L15: Large vocabulary continuous speech recognition

- Introduction
- Acoustic modeling
- Language modeling
- Decoding
- Evaluating LVCSR systems

This lecture is based on [Holmes, 2001, ch. 12; Young, 2008, in Benesty et al., (Eds)]

Introduction

LVCSR falls into two distinct categories

- Speech transcription
  - The goal is to find out exactly what the speaker said, in terms of an orthographic transcription (i.e., text)
  - Performance is measured in terms of word recognition errors
  - Applications include dictation and automatic generation of transcripts (i.e. from broadcast news)
- Speech understanding
  - The goal is to find out the meaning of the message; word recognition errors do not matter as long as they do not affect the inferred meaning
  - Applications include interactive dialogue systems, and audio summarization (i.e., from broadcast news)
- In this lecture we focus on speech transcription

Speech transcription

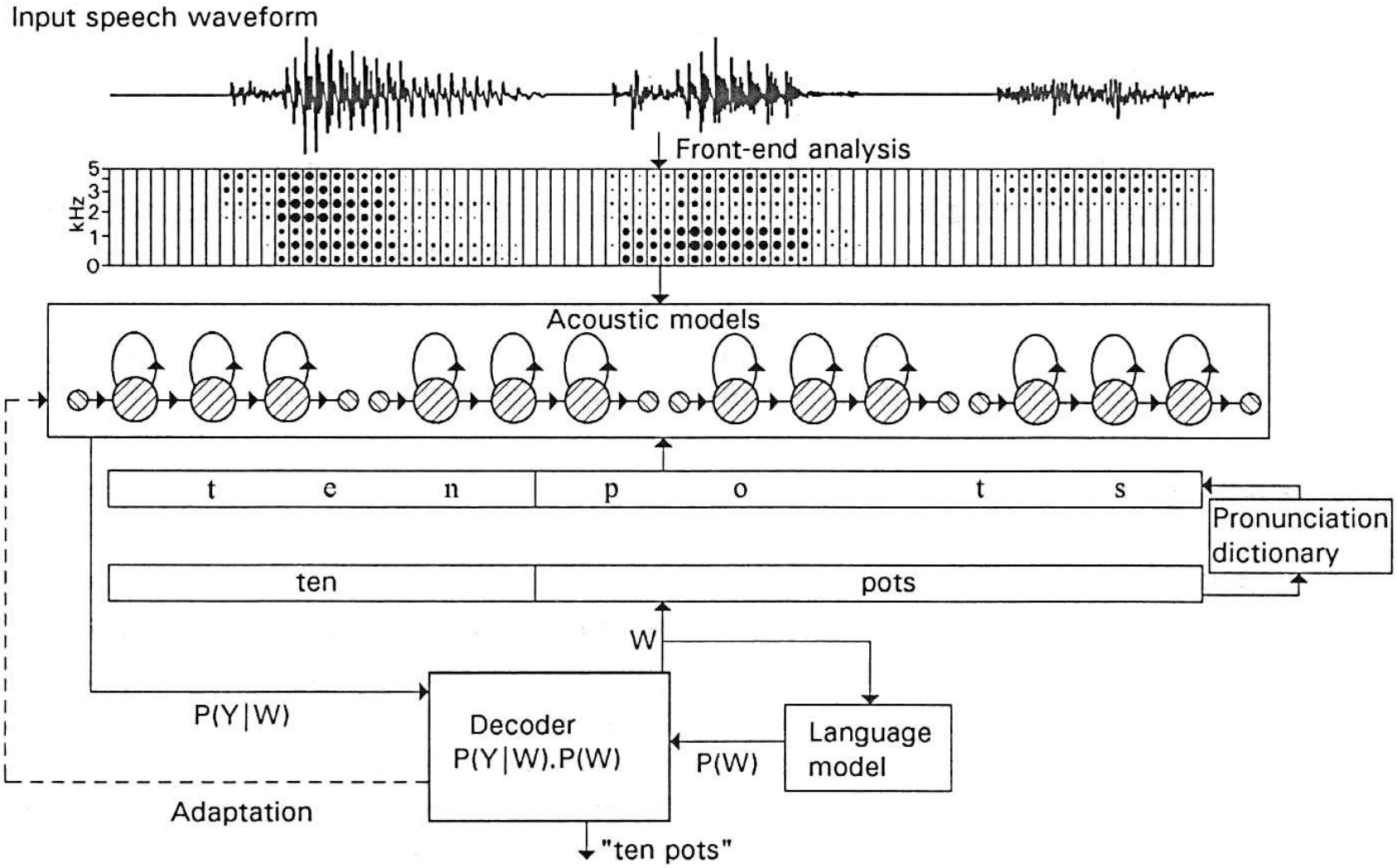
- Once the speech signal has been converted into a sequence of feature vectors, the recognition task consists of finding the most probable word sequence W given the observed data Y

$$\widehat{W} = \arg\max_{W} P(W|Y) = \arg\max_{W} \frac{P(Y|W)P(W)}{P(Y)} = \arg\max_{W} P(Y|W)P(W)$$

- The term $P(Y|W)$ is determined by an acoustic model, generally based on hidden Markov models learned from a database of utterances
- The term $P(W)$ is determined by a language model, generally based on n-gram statistical models built from text material chosen to be representative of the application

The example next page illustrates the overall procedure

- Language model postulates a word sequence, in this case ‘ten pots’
- Word sequence is decomposed into a phonetic sequence by means of a pronunciation dictionary
- Phoneme-level HMMs are concatenated to form a model of the word sequence
- The likelihood of the data given the word sequence $P(Y|W)$ is calculated, and multiplied by the probability of the word sequence $P(W)$
- In principle, this process is repeated for a number of word sequences and the best one is chosen as the recognizer output
- In practice, a decoder is used to make the latter step computationally effective


Input speech waveform
Front-end analysis
kHz
5
3
2
1
0
Acoustic models
t e n p o t s
Pronunciation dictionary
ten pots
W
P(Y|W)
Decoder
P(Y|W).P(W)
P(W)
Language model
Adaptation
"ten pots"

[Holmes, 2001]

Challenges posed by large vocabularies

- In continuous speech, words may not be distinguishable based on their acoustic information alone
  - First, due to coarticulation, word boundaries are not usually clear. In some instances, linguistically different sequences have very similar or identical acoustic information (e.g., 'grey day' vs. 'grade A')
  - Second, the pronunciation of many words, particularly function words (e.g., articles, pronouns, conjunctions…), can be reduced to where there is hardly any acoustic information
- Memory and computational requirements become very large, particularly in terms of decoding
- With increasing vocabularies, it becomes increasingly harder to find sufficient data to train the acoustic models and even the language models

Acoustic modeling

Context-dependent phone modeling

- Considering the amount of words in a typical language (500k to 1M words in English, depending on the source), it is impractical to train a separate HMM for each word in a LVCSR
  - Note also that even if it was possible, it would be highly impractical since many words can share subcomponents
- For these reasons, and as illustrated in the previous example, LVCSR systems are based on sub-word units, generally phoneme-sized
  - This unit size is more effective and allows new words to be added simply by extending the pronunciation dictionary
  - Approximately 44 phonemes are needed to represent all English words
- Due to co-articulation, however, the acoustic realization of any one phoneme can vary dramatically depending on its context
- For this reason, context-dependent HMMs are generally used

Triphones

- The most popular context-dependent unit is the triphone, whereby each phone has a distinct HMM for every pair of left and right contexts
  - Using triphones, the word 'ten' spoken in isolation would be modeled as $\{sil\ _{sil}t_e\ _te_n\ _en_{sil}\ sil\}$
  - In contrast, the phrase 'ten pots' would be modeled by the triphone sequence $\{sil\ _{sil}t_e\ _te_n\ _en_p\ _np_o\ _po_t\ _ot_s\ _ts_{sil}\ sil\}$
  - Notice how the two instances of phone [t] are represented by a different triphone because their contexts are different
- The above are known as a <u>cross-word triphones</u>
  - CWTs are beneficial because they model coarticulation effects across word boundaries, but complicate the decoding process since the sequence of HMMs for any one word will depend on the following word
- An alternative is to use <u>word-internal triphones</u>
  - WITs explicitly encode word boundaries, which facilitates decoding; in the example above , the triphones $\{_en_p\ _np_o\}$ would be replaced by $\{_en_-\ _-p_o\}$
  - However, their inability to model contextual effects across words is too much of a disadvantage, and current systems generally use CWTs

Training issues with context-dependent models

- With 44 phones there are 44^3=85,184 triphones, though many of these combinations do not occur due to phonotactic constraints
- Nonetheless, LVCSR systems will need around 60,000 triphones, which is a large enough number to pose challenges for model training
  - First, the models add up to a very large number of parameters
    - Assuming 39-dimensional vectors (12 MFCC + energy, Δ, Δ^2) and diagonal matrices, each state needs 790 parameters (30×10 means, 30×10 variances, 10 mixture weights)
    - Assuming 3-state models (typical in HTK) and 10 mixture components per state (needed to model speaker variability), a system with 60k triphones will require over 142M parameters!
  - In addition, many triphones will not occur in most training sets, so some method is required to generate models for these unseen triphones
- Several smoothing techniques can be used to address these issues, as we see next

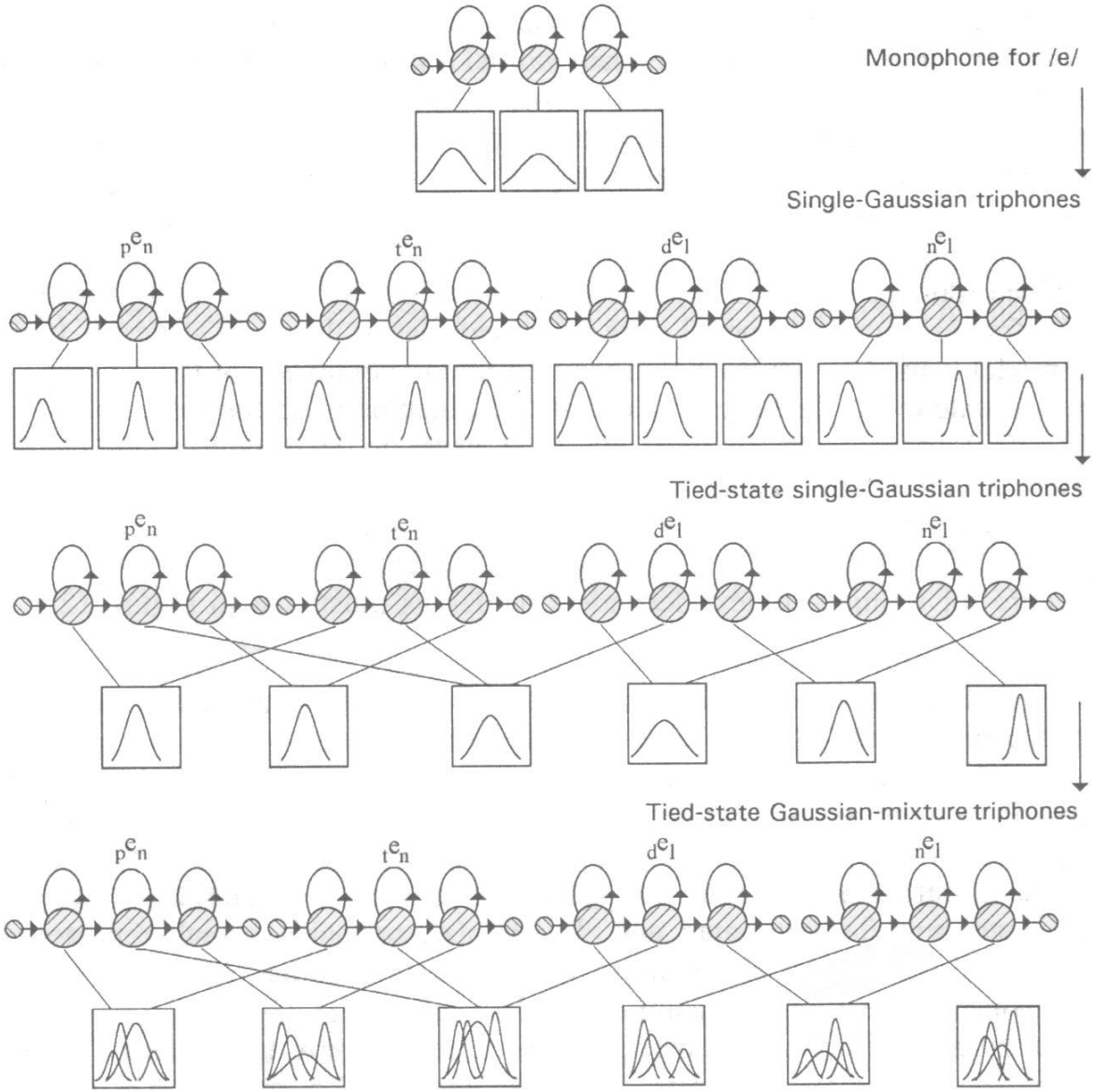
Smoothing techniques

- Backing off
  - When there is insufficient data to train a context-dependent model, one can “back-off” to a less-specific model for which data is available
    - As an example, one may replace a triphone by a relevant biphone, generally a *right*-biphone since coarticulation tends to be anticipatory
    - In there are insufficient examples to train a biphone, one may then use a context-independent phone model: a monophone
  - Backing-off ensures that every model is adequately trained, though at the expense that some context are not modeled very accurately
- Interpolation
  - One may also interpolate the parameters of a context-dependent model with those of a less-specific model to establish a compromise between context-dependency and model robustness

- **Parameter tying**
  - Alternatively, one may cluster all the triphones representing any one phone into groups with similar characteristics
    - This approach can retain a greater degree of specificity than the previous method and is most commonly used in LVCSR systems
  - The first attempts at parameter tying focused on clustering triphone models into generalized triphones
    - This approach assumed that the similarity between two models is the same for all the states in the models
    - To see how this is an erroneous assumption, consider triphones $\{{}_{t}e_{n\ t}e_{p\ k}e_{n}\}$: for triphones 1-2 the first state may be expected to be very similar, whereas for triphones 1-3 it is the last state that may expected to be similar
  - Thus, tying at the state level rather than at the model level offers much more flexibility in terms of making the best use of the training data
  - Next, we discuss two issues one encounters when using parameter tying
    - The general procedure to train tied-state mixture models
    - The choice of clustering method to decide on state groupings

Training procedure for tied-state models (typical)

- Monophone HMMs (1-Gaussian, diagonal Σ) are created and trained
- All training utterances are transcribed into triphones
  - For each triphone, an initial model is "cloned" from its monophone
  - Triphone model parameters are re-estimated and state occupancies are stored for later use
- Triphones representing each phone are clustered to create tied states
  - In the process, one needs to make sure sufficient data are available for each state (i.e., by ensuring state occupancies exceed a threshold count)
  - Parameters of the tied-state single-Gaussian models are re-estimated
- Multiple-component mixtures are trained with a mixture-splitting procedure
  - Starting from a single Gaussian, a 2-Gaussian is obtained by duplicating and perturbing the means in opposite directions (e.g., $\pm 0.2\sigma$); covariances are left unaltered and mixing coefficients are set to 0.5
  - Mean, covariance and mixing coefficient are re-estimated
  - Mixture-splitting is reapplied to the component with largest weight, and the process is repeated until the desired complexity is reached


Monophone for /e/
Single-Gaussian triphones
Tied-state single-Gaussian triphones
Tied-state Gaussian-mixture triphones

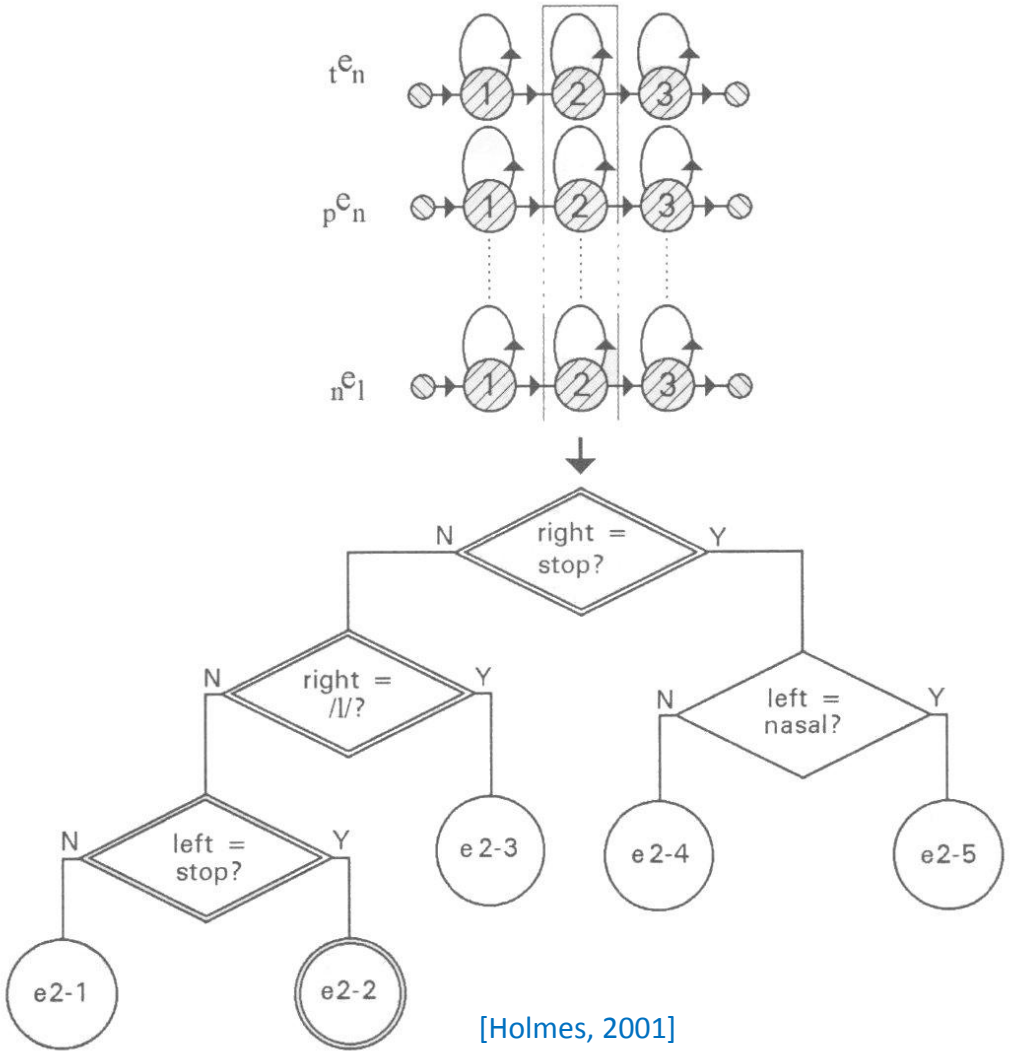
[Holmes, 2001]

- Introducing the multi-component Gaussians in the last stage has several advantages
  - Triphone mixture models are trained only after the model inventory has been setup to ensure adequate training data is available for each state
  - State-typing procedure is simpler because the state similarity measure consists of comparing pairs of single Gaussians (rather than pairs of mixtures)
  - By not introducing mixtures for monophone models one avoids using the mixture to capture contextual variation, a job that is reserved to the triphones (mixture components are needed to model speaker variability!)

Clustering procedures for tied-state models

- Bottom-up (agglomerative) clustering
  - Start with a separate model for each triphone
  - Merge similar states to form a new model state
  - Repeat until sufficient training data is available for each state
  - For triphones not included in the training set, back off to bi/mono-phones
- Top-down clustering (phonetic decision tree)
  - All triphones for a phoneme are initially grouped together
  - Hierarchical splitting procedure is used to progressively divide the group
    - Splitting is based on binary questions about the left or right phonetic context
      - Questions may relate to specific phones (i.e., is the phone to the right /n/?) or to broad phonetic classes (i.e. is the phone to the right a nasal?)
    - Questions are arranged as a phonetic decision tree
  - All states clustered at each leaf node are tied together
  - This approach to clustering ensures that a model will be specified for *any* triphone, regardless of whether it occurred in the training set
    - This method builds more accurate models than backing off

Decision tree used to cluster the center state of some /e/ triphones



[Holmes, 2001]

Constructing a phonetic decision tree

- Linguistic knowledge is used to choose context questions
  - Questions may include tests for a specific phone, phonetic classes (e.g., stop, vowel), more restrictive classes (e.g. voiced stop, front vowel) or more general classes (e.g., voiced consonant, continuant)
  - Typically, there are about 100 questions for each context (left vs. right)
- The tree building procedure works as follows
  - Place all states to be clustered at the root node
  - Find the best question for splitting S into two groups
    - Compute mean and variance assuming that all states in S are tied
    - Estimate the likelihood of the data given the pool of states $L(S)$
    - For each question, compute likelihoods for yes/no groups $L\left(S_{y/n}(q)\right)$
    - Choose question that maximizes $\Delta L_q = L\left(S_y(q)\right) + L(S_n(q)) - L(S)$
    - Split nodes according to the winning question, and repeat process
    - Process terminates when (1) splitting leads to a node with fewer examples than an established occupancy threshold, or (2) ΔL_q falls below a threshold, which avoids splitting a node when all its states are acoustically similar

Language modeling

N-grams

- The purpose of the language model is to take advantage of linguistic constraints to compute the probability of different word sequences
- Assuming a sequence of K words, $W = \{w_1, w_2, \dots, w_K\}$, the probability $P(W)$ can be expanded as

$$P(W) = P(w_1, w_2, \dots, w_K) = \prod_{k=1}^{K} P(w_k | w_1, w_2, \dots, w_{k-1})$$

- Since it is unfeasible to specify this probability for every possible word sequence, we generally make the simplifying assumption that any word w_k depends only on the previous $N-1$ words in the sequence

$$P(W) = \prod_{k=1}^{K} P(w_k | w_1, w_2, \dots, w_{k-1}) \approx \prod_{k=1}^{K} P(w_k | w_{k-N+1}, \dots, w_{k-1})$$

- This is known as an N-gram model
  - A unigram (N=1) represents the probability of each word
  - A bigram (N=2) models the probability of a word given its previous word
  - A trigram (N=3) takes into account the previous two words, and so on

- N-gram probabilities can be estimated using simple frequency counts from a text corpus
  - For a bigram model

$$\hat{P}(w_k|w_{k-1}) = \frac{C(w_k, w_{k-1})}{C(w_{k-1})}$$

  - For a trigram model

$$\hat{P}(w_k|w_{k-1}, w_{k-2}) = \frac{C(w_k, w_{k-1}, w_{k-2})}{C(w_{k-1}, w_{k-2})}$$

Perplexity of a language model

- Given a particular sequence of K words in some database, the value of $P(W)$ for that sequence is an indication of how well the LM can predict the sequence (the higher $P(W)$ the better)
- To account for word length, one then takes the K^{th} root, the inverse of which defines the perplexity $PP(W)$

$$PP(W) = [P(w_1, w_2 \dots w_K)]^{-1/K} = \left[\prod_{k=1}^{K} P(w_k | w_1, \dots, w_{k-1})\right]^{-1/K}$$

- Perplexity represents the average branching factor
  - i.e., the average number of words that need to be distinguished anywhere in the sequence assuming all words at any point were equiprobable
  - Perplexity is bounded by 1 (for a system where only one word sequence is allowed) and by ∞ (when any word in a sequence has zero probability)
- A good language model should have low perplexity when computed on a large corpus of unseen text material (i.e., outside the training set)
  - Thus, perplexity is a good measure for comparing different LMs
  - It also provides a good indicator of the difficulty of the recognition task that must be performed by the acoustic models

Data sparsity in language models

- A vocabulary with V words provides V^2 bigrams and V^3 trigrams
  - For a 20k-word dictionary, there are 400M bigrams and 8e6 trigrams
  - While typical text corpora may contain over 100M words, most of the possible bigrams and the vast majority of trigrams will not occur at all
- Thus, data sparsity is a much larger issue in LMs due to the larger number of units in the inventory (words vs. phones)
  - Hence, smoothing techniques are needed in order to obtain accurate, robust (non-zero) probability estimates for all possible N-grams
  - Smoothing refers to adjusting upwards zero or low-value probabilities, and adjusting downwards high probabilities
- Several smoothing techniques can be used, as described next

Smoothing in language models

- Discounting
  - For any set of events (bigrams or trigrams), the sum of probabilities for all possibilities must add up to one
    - When only a subset of all possible events occur in the training set (as is the case), then the sum must be less than one
  - This rationale is used in discounting to “free” probability mass from the observed events, which can be redistributed to the unseen events
    - One simple and effective method (among several) is absolute discounting, where some small fixed amount is subtracted from each frequency count
- Backing off
  - If a trigram is not observed (or has a very low frequency count) , then one backs off to the relevant bigram, or even to the monogram if the bigram is not available either
  - For words that do not occur in the corpus, one then backs off to a uniform distribution where all these words are assumed equiprobable

- Interpolation
  - Backing off involves choosing b/w a specific and a more general model
    - An alternative is to compute a weighted average of different probability estimates from contexts ranging from very specific to very general
  - As an example, a trigram probability could be estimated by linear interpolation b/w relevant trigrams, bigrams and unigrams

$$P(w_k|w_{k-2},w_{k-1}) = \lambda_3 \frac{C(w_{k-2},w_{k-1},w_k)}{C(w_{k-2},w_{k-1})} + \lambda_2 \frac{C(w_{k-1},w_k)}{C(w_{k-1})} + +\lambda_1 \frac{C(w_k)}{K}$$

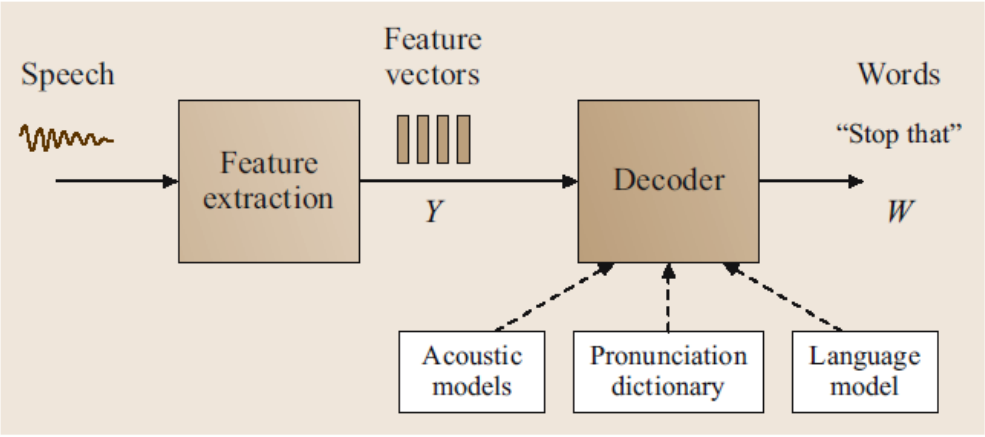
    - where K is the number of different words, and $\lambda_1 + \lambda_2 + \lambda_3 = 1$

  - When using interpolation, the training data is divided into two sets
    - The first (larger) set is used to derive the frequency counts
    - The second set is used to find the optimum value of the weights λ_i
  - One generally applies this process for different ways of splitting the data, and the individual estimates are combined

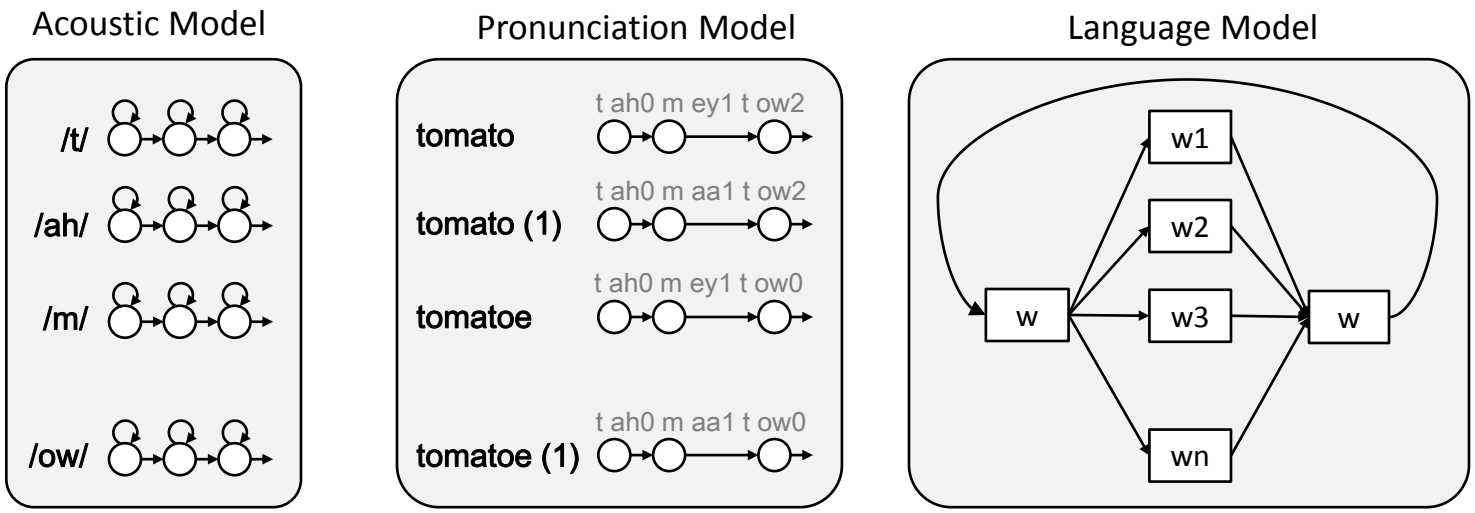
Decoding

Putting things together

- Once acoustic and language models are in place, the final step is to put all the elements together to find the most likely state sequence $\widehat{W}$ for a given sequence of feature vectors $Y = \{y_1, y_2 \dots y_T\}$
- In theory, this is "just" a search through a multi-level statistical model
  - At the lowest level, a network of states (an HMM) represents a triphone (the acoustic model)
  - At the next level, a network of triphones represents a word (the lexicon or pronunciation dictionary)
  - At the highest level, a network of words forms a sentence (the language model)


Speech
Feature vectors
Words
"Stop that"
Feature extraction
Y
Decoder
W
Acoustic models
Pronunciation dictionary
Language model

[Young, 2008]


Acoustic Model
/t/
/ah/
/m/
/ow/
Pronunciation Model
tomato
t ah0 m ey1 t ow2
tomato (1)
t ah0 m aa1 t ow2
tomatoe
t ah0 m ey1 t ow0
tomatoe (1)
t ah0 m aa1 t ow0
Language Model
w
w1
w2
w3
wn
w

- An efficient way to solve this problem is to use dynamic programming
  - Let $\phi_j(t) = \max_X\{p(y_1, \dots y_t, x(t) = j|\lambda)\}$ be the maximum probability of observing the partial sequence $\{y_1 \dots y_t\}$ and then being in state j at time t given model λ
  - As we saw in a previous lecture, this probability can be efficiently computed using the Viterbi algorithm

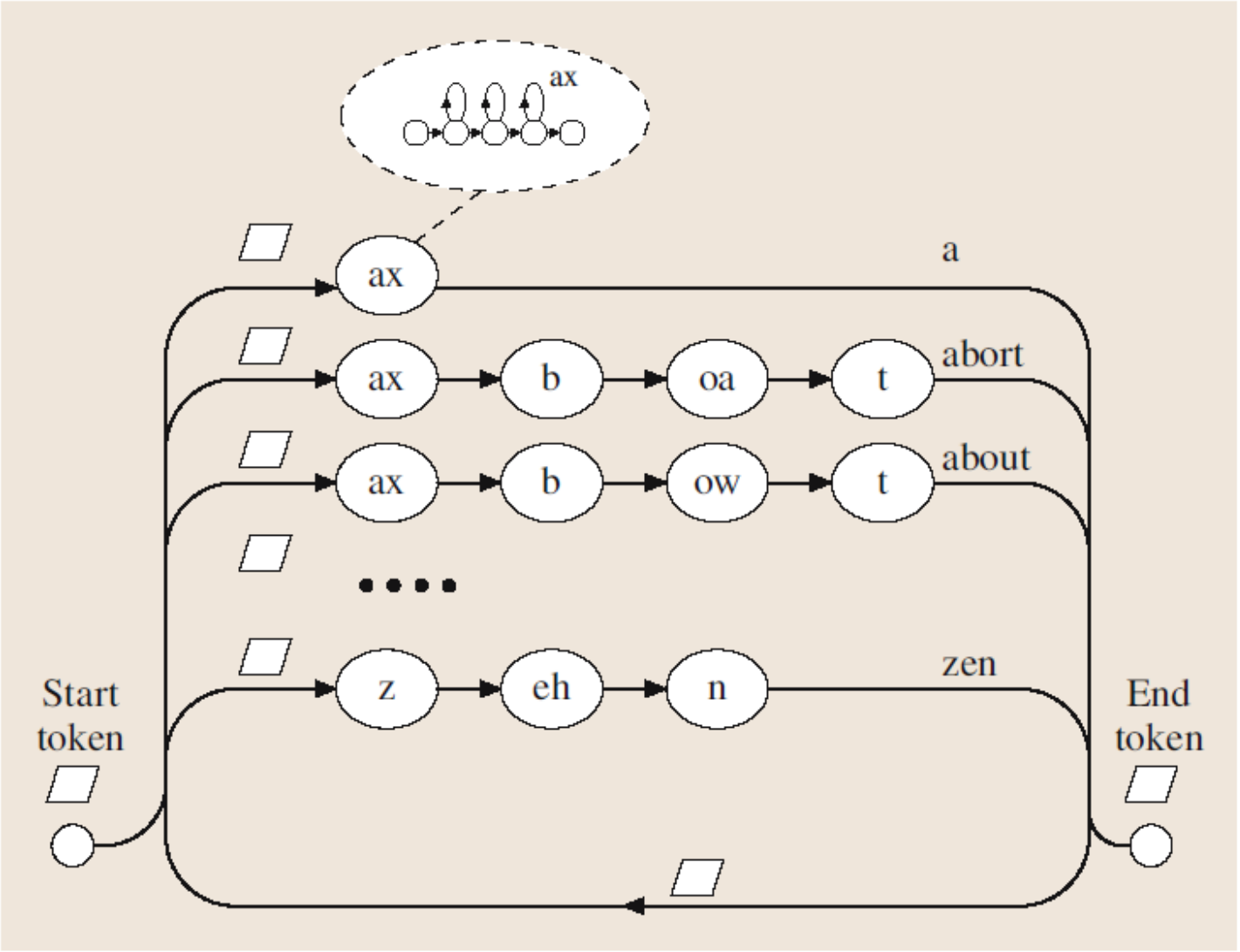
$$\phi_j(t) = \max_i\big(\phi_i(t-1)a_{ij}\big)b_j(y_t)$$

    - Initializing $\phi_j(t) = 1$ for the initial state, and zero elsewhere, the probability of the most likely state sequence is then $\max_j\{\phi_j(T)\}$
    - By recording every maximization decision, a traceback will then yield the required best matching state/word sequence

- As you may imagine, though, direct implementation of the Viterbi algorithm for decoding becomes unmanageable for LVCSR
- Fortunately, much of this complexity can be abstracted away by changing viewpoints: token passing

Token passing

- The HMM topology can be shown by building a recognition network
  - For task-oriented applications, it represents all allowable utterances
  - For LVCSR, it will consist of all vocabulary words in parallel in a loop
- At any time t in the search, a single hypothesis consists of a path through the network representing an alignment of states with feature vectors and having a log likelihood $\log \phi_j(t)$
- We now define a token as a pair of values $\langle \log P , link \rangle$, where
  - $\log P$ is the log likelihood (or score)
  - $link$ is a pointer to a record of history information
- In this way, each network node corresponding to a HMM state can store a single token and recognition proceeds by propagating these tokens around the network

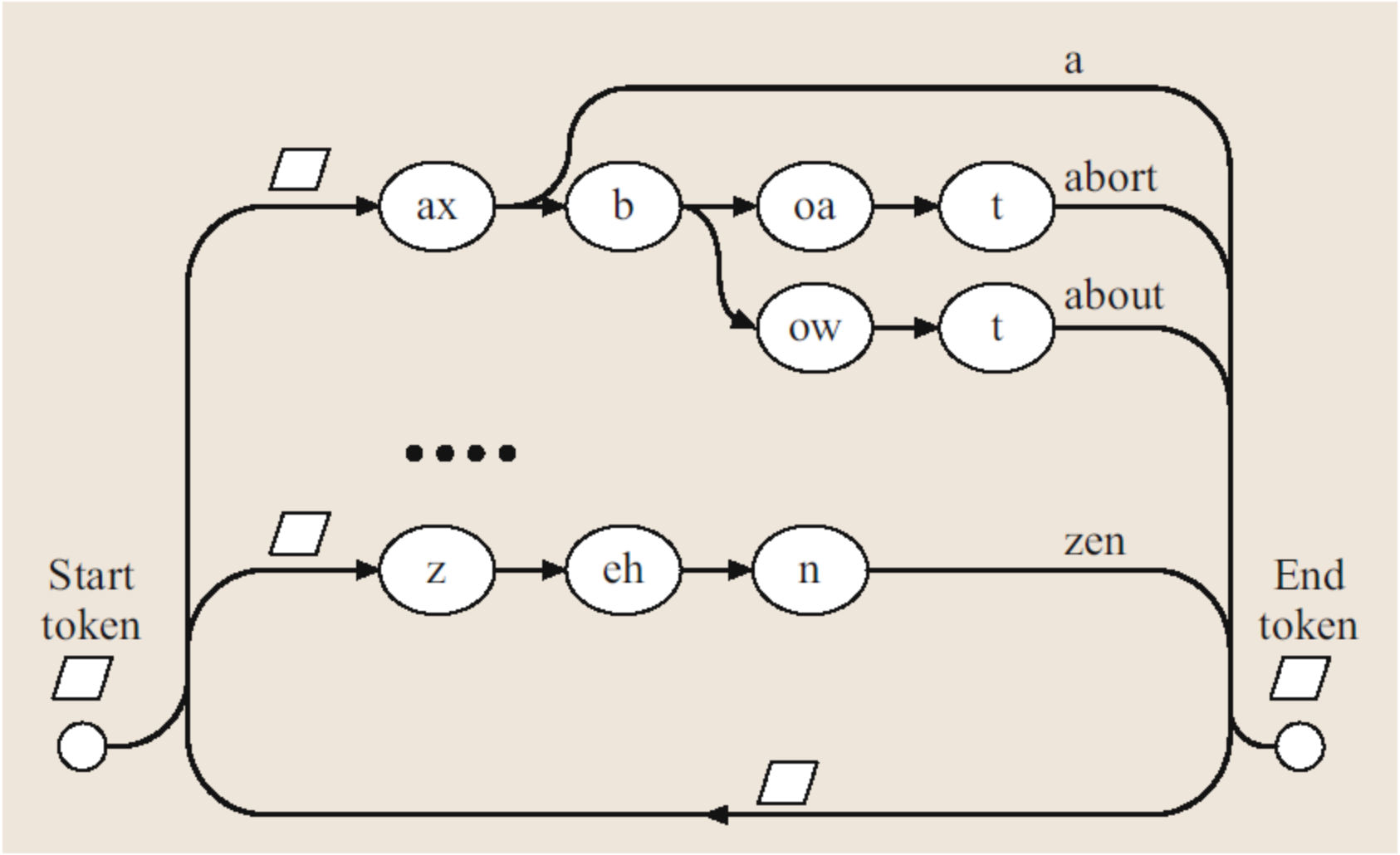

ax
ax
a
ax
b
oa
t
abort
ax
b
ow
t
about
z
eh
n
zen
Start token
End token

[Young, 2008]

- Viterbi can now be recast for LVCSR as a token-passing algorithm
  - When a token is passed between two internal states, its score is updated by the corresponding transition cost a_{ij} and observation cost $b_j(y_t)$
  - Each node then compares all of its tokens and discards all but the best
  - When a token transitions from the exit of a word to the start of the next word, its score is updated by the language model probability
  - At the same time, the transition is recorded in a record R containing a copy of the tokens, the current time and the identity of the previous word
  - The link field is then updated to point to the record R
  - As each token proceeds through the network, it accumulates a chain of these records
  - The best token at time T in a valid network exit point can then be examined and traced back to recover the most likely state sequence and the boundary times

Optimizing the token-passing algorithm

- Token passing leads to an exact implementation of Viterbi
- To make it practical for LVCSR, however, several improvements are needed, the most common being
  - Beam search
    - For efficiency, propagate only those tokens that have some likelihood of being on the best path
    - This can be achieved by discarding all tokes whose probabilities fall more than a constant below that of the most likely token
  - Tree-structured networks
    - As a result of beam search, 90% of the computation is spent on the first two phones of every word, after which most of the token are pruned
      - To exploit this, structure the recognition network such that word-initial phones are shared (see next slide)
    - Note that this prevents the LM probability to be added during word-external token propagation since the next word is not known
      - To address this issue, an incremental approach is used where the LM probability is taken to be the maximum of all possible following words; as tokens move forward, the choices become narrower and the LM probability can be updated



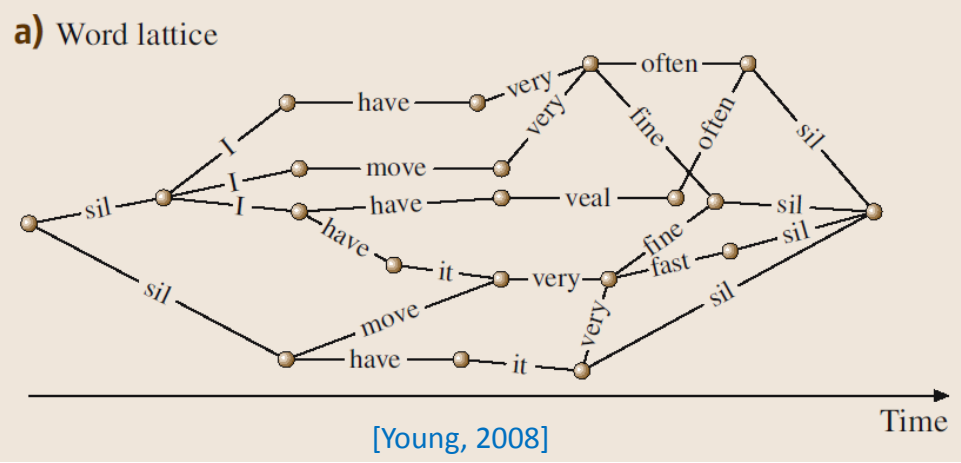
[Young, 2008]

N-grams and token-passing

- The DP principle assumes that the optimal path at any point can be extended by considering only the state information at that node
- This is an issue with N-gram models, because one then needs to keep track of all possible $N-1$ histories, which is intractable for LVCSR
- Thus, the algorithm just described only works for bigram models
- A solution for higher-order LMs is to store multiple tokens at each state, which allows multiple histories to "stay alive" in parallel during the search

Multi-pass Viterbi decoding

- The token-passing algorithm performs decoding in a single pass
- For off-line applications, significant improvements can be achieved by performing multiple passes through the data
  - The first pass could employ word-internal triphones and a bigram
  - The second pass could then use cross-word triphones and trigrams
- The output of the first recognition pass is generally expressed as
  - A rank-ordered N-best list of possible word sequences, or
  - A word graph or lattice describing all the possibilities as a network



[Young, 2008]

Stack decoding

- Viterbi can be described as a breadth-first search, because all the possibilities are considered in parallel
- An alternative is to adopt a depth-first search, whereby one pursues the most promising hypothesis until the end of the utterance
- This is know as stack decoding
  - the idea is to keep an ordered stack of possible hypotheses, take the best hypothesis from the stack, choose the most likely next word and add it to the stack, and re-order the stack if necessary
- Because the score is a product of probabilities, it will decrease with time, which biases the comparisons towards shorter sequences
  - To address this issue one normalizes each path by its number of frames
- Stack decoders, however, are expensive in terms of memory and processing requirements

Weighted finite state transducers (WFST)

- As we have seen, the decoder integrates a number of sources of knowledge (acoustic models, lexicon, language models)
- These knowledge sources, however, are generally hardwired into the decoder architecture, which makes modifications non-trivial
- For these reasons, in recent years considerable effort has been invested in developing more flexible architectures based on WFSTs
  - A FST is a finite automaton whose state transitions are labeled with both input and output symbols
    - Therefore, a path through the transducer encodes a mapping from an input symbol sequence to an output symbol sequence
  - A WFST is a FST with additional weights on transitions
- WFSTs allow us to integrate all of the required knowledge (acoustic models, pronunciation, language models) into a single, very large, but highly optimized network
  - For more details see [M Mohri, F Pereira and M Riley (2008), "Speech Recognition with Weighted Finite-State Transducers," in Springer Handbook of Speech Processing , ch. 28]

Evaluating LVCSR

Recognition errors

- When recognizing connected speech there are three types of errors
  - Substitution errors (the wrong word is recognized)
  - Deletions (a word is omitted)
  - Insertions (a n extra word is recognized)
- These three errors are generally reported as word error rates (WER)

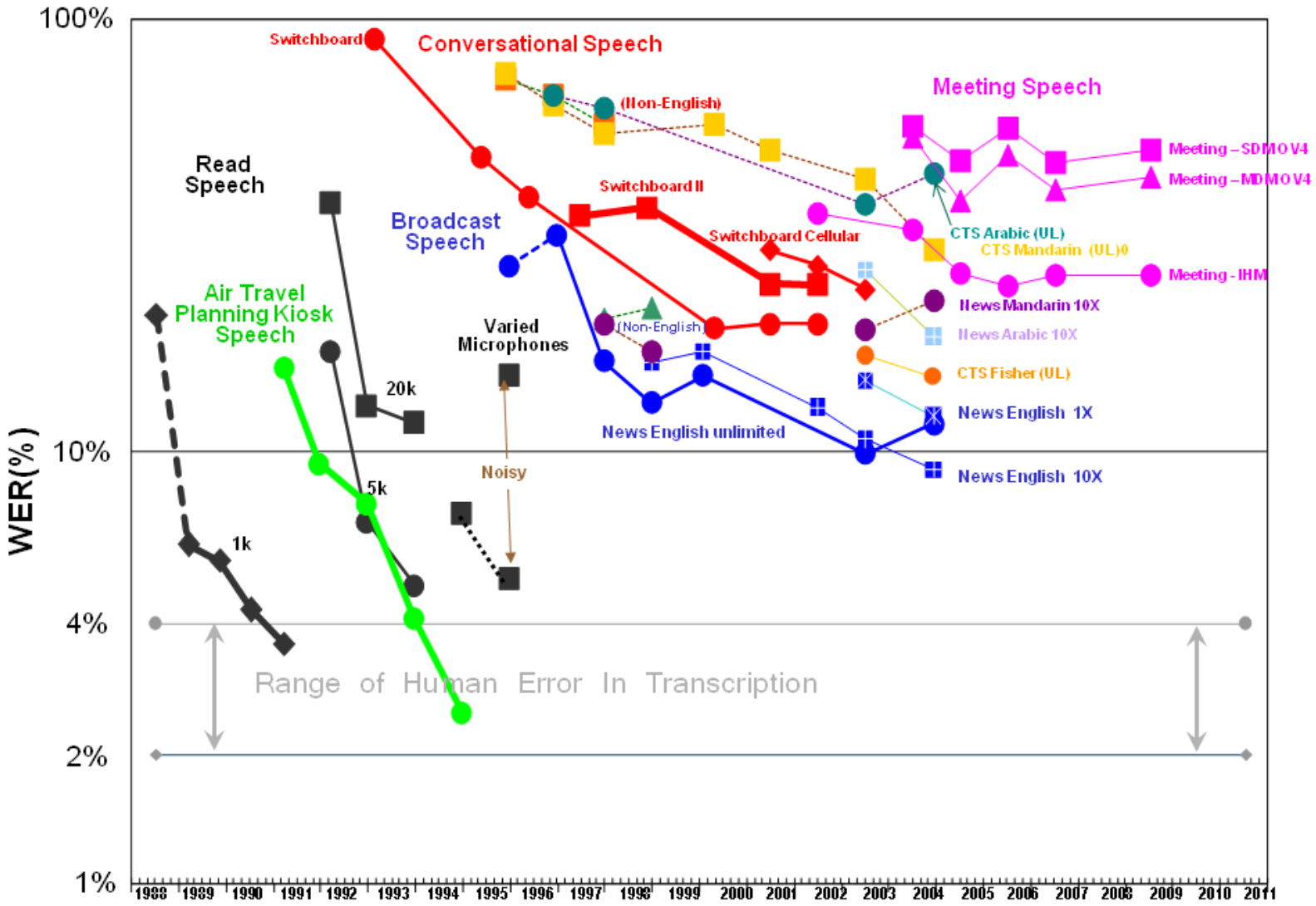
$$WER = \frac{C(subs) + C(del) + C(ins)}{N}$$

  - where N is the number of words in the text speech and $C(x)$ is the count of errors of type x

Controlling word insertion errors

- The final word sequence produced by the decoder will depend on the relative contributions from the acoustic and language models
- In general, the acoustic model has a disproportionately large influence relative to that of the LM
- This generally results in a large number of errors due to insertion of many short function words
  - Since they are short and have large variability, a sequence of their models mat provide the best acoustic match to short speech segments, even though the word sequence has very low probability according to the LM
- There are two practical solutions to this problem
  - Impose a word insertion penalty such that the probability of transitions between words is penalized by a multiplicative term less than one
  - Increase the influence of the language model by means of a multiplicative term greater than one

NIST STT Benchmark Test History – May. '09



http://itl.nist.gov/iad/mig/publications/ASRhistory/index.html