CS 224S / LINGUIST 281
Speech Recognition, Synthesis, and Dialogue

Dan Jurafsky

Lecture 2: TTS: Brief History, Text Normalization and Part-of-Speech Tagging

IP Notice: lots of info, text, and diagrams on these slides comes (thanks!) from Alan Black's excellent lecture notes and from Richard Sproat's slides.

Outline

I. History of Speech Synthesis
II. State of the Art Demos
III. Brief Architectural Overview
IV. Text Processing
    1) Text Normalization
        - Tokenization
        - End of sentence detection
            - Methodology: decision trees
    2) Homograph disambiguation
    3) Part-of-speech tagging
        - Methodology: Hidden Markov Models

Dave Barry on TTS

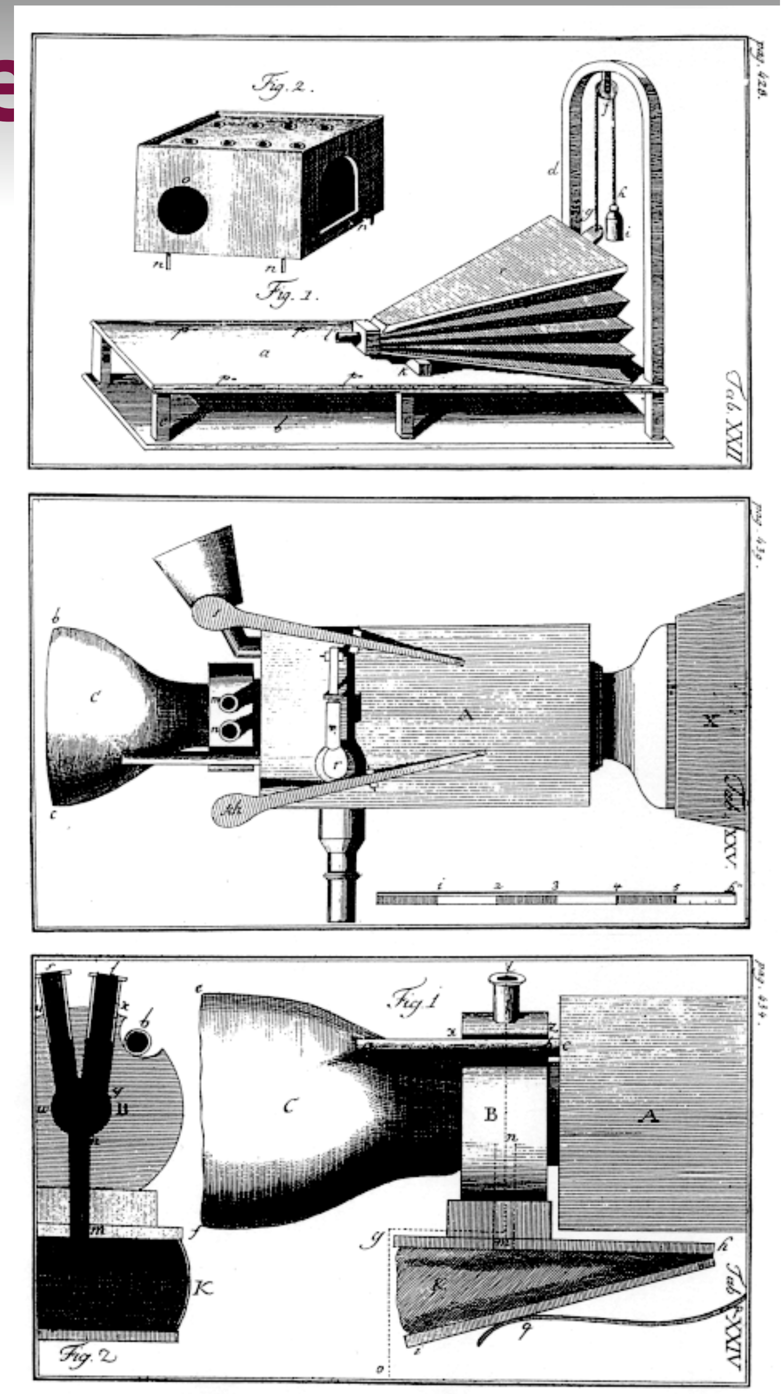
"And computers are getting smarter all the time; scientists tell us that soon they will be able to talk with us.

(By "they", I mean computers; I doubt scientists will ever be able to talk to us.)

History of TTS

- Pictures and some text from Hartmut Traunmüller's web site:
  - http://www.ling.su.se/staff/hartmut/kemplne.htm
- Von Kempeln 1780 b. Bratislava 1734 d. Vienna 1804
- Leather resonator manipulated by the operator to try and copy vocal tract configuration during sonorants (vowels, glides, nasals)
- Bellows provided air stream, counterweight provided inhalation
- Vibrating reed produced periodic pressure wave

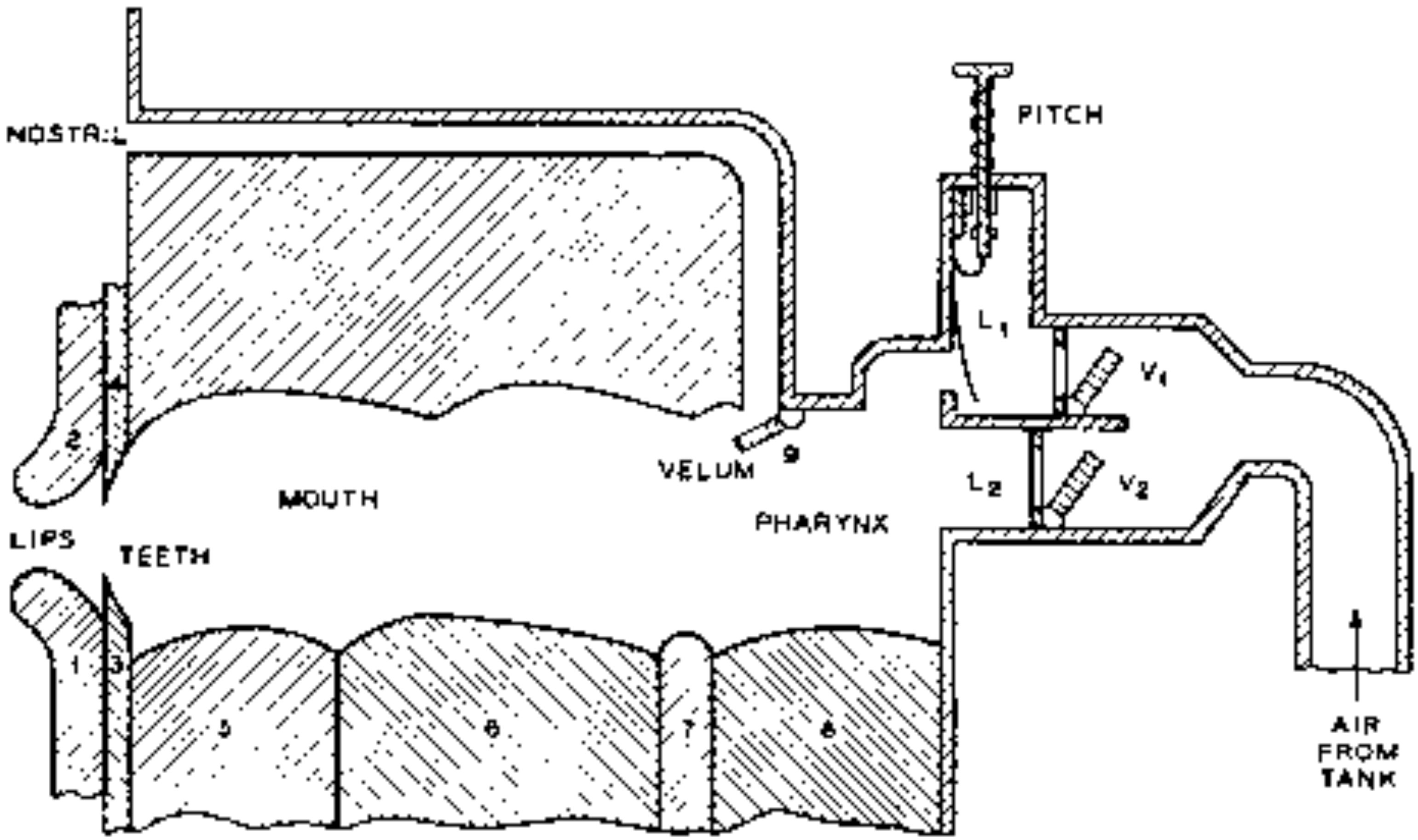
Von Kempe



- Small whistles controlled consonants
- Rubber mouth and nose; nose had to be covered with two fingers for non-nasals
- Unvoiced sounds: mouth covered, auxiliary bellows driven by string provides puff of air

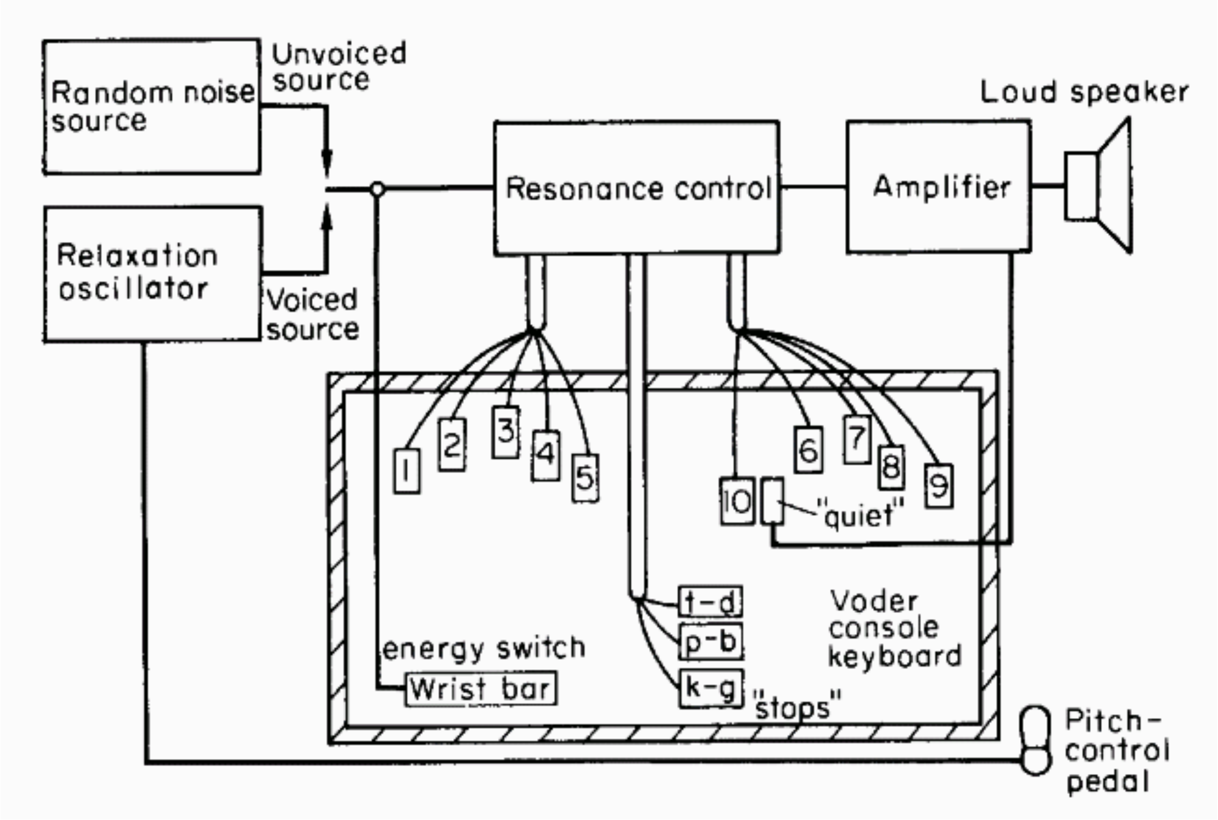
From Traunmüller's web site

Closer to a natural vocal tract: Riesz 1937



Homer Dudley 1939 VODER

- Synthesizing speech by electrical means
- 1939 World's Fair



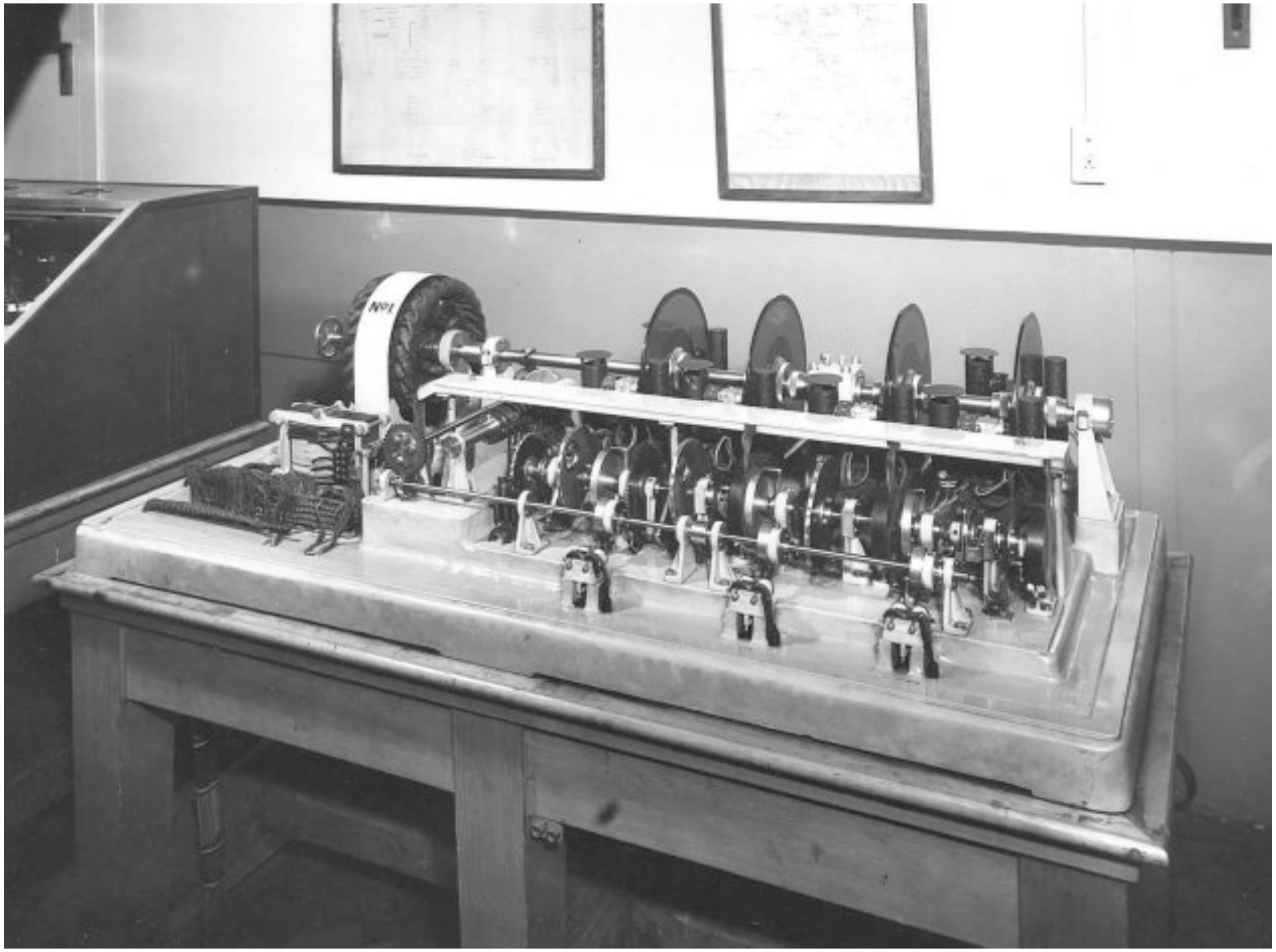
Homer Dudley’s VODER

- Manually controlled through complex keyboard
- Operator training was a problem

An aside on demos

- That last slide
- Exhibited Rule 1 of playing a speech synthesis demo:
- Always have a human say what the words are right before you have the system say them

The 1936 UK Speaking Clock



From http://web.ukonline.co.uk/freshwater/clocks/spkgclock.htm

The UK Speaking Clock

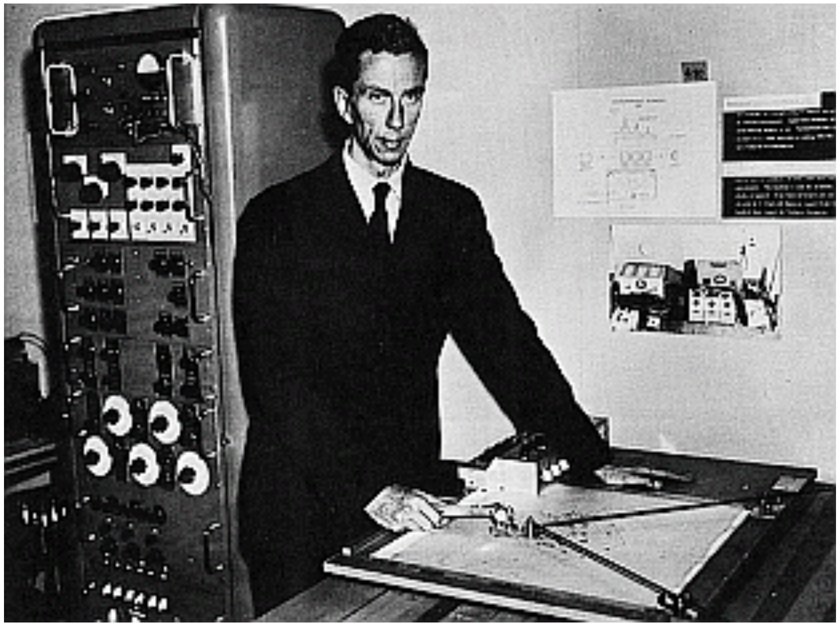
- July 24, 1936
- Photographic storage on 4 glass disks
- 2 disks for minutes, 1 for hour, one for seconds.
- Other words in sentence distributed across 4 disks, so all 4 used at once.
- Voice of “Miss J. Cain”

A technician adjusts the amplifiers of the first speaking clock

From http://web.ukonline.co.uk/freshwater/clocks/spkgclock.htm

Gunnar Fant's OVE synthesizer

- Of the Royal Institute of Technology, Stockholm
- Formant Synthesizer for vowels
- F1 and F2 could be controlled

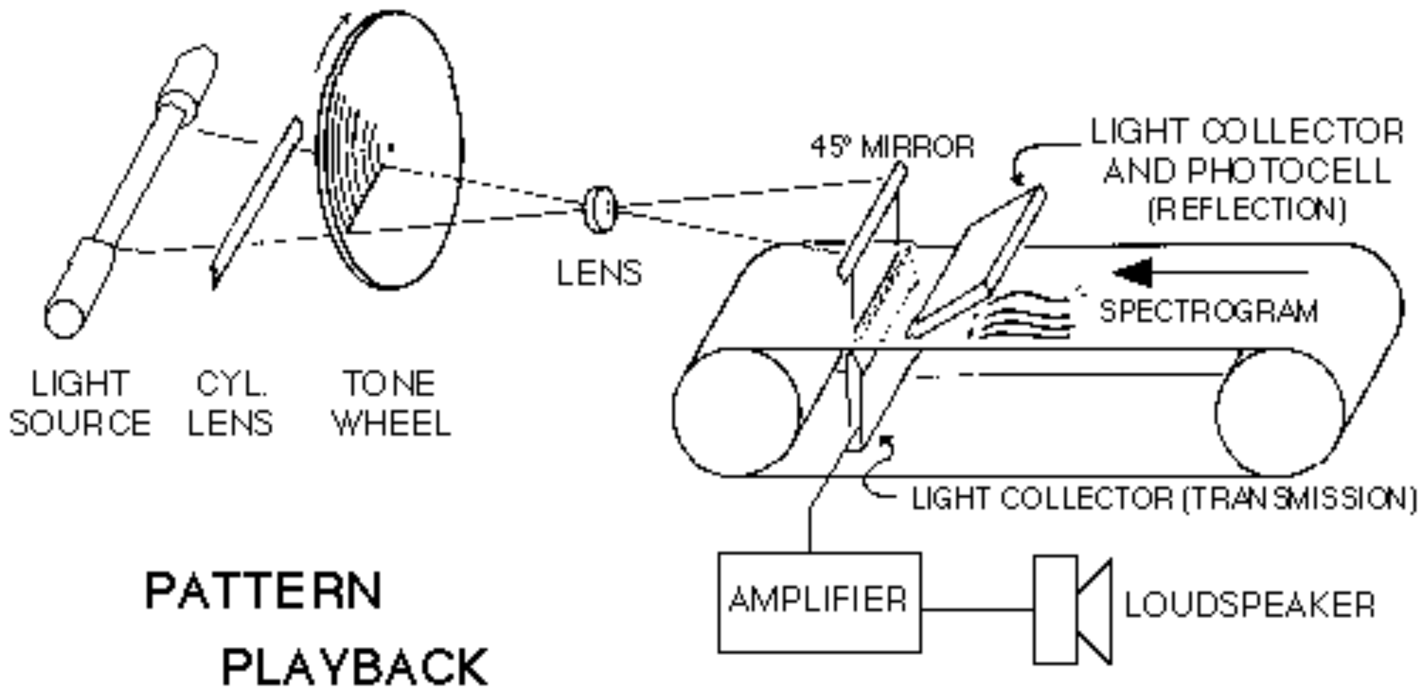


From Traunmüller's web site

Cooper's Pattern Playback

- Haskins Labs for investigating speech perception
- Works like an inverse of a spectrograph
- Light from a lamp goes through a rotating disk then through spectrogram into photovoltaic cells
- Thus amount of light that gets transmitted at each frequency band corresponds to amount of acoustic energy at that band

Cooper’s Pattern Playback



Modern TTS systems

- 1960's first full TTS: Umeda et al (1968)
- 1970's
  - Joe Olive 1977 concatenation of linear-prediction diphones
  - Texas Instruments Speak and Spell,
    - June 1978
    - Paul Breedlove

- 1980's
  - 1979 MIT MITalk (Allen, Hunnicut, Klatt)
- 1990's-present
  - Diphone synthesis
  - Unit selection synthesis
  - HMM synthesis

TTS Demos (Unit-Selection)

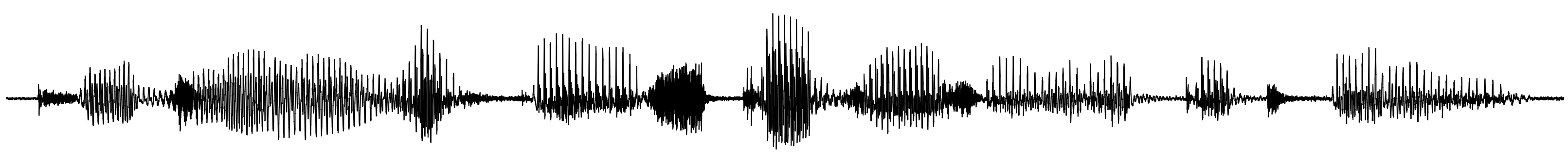
- ATT:
  - http://www.naturalvoices.att.com/demos/
- Festival
  - http://www-2.cs.cmu.edu/~awb/festival_demos/index.html
- Cepstral
  - http://www.cepstral.com/cgi-bin/demos/general
- IBM
  - http://www-306.ibm.com/software/pervasive/tech/demos/tts.shtml

Two steps

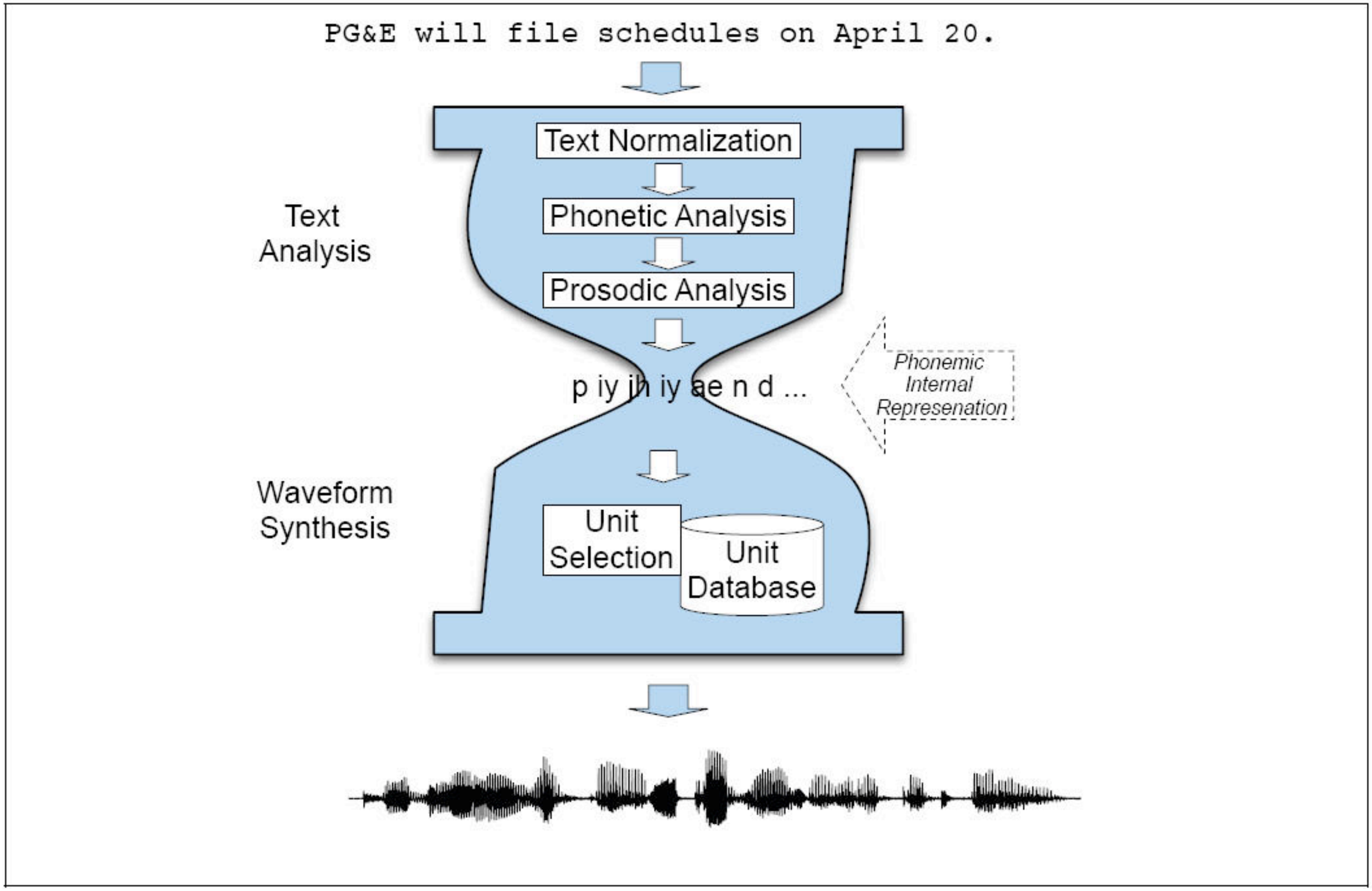
- `PG&E will file schedules on April 20.`
- TEXT ANALYSIS: Text into intermediate representation:

			*			*			* L-L%
P	G	AND	E	WILL	FILE	SCHEDULES	ON	APRIL	TWENTIETH
p iy	jh iy	ae n d	iy	w ih l	f ay l	s k eh jh ax l z	aa n	ey p r ih l	t w eh n t iy ax th

- WAVEFORM SYNTHESIS: From the intermediate representation into waveform



Architecture



Types of Waveform Synthesis

- Articulatory Synthesis:
  - Model movements of articulators and acoustics of vocal tract
- Formant Synthesis:
  - Start with acoustics, create rules/filters to create each formant
- Concatenative Synthesis:
  - Use databases of stored speech to assemble new utterances.
    - Diphone
    - Unit Selection
- Statistical (HMM) Synthesis
  - Trains parameters on databases of speech

Text modified from Richard Sproat slides

Formant Synthesis

- Were the most common commercial systems when computers were slow and had little memory.
- 1979 MIT MITalk (Allen, Hunnicut, Klatt)
- 1983 DECtalk system
  - "Perfect Paul" (The voice of Stephen Hawking)
  - "Beautiful Betty"

2nd Generation Synthesis

- Diphone Synthesis
  - Units are diphones; middle of one phone to middle of next.
  - Why? Middle of phone is steady state.
  - Record 1 speaker saying each diphone
  - ~1400 recordings
  - Paste them together and modify prosody.

3rd GenerationSynthesis

- All current commercial systems.

- Unit Selection Synthesis
  - Larger units of variable length
  - Record one speaker speaking 10 hours or more,
    - Have multiple copies of each unit
  - Use search to find best sequence of units

- Hidden Markov Model Synthesis
  - Train a statistical model on large amounts of data.

1. Text Normalization

- Analysis of raw text into pronounceable words:

He said the increase in credit limits helped B.C. Hydro achieve record net income of about $1 billion during the year ending March 31. This figure does not include any write-downs that may occur if Powerex determines that any of its customer accounts are not collectible. Cousins, however, was insistent that all debts will be collected: "We continue to pursue monies owing and we expect to be paid for electricity we have sold."

- Sentence Tokenization
- Text Normalization
  - Identify tokens in text
  - Chunk tokens into reasonably sized sections
  - Map tokens to words
  - Identify types for words

I. Text Processing

- He stole $100 million from the bank
- It’s 13 St. Andrews St.
- The home page is http://www.stanford.edu
- Yes, see you the following tues, that’s 11/12/01
- IV: four, fourth, I.V.
- IRA: I.R.A. or Ira
- 1750: seventeen fifty (date, address) or one thousand seven… (dollars)

I.1 Text Normalization Steps

- Identify tokens in text
- Chunk tokens
- Identify types of tokens
- Convert tokens to words

Step 1: identify tokens and chunk

- Whitespace can be viewed as separators
- Punctuation can be separated from the raw tokens
- Festival converts text into
  - ordered list of tokens
  - each with features:
    - its own preceding whitespace
    - its own succeeding punctuation

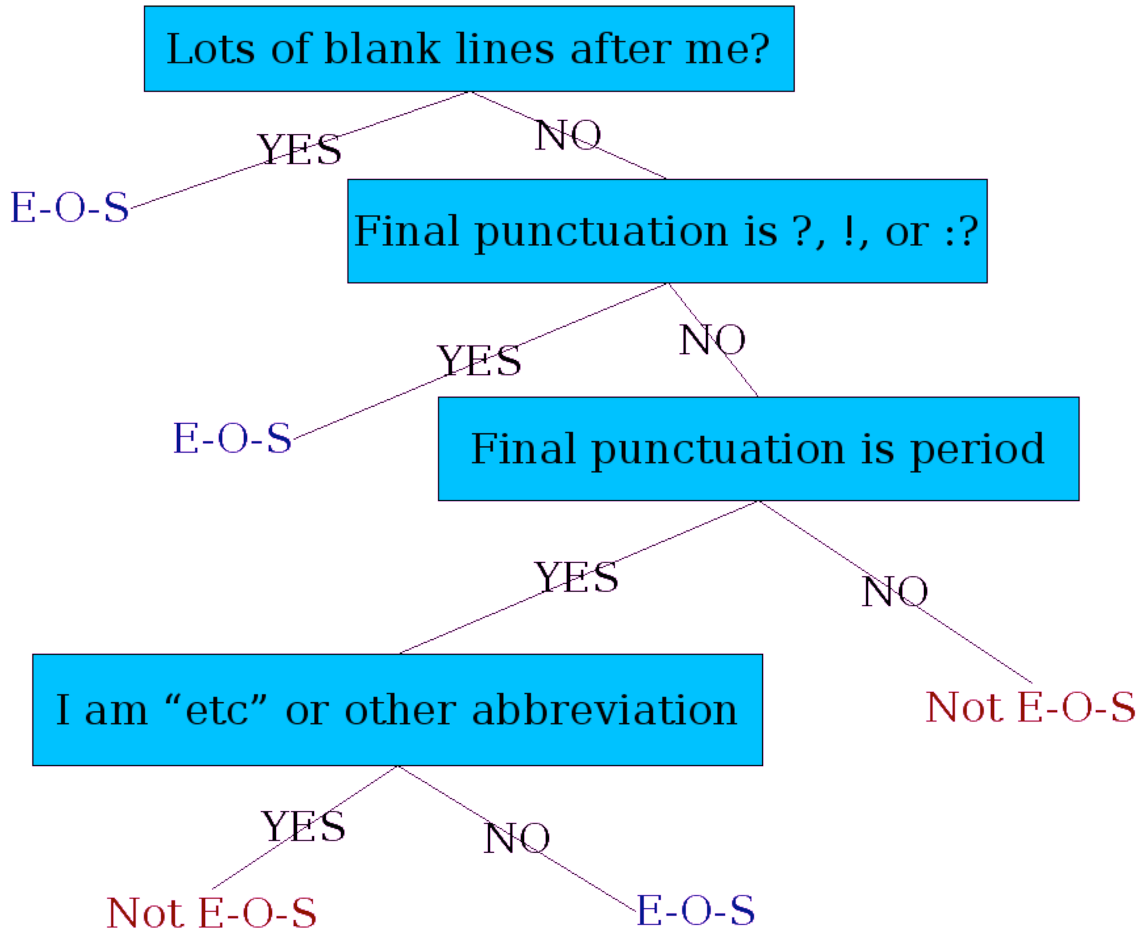
Important issue in tokenization: end-of-utterance detection

- Relatively simple if utterance ends in ?!
- But what about ambiguity of "."
- Ambiguous between end-of-utterance and end-of-abbreviation
  - My place on Winfield St. is around the corner.
  - I live at 151 Winfield St.
  - (Not "I live at 151 Winfield St..")
- How to solve this period-disambiguation task?

How about rules for end-of-utterance detection?

- A dot with one or two letters is an abbrev
- A dot with 3 cap letters is an abbrev.
- An abbrev followed by 2 spaces and a capital letter is an end-of-utterance
- Non-abbrevs followed by capitalized word are breaks

Determining if a word is end-of-utterance: a Decision Tree



CART

- Breiman, Friedman, Olshen, Stone. 1984. Classification and Regression Trees. Chapman & Hall, New York.
- Description/Use:
  - Binary tree of decisions, terminal nodes determine prediction ("20 questions")
  - If dependent variable is categorial, "classification tree",
  - If continuous, "regression tree"

Text from Richard Sproat

Determining end-of-utterance
The Festival hand-built decision tree

```
((n.whitespace matches ".*\n.*\n[ \n]*") ;; A significant break in text
  ((1))
  ((punc in ("?" ":" "!"))
   ((1))
   ((punc is ".")
    ;; This is to distinguish abbreviations vs periods
    ;; These are heuristics
    ((name matches "\\(.*\\..*\\|[A-Z][A-Za-z]?[A-Za-z]?\\|etc\\)")
     ((n.whitespace is " ")
      ((0))                 ;; if abbrev, single space enough for break
      ((n.name matches "[A-Z].*")
       ((1))
       ((0))))
     ((n.whitespace is " ")  ;; if it doesn't look like an abbreviation
      ((n.name matches "[A-Z].*")  ;; single sp. + non-cap is no break
       ((1))
       ((0)))
      ((1))))
    ((0)))))
```

The previous decision tree

- Fails for
  - Cog. Sci. Newsletter
  - Lots of cases at end of line.
  - Badly spaced/capitalized sentences

More sophisticated decision tree features

- Prob(word with "." occurs at end-of-s)
- Prob(word after "." occurs at begin-of-s)
- Length of word with "."
- Length of word after "."
- Case of word with ".": Upper, Lower, Cap, Number
- Case of word after ".": Upper, Lower, Cap, Number
- Punctuation after "." (if any)
- Abbreviation class of word with "." (month name, unit-of-measure, title, address name, etc)

From Richard Sproat slides

Learning DTs

- DTs are rarely built by hand
- Hand-building only possible for very simple features, domains
- Lots of algorithms for DT induction
- Covered in detail in Machine Learning or AI classes
  - Russell and Norvig AI text.
- I'll give quick intuition here

CART Estimation

- Creating a binary decision tree for classification or regression involves 3 steps:
  - Splitting Rules: Which split to take at a node?
  - Stopping Rules: When to declare a node terminal?
  - Node Assignment: Which class/value to assign to a terminal node?

From Richard Sproat slides

Splitting Rules

- Which split to take a node?
- Candidate splits considered:
  - Binary cuts: for continuous (-inf < x < inf) consider splits of form:
    - $X <= k$ **vs.** $x > k \ \forall K$
  - Binary partitions: For categorical $x \in \{1,2,\ldots\} = X$ consider splits of form:
  - $x \in A$ **vs.** $x \in X\text{-}A, \forall A \in X$

From Richard Sproat slides

Splitting Rules

- Choosing best candidate split.
  - Method 1: Choose k (continuous) or A (categorical) that minimizes estimated classification (regression) error after split
  - Method 2 (for classification): Choose k or A that minimizes estimated entropy after that split.

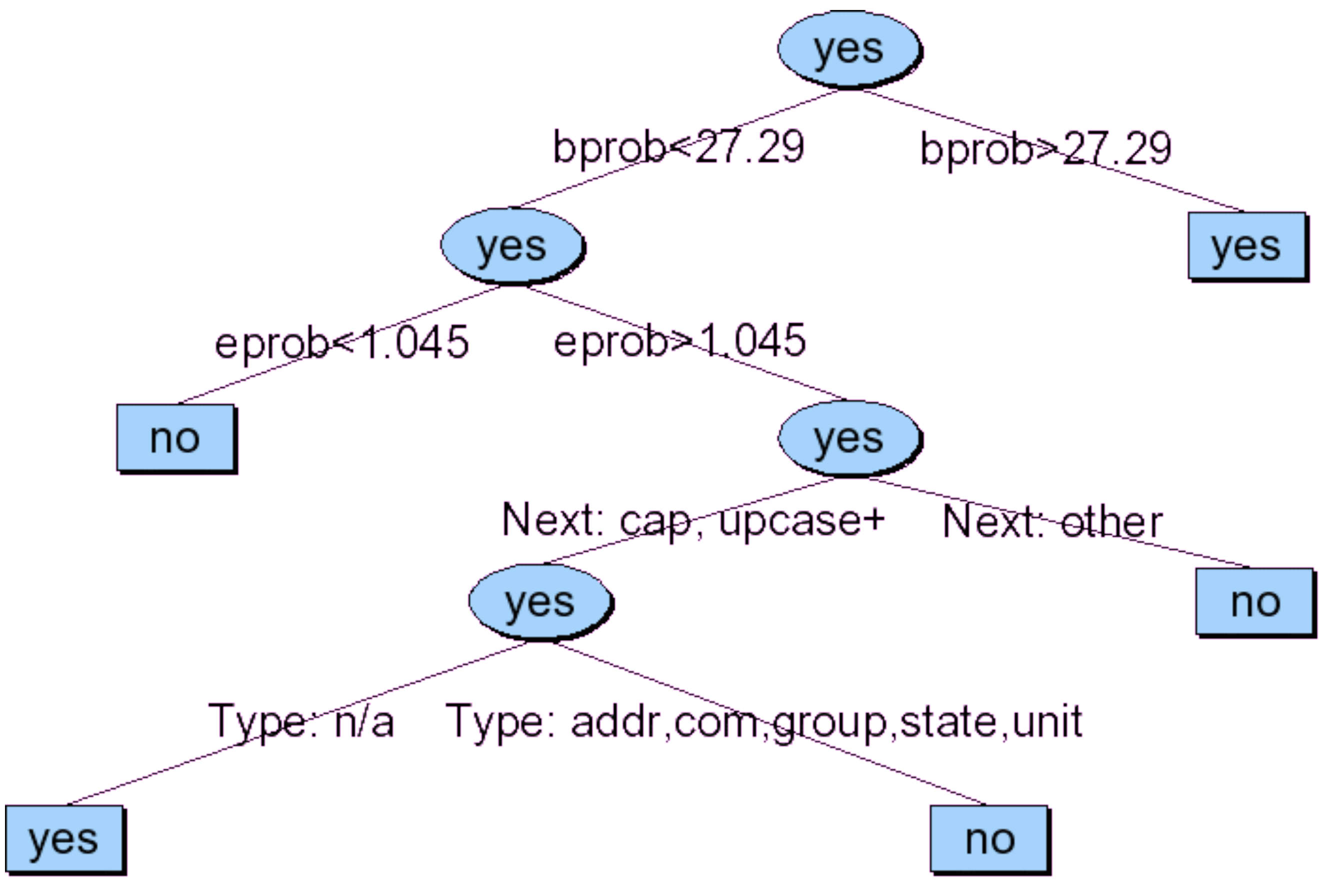
From Richard Sproat slides

Decision Tree Stopping

- When to declare a node terminal?
- Strategy (Cost-Complexity pruning):
  1. Grow over-large tree
  2. Form sequence of subtrees, T0…Tn ranging from full tree to just the root node.
  3. Estimate "honest" error rate for each subtree.
  4. Choose tree size with minimum "honest" error rate.
- To estimate "honest" error rate, test on data different from training data (i.e. grow tree on 9/10 of data, test on 1/10, repeating 10 times and averaging (cross-validation).

From Richard Sproat

Sproat EOS tree



From Richard Sproat slides

Summary on end-of-sentence detection

- Best references:
  - David Palmer and Marti Hearst. 1997. Adaptive Multilingual Sentence Boundary Disambiguation. Computational Linguistics 23, 2. 241-267.
  - David Palmer. 2000. Tokenisation and Sentence Segmentation. In "Handbook of Natural Language Processing", edited by Dale, Moisl, Somers.

Steps 3+4: Identify Types of Tokens, and Convert Tokens to Words

- Pronunciation of numbers often depends on type. 3 ways to pronounce 1776:
  - 1776 date: seventeen seventy six.
  - 1776 phone number: one seven seven six
  - 1776 quantifier: one thousand seven hundred (and) seventy six
  - Also:
    - 25 day: twenty-fifth

Festival rule for dealing with "$1.2 million"

```
(define (token_to_words utt token name)
 (cond
  ((and (string-matches name "\\$[0-9,]+\\(\\.[0-9]+\\)?")
        (string-matches (utt.streamitem.feat utt token "n.name")
                        ".*illion.?"))
   (append
    (builtin_english_token_to_words utt token (string-after name "$"))
    (list
     (utt.streamitem.feat utt token "n.name"))))
  ((and (string-matches (utt.streamitem.feat utt token "p.name")
                        "\\$[0-9,]+\\(\\.[0-9]+\\)?")
        (string-matches name ".*illion.?"))
   (list "dollars"))
  (t
   (builtin_english_token_to_words utt token name))))
```

Rule-based versus machine learning

- As always, we can do things either way, or more often by a combination
- Rule-based:
  - Simple
  - Quick
  - Can be more robust
- Machine Learning
  - Works for complex problems where rules hard to write
  - Higher accuracy in general
  - But worse generalization to very different test sets
- Real TTS and NLP systems
  - Often use aspects of both.

Machine learning method for Text Normalization

- From 1999 Hopkins summer workshop "Normalization of Non-Standard Words"
  - Sproat, R., Black, A., Chen, S., Kumar, S., Ostendorf, M., and Richards, C. 2001. Normalization of Non-standard Words, Computer Speech and Language, 15(3):287-333
- NSW examples:
  - Numbers:
    - 123, 12 March 1994
  - Abrreviations, contractions, acronyms:
    - approx., mph. ctrl-C, US, pp, lb
  - Punctuation conventions:
    - 3-4, +/-, and/or
  - Dates, times, urls, etc

How common are NSWs?

- Varies over text type
- Word not in lexicon, or with non-alphabetic characters:

Text Type	% NSW
novels	**1.5%**
press wire	**4.9%**
e-mail	**10.7%**
recipes	**13.7%**
classified	**17.9%**

From Alan Black slides

How hard are NSWs?

- Identification:
  - Some homographs "Wed", "PA"
  - False positives: OOV
- Realization:
  - Simple rule: money, $2.34
  - Type identification+rules: numbers
  - Text type specific knowledge (in classified ads, BR for bedroom)
- Ambiguity (acceptable multiple answers)
  - "D.C." as letters or full words
  - "MB" as "meg" or "megabyte"
  - 250

Step 1: Splitter

- Letter/number conjunctions (WinNT, SunOS, PC110)
- Hand-written rules in two parts:
  - Part I: group things not to be split (numbers, etc; including commas in numbers, slashes in dates)
  - Part II: apply rules:
    - At transitions from lower to upper case
    - After penultimate upper-case char in transitions from upper to lower
    - At transitions from digits to alpha
    - At punctuation

From Alan Black Slides

Step 2: Classify token into 1 of 20 types

- EXPN: abbrev, contractions (adv, N.Y., mph, gov't)
- LSEQ: letter sequence (CIA, D.C., CDs)
- ASWD: read as word, e.g. CAT, proper names
- MSPL: misspelling
- NUM: number (cardinal) (12,45,1/2, 0.6)
- NORD: number (ordinal) e.g. May 7, 3rd, Bill Gates II
- NTEL: telephone (or part) e.g. 212-555-4523
- NDIG: number as digits e.g. Room 101
- NIDE: identifier, e.g. 747, 386, I5, PC110
- NADDR: number as stresst address, e.g. 5000 Pennsylvania
- NZIP, NTIME, NDATE, NYER, MONEY, BMONY, PRCT,URL,etc
- SLNT: not spoken (KENT*REALTY)

More about the types

- 4 categories for alphabetic sequences:
  - EXPN: expand to full word or word seq (fplc for fireplace, NY for New York)
  - LSEQ: say as letter sequence (IBM)
  - ASWD: say as standard word (either OOV or acronyms)
- 5 main ways to read numbers:
  - Cardinal (quantities)
  - Ordinal (dates)
  - String of digits (phone numbers)
  - Pair of digits (years)
  - Trailing unit: serial until last non-zero digit: 8765000 is "eight seven six five thousand" (some phone numbers, long addresses)
  - But still exceptions: (947-3030, 830-7056)

Type identification algorithm

- Create large hand-labeled training set and build a DT to predict type
- Example of features in tree for subclassifier for alphabetic tokens:
  - P(t|o) = p(o|t)p(t)/p(o)
  - P(o|t), for t in ASWD, LSWQ, EXPN (from trigram letter model)

$$p(o \mid t) = \sum_{i=1}^{N} p(l_{i1} \mid l_{i-1}, l_{i-2})$$

  - P(t) from counts of each tag in text
  - P(o) normalization factor

Type identification algorithm

- Hand-written context-dependent rules:
  - List of lexical items (Act, Advantage, amendment) after which Roman numbers read as cardinals not ordinals
- Classifier accuracy:
  - 98.1% in news data,
  - 91.8% in email

Step 3: expanding NSW Tokens

- Type-specific heuristics
  - ASWD expands to itself
  - LSEQ expands to list of words, one for each letter
  - NUM expands to string of words representing cardinal
  - NYER expand to 2 pairs of NUM digits…
  - NTEL: string of digits with silence for puncutation
  - Abbreviation:
    - use abbrev lexicon if it's one we've seen
    - Else use training set to know how to expand
    - Cute idea: if "eat in kit" occurs in text, "eat-in kitchen" will also occur somewhere.

What about unseen abbreviations?

- Problem: given a previously unseen abbreviation, how do you use corpus-internal evidence to find the expansion into a standard word?
- Example:
  - Cus wnt info on services and chrgs
- Elsewhere in corpus:
  - …customer wants…
  - …wants info on vmail…

From Richard Sproat

4 steps to Sproat et al. algorithm

1) Splitter (on whitespace or also within word ("AltaVista")
2) Type identifier: for each split token identify type
3) Token expander: for each typed token, expand to words
   - Deterministic for number, date, money, letter sequence
   - Only hard (nondeterministic) for abbreviations
4) Language Model: to select between alternative pronunciations

From Alan Black slides

I.2 Homograph disambiguation

It's no use (/y uw s/) to ask to use (/y uw z/) the telephone.

Do you live (/l ih v/) near a zoo with live (/l ay v/) animals?

I prefer bass (/b ae s/) fishing to playing the bass (/b ey s/) guitar.

Final voicing			Stress shift			-ate final vowel		
	N (/s/)	**V** (/z/)		**N** (init. stress)	**V** (fin. stress)		**N/A** (final /ax/)	**V** (final /ey/)
use	y uw s	y uw z	record	r eh1 k axr0 d	r ix0 k ao1 r d	estimate	eh s t ih m ax t	eh s t ih m ey t
close	k l ow s	k l ow z	insult	ih1 n s ax0 l t	ix0 n s ah1 l t	separate	s eh p ax r ax t	s eh p ax r ey t
house	h aw s	h aw z	object	aa1 b j eh0 k t	ax0 b j eh1 k t	moderate	m aa d ax r ax t	m aa d ax r ey t

I.2 Homograph disambiguation

19 most frequent homographs, from Liberman and Church

use	319
increase	230
close	215
record	195
house	150
contract	143
lead	131
live	130
lives	105
protest	94

survey	91
project	90
separate	87
present	80
read	72
subject	68
rebel	48
finance	46
estimate	46

Not a huge problem, but still important

POS Tagging for homograph disambiguation

- Many homographs can be distinguished by POS
  - use y uw s y uw z
  - close k l ow s k l ow z
  - house h aw s h aw z
  - live l ay v l ih v
  - REcord reCORD
  - INsult inSULT
  - OBject obJECT
  - OVERflow overFLOW
  - DIScount disCOUNT
  - CONtent conTENT
- POS tagging also useful for CONTENT/FUNCTION distinction, which is useful for phrasing

Part of speech tagging

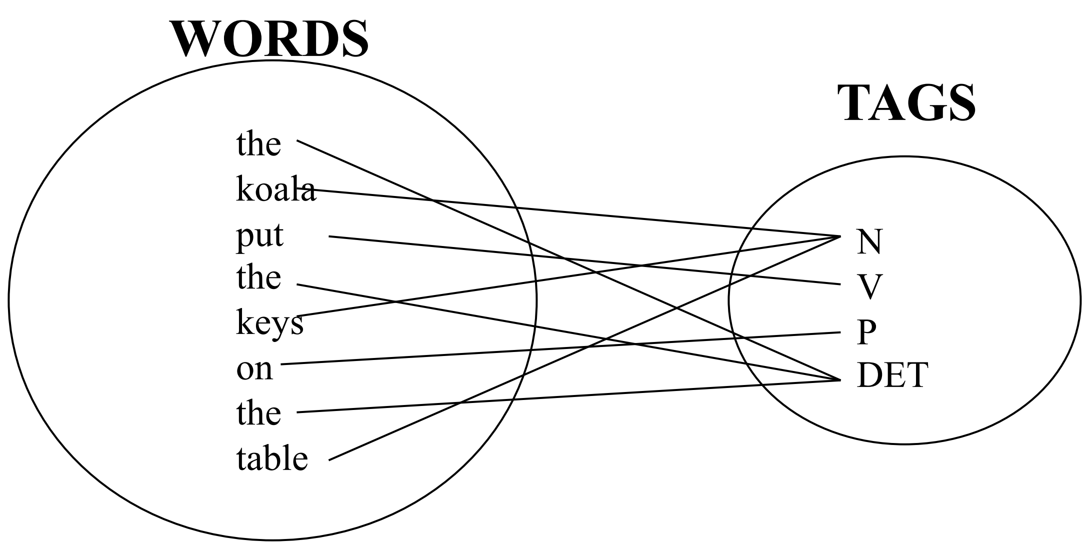
- 8 (ish) traditional parts of speech
  - Noun, verb, adjective, preposition, adverb, article, interjection, pronoun, conjunction, etc
  - This idea has been around for over 2000 years (Dionysius Thrax of Alexandria, c. 100 B.C.)
  - Called: parts-of-speech, lexical category, word classes, morphological classes, lexical tags, POS
  - We'll use POS most frequently
  - I'll assume that you all know what these are

POS examples

N	noun	*chair, bandwidth, pacing*
V	verb	*study, debate, munch*
ADJ	adj	*purple, tall, ridiculous*
ADV	adverb	*unfortunately, slowly,*
P	preposition	*of, by, to*
PRO	pronoun	*I, me, mine*
DET	determiner	*the, a, that, those*

POS Tagging: Definition

- The process of assigning a part-of-speech or lexical class marker to each word in a corpus:



POS Tagging example

WORD	tag
the	**DET**
koala	**N**
put	**V**
the	**DET**
keys	**N**
on	**P**
the	**DET**
table	**N**

POS tagging: Choosing a tagset

- There are so many parts of speech, potential distinctions we can draw
- To do POS tagging, need to choose a standard set of tags to work with
- Could pick very coarse tagets
  - N, V, Adj, Adv.
- More commonly used set is finer grained, the "UPenn TreeBank tagset", 45 tags
  - PRP$, WRB, WP$, VBG
- Even more fine-grained tagsets exist

Penn TreeBank POS Tag set

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two, three*	TO	“to”	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential ‘there’	*there*	VB	verb, base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb, past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb, gerund	*eating*
JJ	adjective	*yellow*	VBN	verb, past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb, non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb, 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, singular	*IBM*	$	dollar sign	$
NNPS	proper noun, plural	*Carolinas*	#	pound sign	#
PDT	predeterminer	*all, both*	“	left quote	‘ or “
POS	possessive ending	*’s*	”	right quote	’ or ”
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	*your, one’s*	)	right parenthesis	],), }, >
RB	adverb	*quickly, never*	,	comma	,
RBR	adverb, comparative	*faster*	.	sentence-final punc	. ! ?
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	: ; ... – -
RP	particle	*up, off*			

Using the UPenn tagset

- The/DT grand/JJ jury/NN commmented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.
- Prepositions and subordinating conjunctions marked IN ("although/IN I/PRP..")
- Except the preposition/complementizer "to" is just marked "to".

POS Tagging

- Words often have more than one POS: *back*
  - The *back* door = JJ
  - On my *back* = NN
  - Win the voters *back* = RB
  - Promised to *back* the bill = VB
- The POS tagging problem is to determine the POS tag for a particular instance of a word.

These examples from Dekang Lin

How hard is POS tagging? Measuring ambiguity

		Original 87-tag corpus		Treebank 45-tag corpus	
Unambiguous (1 tag)		**44,019**		**38,857**	
Ambiguous (2–7 tags)		**5,490**		**8844**	
Details:	2 tags	4,967		6,731	
	3 tags	411		1621	
	4 tags	91		357	
	5 tags	17		90	
	6 tags	2	(*well*, *beat*)	32	
	7 tags	2	(*still*, *down*)	6	(*well*, *set*, *round*, *open*, *fit*, *down*)
	8 tags			4	(*'s*, *half*, *back*, *a*)
	9 tags			3	(*that*, *more*, *in*)

3 methods for POS tagging

1. Rule-based tagging
   - (ENGTWOL)
2. Stochastic (=Probabilistic) tagging
   - HMM (Hidden Markov Model) tagging
3. Transformation-based tagging
   - Brill tagger

Hidden Markov Model Tagging

- Using an HMM to do POS tagging
- Is a special case of Bayesian inference
  - Foundational work in computational linguistics
  - Bledsoe 1959: OCR
  - Mosteller and Wallace 1964: authorship identification
- It is also related to the "noisy channel" model that we'll do when we do ASR (speech recognition)

POS tagging as a sequence classification task

- We are given a sentence (an "observation" or "sequence of observations")
  - Secretariat is expected to race tomorrow
- What is the best sequence of tags which corresponds to this sequence of observations?
- Probabilistic view:
  - Consider all possible sequences of tags
  - Out of this universe of sequences, choose the tag sequence which is most probable given the observation sequence of n words w1…wn.

Getting to HMM

- We want, out of all sequences of n tags $t_1 \dots t_n$ the single tag sequence such that $P(t_1 \dots t_n | w_1 \dots w_n)$ is highest.

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$$

- Hat ^ means "our estimate of the best one"
- Argmax_x f(x) means "the x such that f(x) is maximized"

Getting to HMM

- This equation is guaranteed to give us the best tag sequence

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$$

- But how to make it operational? How to compute this value?
- Intuition of Bayesian classification:
  - Use Bayes rule to transform into a set of other probabilities that are easier to compute

Using Bayes Rule

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n|t_1^n)P(t_1^n)$$

Likelihood and prior

$$\hat{t}_1^n = \operatorname*{argmax}_{t_1^n} \overbrace{P(w_1^n|t_1^n)}^{\text{likelihood}} \; \overbrace{P(t_1^n)}^{\text{prior}}$$

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i)$$

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1})$$

$$\hat{t}_1^n = \operatorname*{argmax}_{t_1^n} P(t_1^n|w_1^n) \approx \operatorname*{argmax}_{t_1^n} \prod_{i=1}^{n} P(w_i|t_i) P(t_i|t_{i-1})$$

Two kinds of probabilities (1)

- Tag transition probabilities $p(t_i|t_{i-1})$
  - Determiners likely to precede adjs and nouns
    - That/DT flight/NN
    - The/DT yellow/JJ hat/NN
    - So we expect P(NN|DT) and P(JJ|DT) to be high
    - But P(DT|JJ) to be:
  - Compute P(NN|DT) by counting in a labeled corpus:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

$$P(NN|DT) = \frac{C(DT, NN)}{C(DT)} = \frac{56,509}{116,454} = .49$$

Two kinds of probabilities (2)

- Word likelihood probabilities $p(w_i|t_i)$
  - VBZ (3sg Pres verb) likely to be "is"
  - Compute P(is|VBZ) by counting in a labeled corpus:

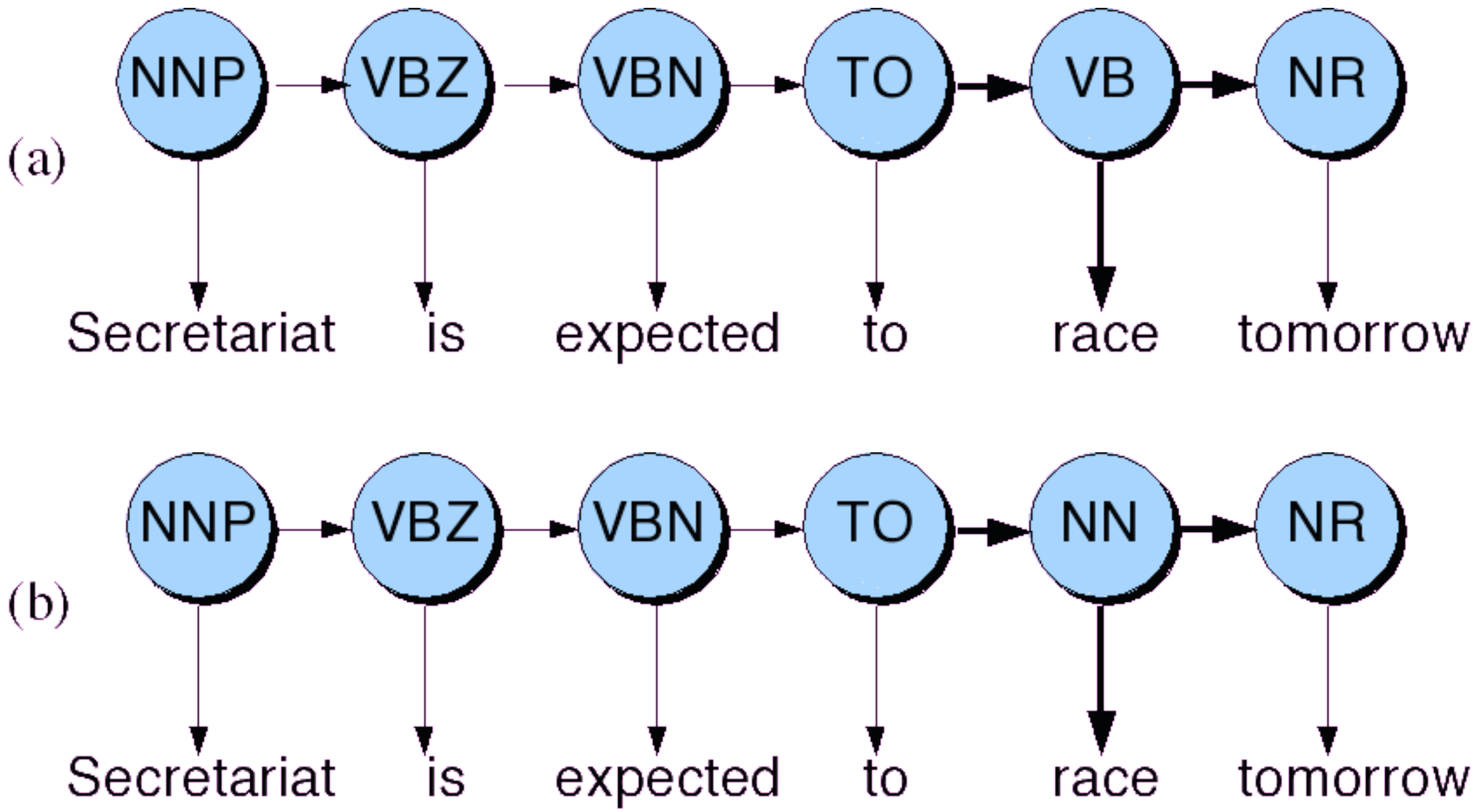
$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

$$P(is|VBZ) = \frac{C(VBZ, is)}{C(VBZ)} = \frac{10,073}{21,627} = .47$$

An Example: the verb "race"

- Secretariat/NNP is/VBZ expected/VBN to/TO **race**/VB tomorrow/NR
- People/NNS continue/VB to/TO inquire/VB the/DT reason/NN for/IN the/DT **race**/NN for/IN outer/JJ space/NN
- How do we pick the right tag?

Disambiguating "race"



- $P(NN|TO) = .00047$
- $P(VB|TO) = .83$
- $P(race|NN) = .00057$
- $P(race|VB) = .00012$
- $P(NR|VB) = .0027$
- $P(NR|NN) = .0012$

- $P(VB|TO)P(NR|VB)P(race|VB) = .00000027$
- $P(NN|TO)P(NR|NN)P(race|NN)=.00000000032$

- So we (correctly) choose the verb reading

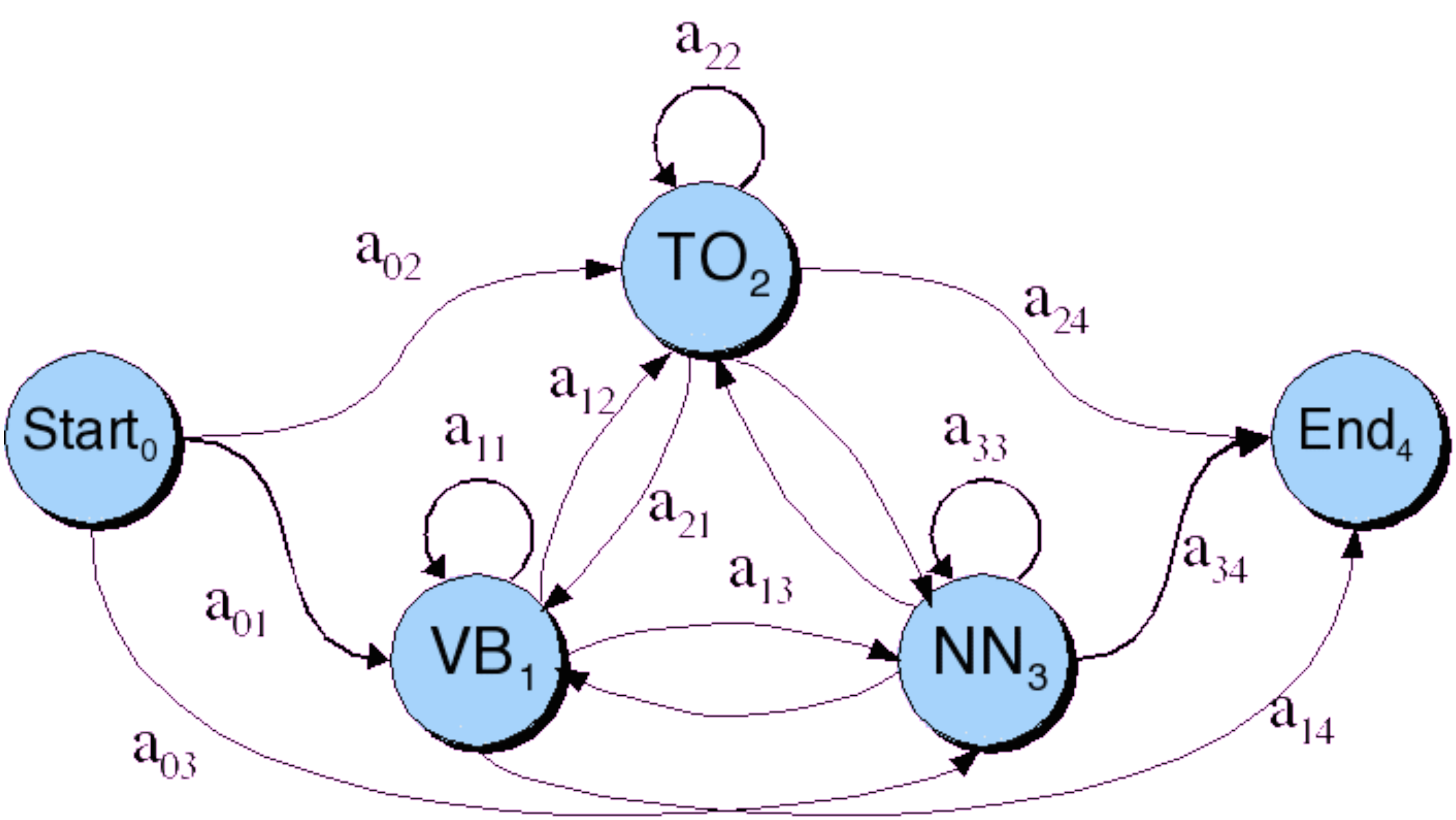
Hidden Markov Models

- What we've described with these two kinds of probabilities is a Hidden Markov Model
- A Hidden Markov Model is a particular probabilistic kind of automaton
- Let's just spend a bit of time tying this into the model
- We'll return to this in much more detail in 3 weeks when we do ASR

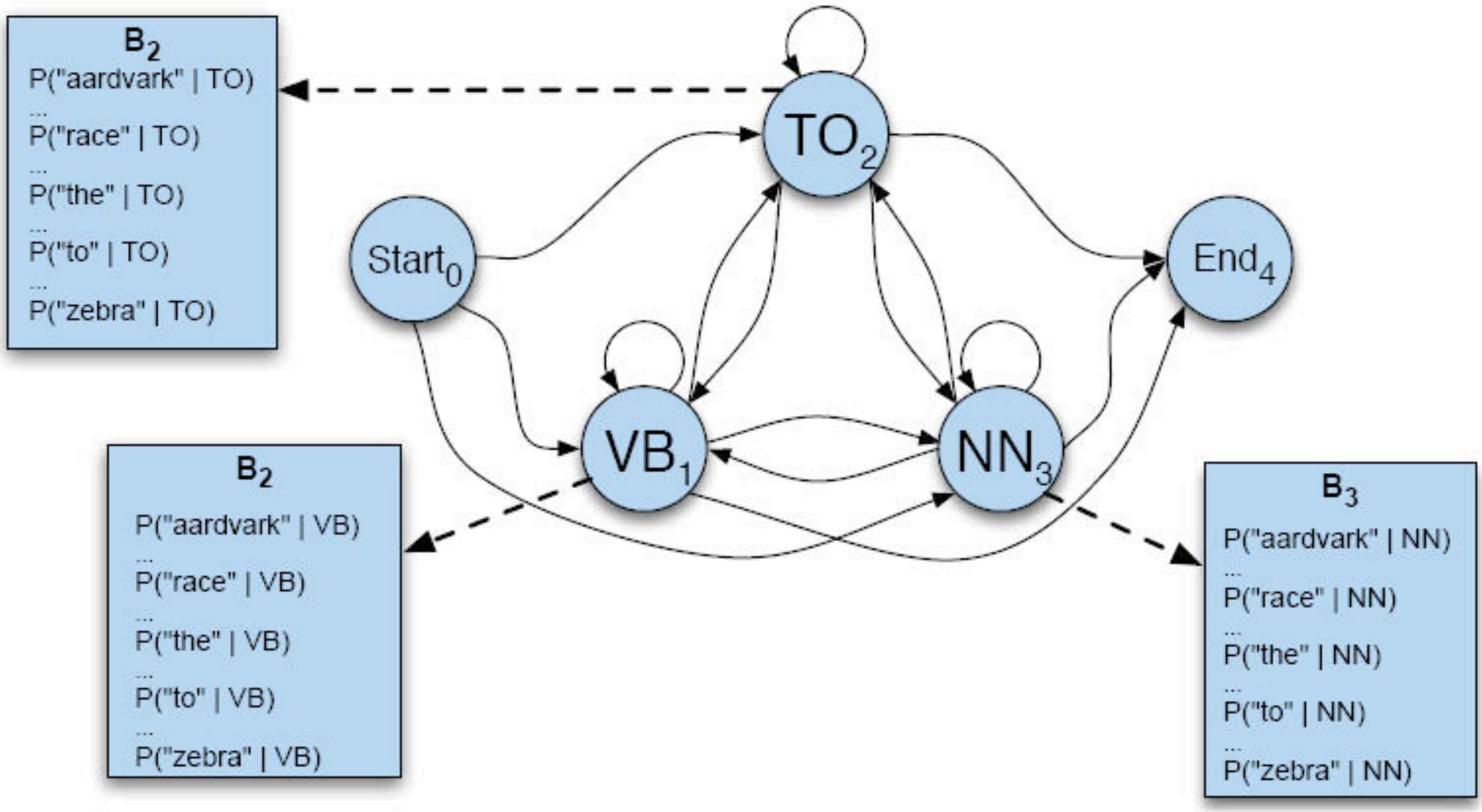
Hidden Markov Model

$Q = q_1 q_2 \ldots q_N$	a set of N **states**.
$A = a_{11} a_{12} \ldots a_{n1} \ldots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$.
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$.
$B = b_i(o_t)$	A sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i.
q_0, q_F	a special **start state** and **end (final) state** that are not associated with observations, together with transition probabilities $a_{01} a_{02} \ldots a_{0n}$ out of the start state and $a_{1F} a_{2F} \ldots a_{nF}$ into the end state.

Transitions between the hidden states of HMM, showing A probs



B observation likelihoods for POS HMM



The A matrix for the POS HMM

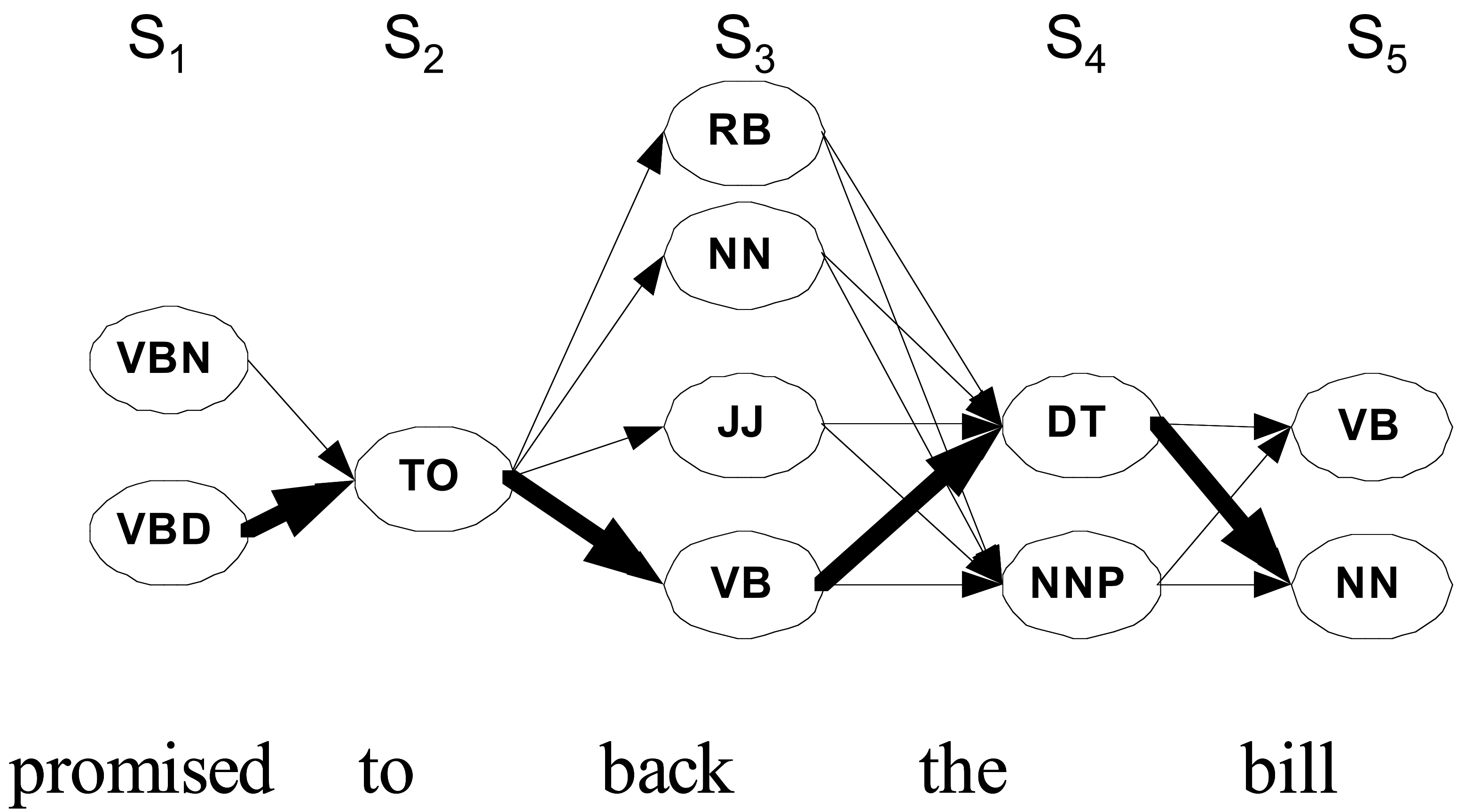
-

	VB	TO	NN	PPSS
<s>	.019	.0043	.041	.067
VB	.0038	.035	.047	.0070
TO	.83	0	.00047	0
NN	.0040	.016	.087	.0045
PPSS	.23	.00079	.0012	.00014

The B matrix for the POS HMM

	I	want	to	race
VB	0	.0093	0	.00012
TO	0	0	.99	0
NN	0	.000054	0	.00057
PPSS	.37	0	0	0

Viterbi intuition: we are looking for the best 'path'



Slide from Dekang Lin

The Viterbi Algorithm

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*

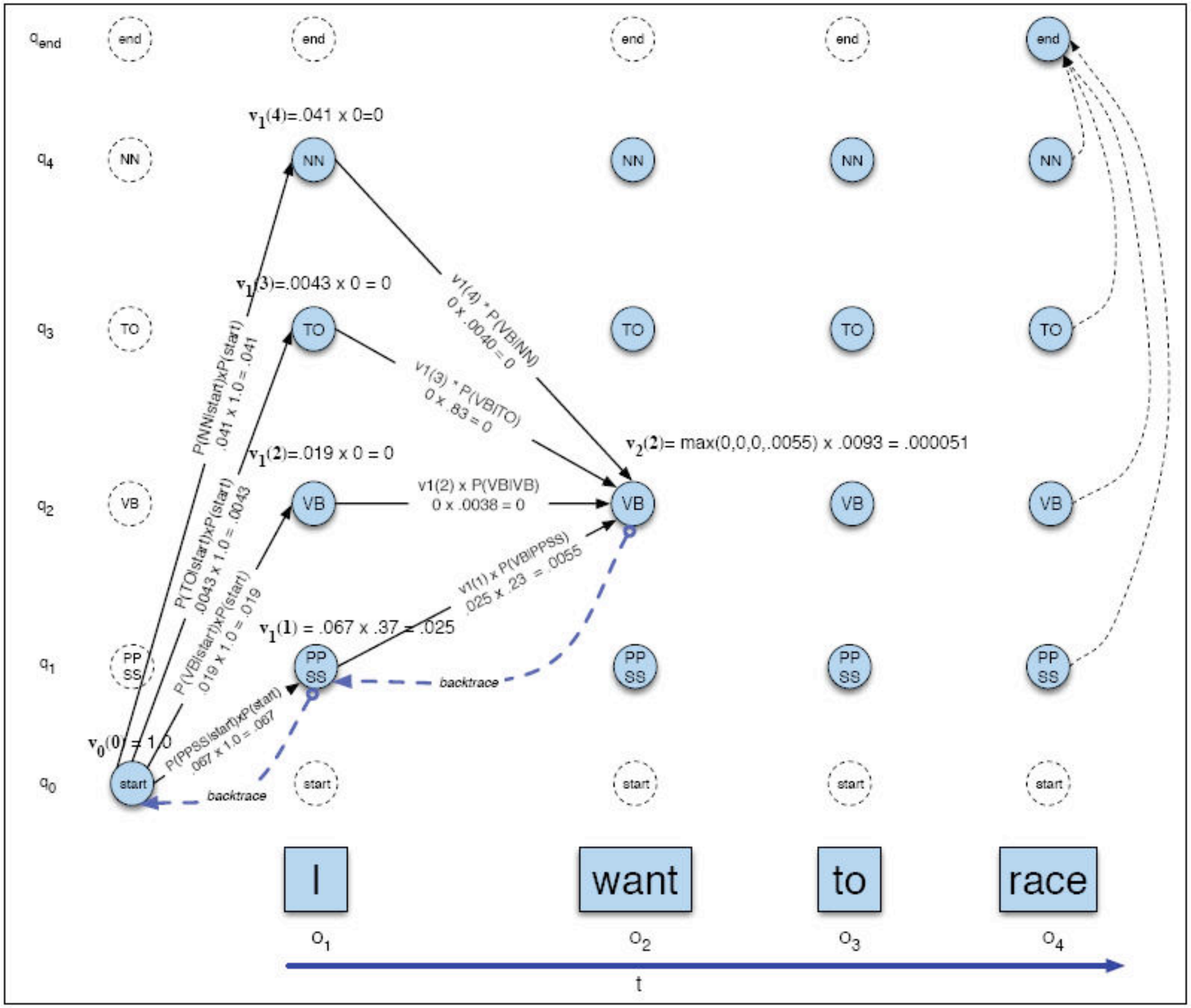
create a path probability matrix *viterbi[N+2,T]*
for each state s **from** 1 **to** N **do** ; initialization step
$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
$backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
for each state s **from** 1 **to** N **do**
$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$
$$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$$
$$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F} \quad \text{; termination step}$$
$$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F} \quad \text{; termination step}$$
return the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$

Intuition

- The value in each cell is computed by taking the MAX over all paths that lead to this cell.
- $$v_t(j) \;=\; \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$
- An extension of a path from state i at time t-1 is computed by multiplying:

$v_{t-1}(i)$	the **previous Viterbi path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

Viterbi example

- 

Error Analysis: ESSENTIAL!!!

- Look at a confusion matrix

	IN	JJ	NN	NNP	RB	VBD	VBN
IN	—	.2			.7		
JJ	.2	—	3.3	2.1	1.7	.2	2.7
NN		8.7	—				.2
NNP	.2	3.3	4.1	—	.2		
RB	2.2	2.0	.5		—		
VBD		.3	.5			—	4.4
VBN		2.8				2.6	—

- See what errors are causing problems
  - Noun (NN) vs ProperNoun (NN) vs Adj (JJ)
  - Adverb (RB) vs Particle (RP) vs Prep (IN)
  - Preterite (VBD) vs Participle (VBN) vs Adjective (JJ)

Evaluation

- The result is compared with a manually coded “Gold Standard”
  - Typically accuracy reaches 96-97%
  - This may be compared with result for a baseline tagger (one that uses no context).
- Important: 100% is impossible even for human annotators.

Summary

- Part of speech tagging plays important role in TTS
- Most algorithms get 96-97% tag accuracy
- Not a lot of studies on whether remaining error tends to cause problems in TTS
  - For example POS taggers don't do well in reading headlines

Summary

I. Text Processing
   1) Text Normalization
      - Tokenization
      - End of sentence detection
         - Methodology: decision trees
   2) Homograph disambiguation
   3) Part-of-speech tagging
      - Methodology: Hidden Markov Models