modifying F0, if duration is to be kept the same

Advantages of TD-PSOLA

- Incredibly simple
- Works in time-domain
- Computationally very fast
- No coding/decoding of waveform (no explicit source-filter separation), so potentially very few artefacts

Problems with TD-PSOLA

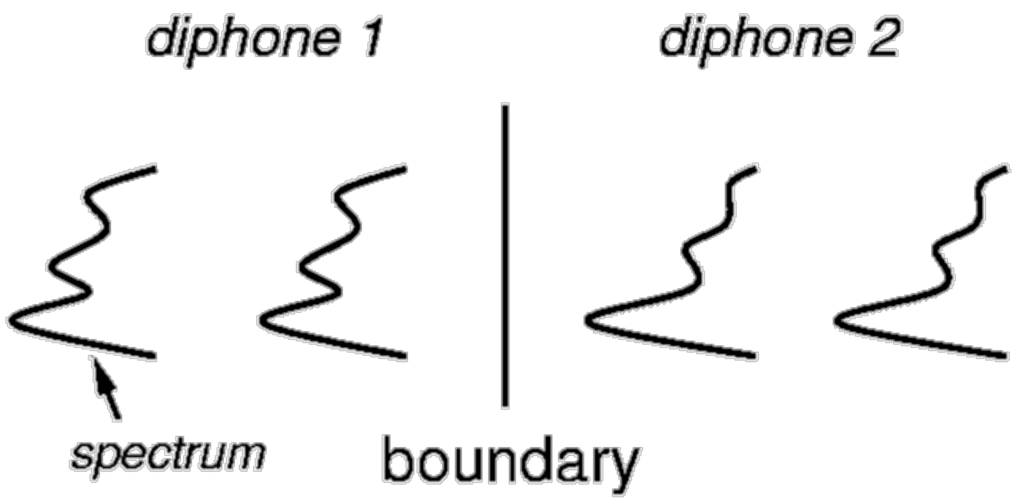
- Overlap-add algorithm can add artefacts
- High F0 or duration modification factors sound bad (as you can discover for yourself with Praat)
- Duration modification limited to whole pitch periods
- Needs very accurate and consistent pitch marking
- Cannot modify spectral shape to smooth joins
- Must use pseudo-pitch marks for unvoiced speech
    - this can introduce periodic sounds - perceived as buzziness

Linear predictive synthesis

- An alternative to time-domain PSOLA for manipulating F0, duration and in additional the spectral envelope (related to vocal tract shape)
- Overcomes some problems of TD-PSOLA
- Widely used
    - the default method in Festival
- With a few tweaks, can sound very good

What is spectral mismatch?

- The spectrum of the two diphones either side of a boundary, will not (usually) be the same
  - because they were recorded separately
- This will be true, however carefully we construct our database

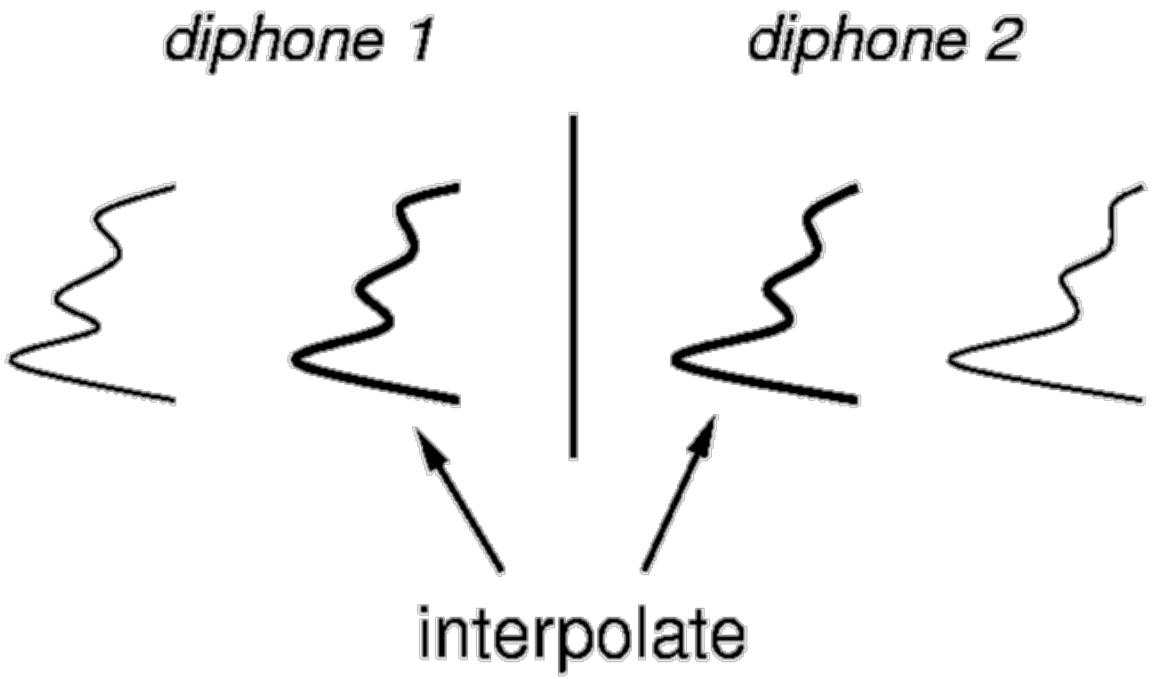


Working in another domain

- Need to hide this spectral mismatch
  - Smooth the transition across the boundary
  - Working in the time domain does not allow this
- We need:
  - to manipulate F0 and duration independently
  - an explicit representation of the spectral envelope in order to remove mismatch across joins

How to remove spectral mismatch

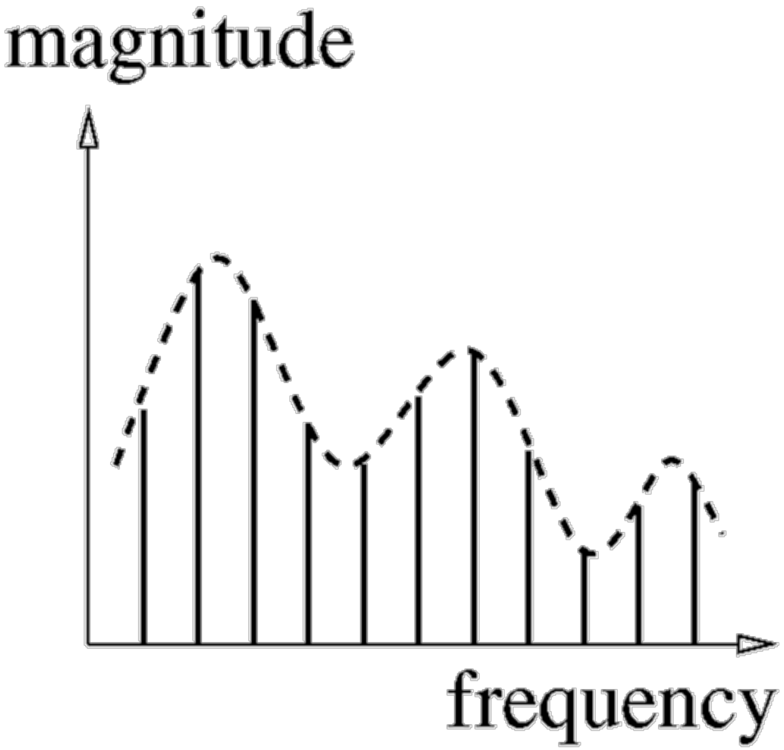
- Interpolate in the frequency domain



Manipulate the spectral envelope to disguise discontinuity in the vocal tract shape

Reminder: spectrum of speech

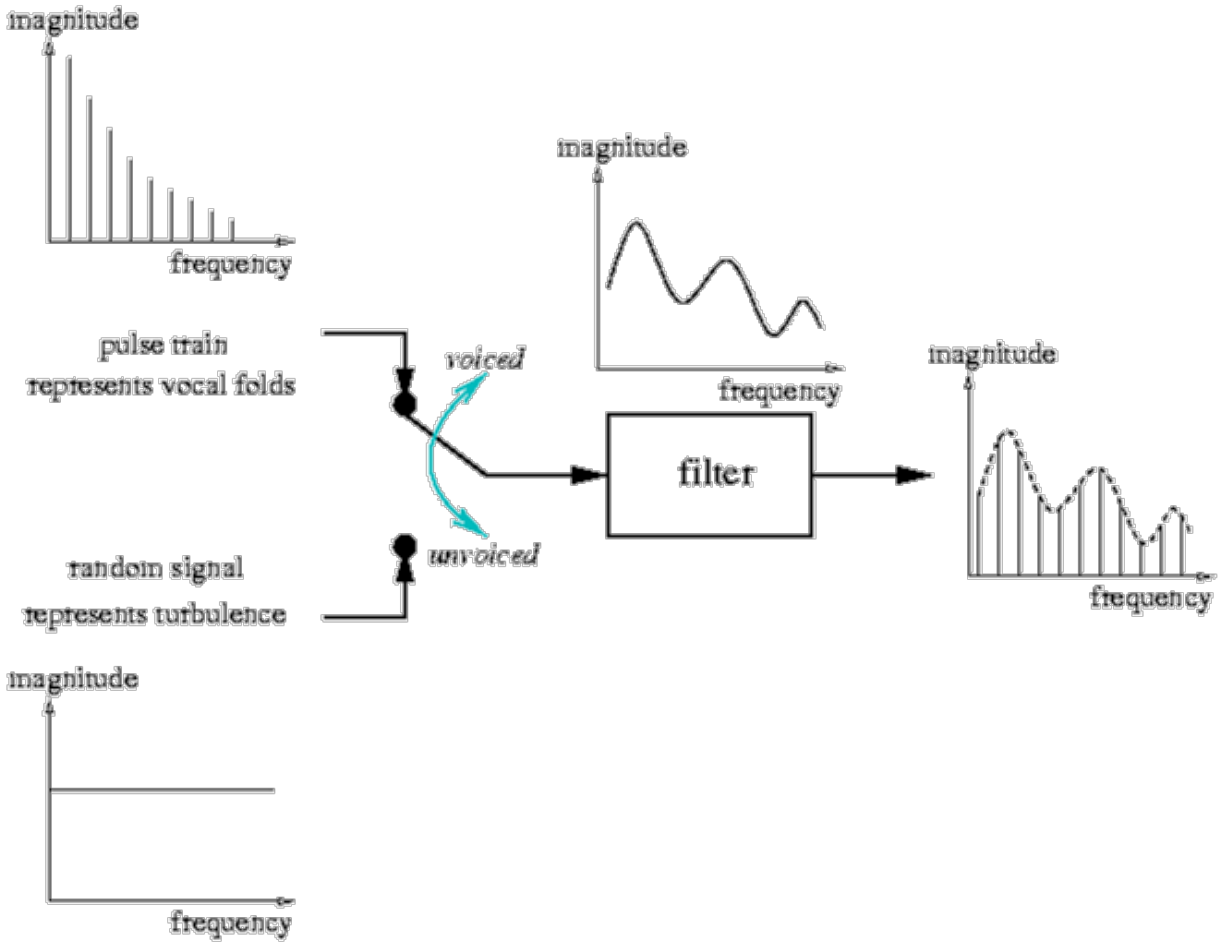
- Remember that
  - overall spectral shape is due to the vocal tract configuration
  - fine spectral detail (harmonics) is due to the source (vocal folds)



Spectral envelope: how do we represent it?

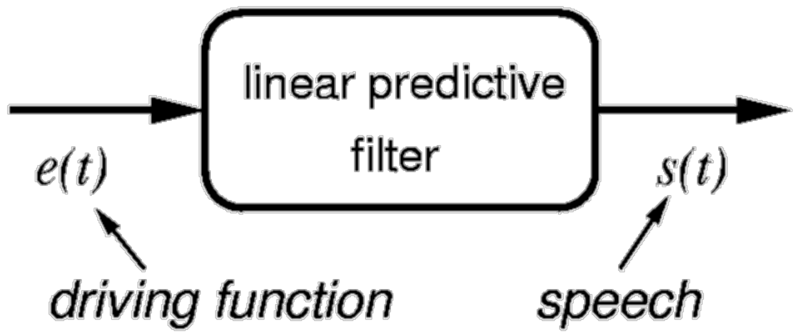
- What exactly is the representation for?
  - So we can modify the spectral shape independently of F0 / duration
- If we separate the source and filter, we could interpolate just the filter part (spectral shape), and manipulate F0 / duration independently
- How can we make this separation?

Reminder: the source-filter model



Linear prediction

- A simple form of filter



$$\hat{s}(t) = \sum_{j=1}^{p} a_j s(t-j)$$

$$s(t) = e(t) + \sum_{j=1}^{p} a_j s(t-j)$$

t = discrete time; p = filter order

Things we can do with linear prediction

- Because the filter represents only the vocal tract, we can use it to get a smooth spectrum
    - try this in Praat or Wavesurfer, by generating a spectrum using ‘LPC’ as the analysis type
- It is possible to convert from the filter parameters to vocal tract area, and so recover vocal tract shape from the acoustic signal
    - applications in speech therapy and acoustic phonetics
- Filter coefficients are a compact and slowly-changing representation of speech
    - can be used for coding and compression (e.g., sending speech over digital channels, like mobile phones)

Using LP for speech synthesis

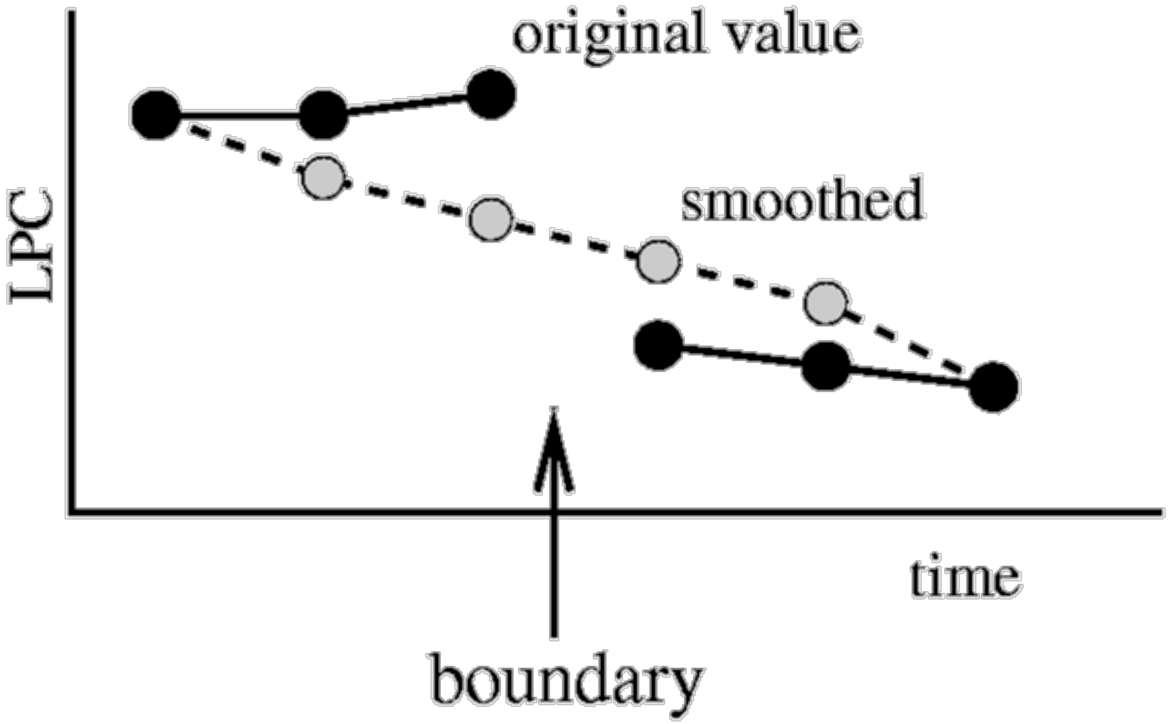
- Building the system
  - Record diphones
  - Perform linear predictive analysis
    - find the filter coefficients for each frame of speech
  - Store the sequence of LP filter coefficients (LPCs) for each diphone in a database
- Synthesising speech
  - Select the sequence of speech units (e.g., diphones)
  - Obtain the sequence of LPCs from the stored database
  - Re-synthesise the waveform using a LP filter + source

Modifying F0 and duration

- Since we now have an explicit source-filter model, this is now trivial
- Modifying F0:
  - Simply change the period of the voiced excitation signal
- Modifying duration:
  - Simply change the duration of the excitation signal

Can we disguise the joins better now?

- One reason for doing LPC analysis was to make smoothing around the concatenation points possible
- This is now simple too: we can smooth the filter coefficients, but leave the excitation signal alone. Consider a single filter coefficient (analogy - think of a formant frequency):



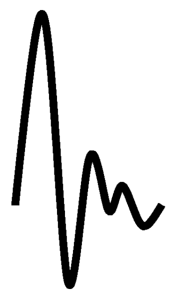
Pros & cons of linear prediction for synthesis

- Easy modification of pitch and duration
- Can smooth the spectral envelope over the joins
- Optionally, compressed storage of diphones
- Fast to compute

- Limited by source filter model
  - Linear predictive filter is not perfect - it's just an approximation
  - Idealised excitation (e.g., either voiced or unvoiced, not mixed)
- Signal processing artefacts
  - Can limit these by doing pitch-synchronous parameter update

How does TD-PSOLA separate source and filter

- Consider a single pitch period, as used by TD-PSOLA



- The shape of the waveform reflects the frequency response of the vocal tract. Stretching it in time would be like stretching the vocal tract in length
- So TD-PSOLA tries to keep the individual pitch periods unmodified
- To modify pitch, it must then slide the pitch periods over one another, hence “overlap-and-add”

Overcoming limitations of linear prediction

- We can overcome some of the limitations of LPC synthesis
- Multi-pulse LPC
    - Replace the voiced excitation signal with multiple pulses per pitch period (reduces the synthesis error)
- Residual excited LPC
    - Replace the voiced excitation signal with the actual error computed during LPC analysis (known as the residual), which leads to almost perfect reproduction of the original signal
- Festival uses residual excited LPC (RELP) by default
    - Near-perfect reconstruction is possible provided F0 and duration are not modified too far from their original values