Chapter 10
Speech Recognition: Advanced Topics

True, their voice-print machine was unfortunately a crude one. It could discriminate among only a few frequencies, and it indicated amplitude by indecipherable blots. But it had never been intended for such vitally important work.

Aleksandr I. Solzhenitsyn, *The First Circle*, p. 505

The *keju* civil service examinations of Imperial China lasted almost 1300 years, from the year 606 until it was abolished in 1905. In its peak, millions of would-be officials from all over China competed for high-ranking government positions by participating in a uniform examination. For the final 'metropolitan' part of this exam in the capital city, the candidates would be locked into an examination compound for a grueling nine days and nights answering questions about history, poetry, the Confucian classics, and policy.

Naturally all these millions of candidates didn't all show up in the capital. Instead, the exam had progressive levels; candidates who passed a one-day local exam in their local prefecture could then sit for the biannual provincial exam, and only upon passing that exam in the provincial capital was a candidate eligible for the metropolitan and palace examinations.

This algorithm for selecting capable officials is an instance of multi-stage search. The final 9-day process requires far too many resources (in both space and time) to examine every candidate. Instead, the algorithm uses an easier, less intensive 1-day process to come up with a preliminary list of potential candidates, and applies the final test only to this list.

The *keju* algorithm can also be applied to speech recognition. We'd like to be able to apply very expensive algorithms in the speech recognition process, such as 4-gram, 5-gram, or even parser-based language models, or context-dependent phone models that can see two or three phones into the future or past. But there are a huge number of potential transcriptions sentences for any given waveform, and it's too expensive (in time, space, or both) to apply these powerful algorithms to every single candidate. Instead, we'll introduce **multipass decoding** algorithms in which efficient but dumber decoding algorithms produce shortlists of potential candidates to be rescored by slow but smarter algorithms. We'll also introduce the **context-dependent acoustic model**, which is one of these smarter knowledge sources that turns out to be essential in large-vocabulary speech recognition. We'll also briefly introduce the important topics of discriminative training and the modeling of variation.

10.1 Multipass Decoding: *N*-best lists and lattices

The previous chapter applied the Viterbi algorithm for HMM decoding. There are two main limitations of the Viterbi decoder, however. First, the Viterbi decoder does not actually compute the sequence of words which is most probable given the input acoustics. Instead, it computes an approximation to this: the sequence of *states* (i.e., *phones* or *subphones*) which is most probable given the input. More formally, recall that the true likelihood of an observation sequence O is computed by the forward algorithm by summing over all possible paths:

$$P(O|W) = \sum_{S \in S_1^T} P(O,S|W) \tag{10.1}$$

The Viterbi algorithm approximates this sum by using the probability of the best path:

$$P(O|W) \approx \max_{S \in S_1^T} P(O,S|W) \tag{10.2}$$

Viterbi approximation

It turns out that this **Viterbi approximation** is not too bad, since the most probable sequence of phones usually turns out to correspond to the most probable sequence of words. But not always. Consider a speech recognition system whose lexicon has multiple pronunciations for each word. Suppose the correct word sequence includes a word with very many pronunciations. Since the probabilities leaving the start arc of each word must sum to 1.0, each of these pronunciation-paths through this multiple-pronunciation HMM word model will have a smaller probability than the path through a word with only a single pronunciation path. Thus because the Viterbi decoder can only follow one of these pronunciation paths, it may ignore this many-pronunciation word in favor of an incorrect word with only one pronunciation path. In essence, the Viterbi approximation penalizes words with many pronunciations.

A second problem with the Viterbi decoder is that it is impossible or expensive for it to take advantage of many useful knowledge sources. For example the Viterbi algorithm as we have defined it cannot take complete advantage of any language model more complex than a bigram grammar. This is because of the fact mentioned earlier that a trigram grammar, for example, violates the **dynamic programming invariant**. Recall that this invariant is the simplifying (but incorrect) assumption that if the ultimate best path for the entire observation sequence happens to go through a state q_i, that this best path must include the best path up to and including state q_i. Since a trigram grammar allows the probability of a word to be based on the two previous words, it is possible that the best trigram-probability path for the sentence may go through a word but not include the best path to that word. Such a situation could occur if a particular word w_x has a high trigram probability given w_y, w_z, but that conversely the best path to w_y didn't include w_z (i.e., $P(w_y|w_q, w_z)$ was low for all q). Advanced probabilistic LMs like SCFGs also violate the same dynamic programming assumptions.

There are two solutions to these problems with Viterbi decoding. The most common is to modify the Viterbi decoder to return multiple potential utterances, instead of just the single best, and then use other high-level language model or pronunciation-

modeling algorithms to re-rank these multiple outputs (Schwartz and Austin, 1991; Soong and Huang, 1990; Murveit et al., 1993).

stack decoder
*A**

The second solution is to employ a completely different decoding algorithm, such as the **stack decoder**, or **A*** decoder (Jelinek, 1969; Jelinek et al., 1975). We begin in this section with multiple-pass decoding, and then return to stack decoding.

In **multiple-pass decoding** we break up the decoding process into two stages. In the first stage we use fast, efficient knowledge sources or algorithms to perform a non-optimal search. So for example we might use an unsophisticated but time-and-space efficient language model like a bigram, or use simplified acoustic models. In the second decoding pass we can apply more sophisticated but slower decoding algorithms on a reduced search space. The interface between these passes is an *N*-**best list** or **word lattice**.

N-best

The simplest algorithm for multipass decoding is to modify the Viterbi algorithm to return the **N-best** sentences (word sequences) for a given speech input. Suppose for example a bigram grammar is used with such an *N*-best-Viterbi algorithm to return the 1000 most highly-probable sentences, each with their AM likelihood and LM prior score. This 1000-best list can now be passed to a more sophisticated language model like a trigram grammar. This new LM is used to replace the bigram LM score of each hypothesized sentence with a new trigram LM probability. These priors can be combined with the acoustic likelihood of each sentence to generate a new posterior probability for each sentence. Sentences are thus **rescored** and re-ranked using this more sophisticated probability. Fig. 10.1 shows an intuition for this algorithm.

Rescoring

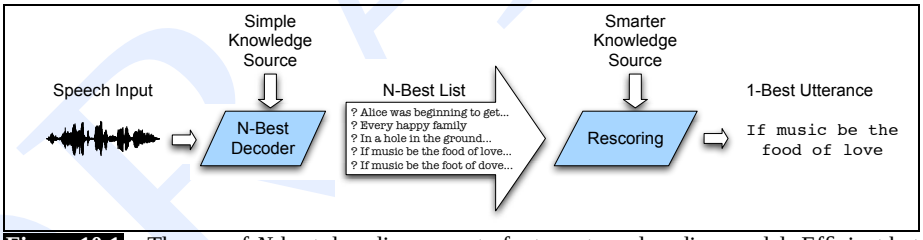

Figure 10.1 The use of *N*-best decoding as part of a two-stage decoding model. Efficient but unsophisticated knowledge sources are used to return the *N*-best utterances. This significantly reduces the search space for second pass models, which can thus be sophisticated but slow.

There are a number of algorithms for augmenting the Viterbi algorithm to generate *N*-best hypotheses. It turns out that there is no polynomial-time admissible algorithm for finding the *N* most likely hypotheses (Young, 1984). There are however, a number of approximate (non-admissible) algorithms; we will introduce just one of them, the "Exact *N*-best" algorithm of Schwartz and Chow (1990). In Exact *N*-best, instead of each state maintaining a single path/backtrace, we maintain up to *N* different paths for each state. But we'd like to insure that these paths correspond to different word paths; we don't want to waste our *N* paths on different state sequences that map to the same words. To do this, we keep for each path the **word history**, the entire sequence of words up to the current word/state. If two paths with the same word history come to a state at the same time, we merge the paths and sum the path probabilities. To keep the *N* best word sequences, the resulting algorithm requires $O(N)$ times the normal

Viterbi time. We'll see this merging of paths again when we introducing decoding for statistical machine translation, where it is called **hypothesis recombination**.

Rank	Path	AM logprob	LM logprob
1.	it's an area that's naturally sort of mysterious	-7193.53	-20.25
2.	that's an area that's naturally sort of mysterious	-7192.28	-21.11
3.	it's an area that's not really sort of mysterious	-7221.68	-18.91
4.	that scenario that's naturally sort of mysterious	-7189.19	-22.08
5.	there's an area that's naturally sort of mysterious	-7198.35	-21.34
6.	that's an area that's not really sort of mysterious	-7220.44	-19.77
7.	the scenario that's naturally sort of mysterious	-7205.42	-21.50
8.	so it's an area that's naturally sort of mysterious	-7195.92	-21.71
9.	that scenario that's not really sort of mysterious	-7217.34	-20.70
10.	there's an area that's not really sort of mysterious	-7226.51	-20.01

Figure 10.2 An example 10-Best list from the Broadcast News corpus, produced by the CU-HTK BN system (thanks to Phil Woodland). Logprobs use $\log_{10}$; the language model scale factor (LMSF) is 15.

The result of any of these algorithms is an N-best list like the one shown in Fig. 10.1. In Fig. 10.1 the correct hypothesis happens to be the first one, but of course the reason to use N-best lists is that isn't always the case. Each sentence in an N-best list is also annotated with an acoustic model probability and a language model probability. This allows a second-stage knowledge source to replace one of those two probabilities with an improved estimate.

One problem with an N-best list is that when N is large, listing all the sentences is extremely inefficient. Another problem is that N-best lists don't give quite as much information as we might want for a second-pass decoder. For example, we might want distinct acoustic model information for each word hypothesis so that we can reapply a new acoustic model for the word. Or we might want to have available different start and end times of each word so that we can apply a new duration model.

word lattice

For this reason, the output of a first-pass decoder is usually a more sophisticated representation called a **word lattice** (Murveit et al., 1993; Aubert and Ney, 1995). A word lattice is a directed graph that efficiently represents much more information about possible word sequences.[1] In some systems, nodes in the graph are words and arcs are transitions between words. In others, arcs represent word hypotheses and nodes are points in time. Let's use this latter model, and so each arc represents lots of information about the word hypothesis, including the start and end time, the acoustic model and language model probabilities, the sequence of phones (the pronunciation of the word), or even the phone durations. Fig. 10.3 shows a sample lattice corresponding to the N-best list in Fig. 10.1. Note that the lattice contains many distinct links (records) for the same word, each with a slightly different starting or ending time. Such lattices are not produced from N-best lists; instead, a lattice is produced during first-pass decoding by including some of the word hypotheses which were active (in the beam) at each time-

[1] Actually an ASR lattice is not the kind of lattice that may be familiar to you from mathematics, since it is not required to have the properties of a true lattice (i.e., be a partially ordered set with particular properties, such as a unique join for each pair of elements). Really it's just a graph, but it is conventional to call it a lattice.

step. Since the acoustic and language models are context-dependent, distinct links need to be created for each relevant context, resulting in a large number of links with the same word but different times and contexts. *N*-best lists like Fig. 10.1 can also be produced by first building a lattice like Fig. 10.3 and then tracing through the paths to produce *N* word strings.

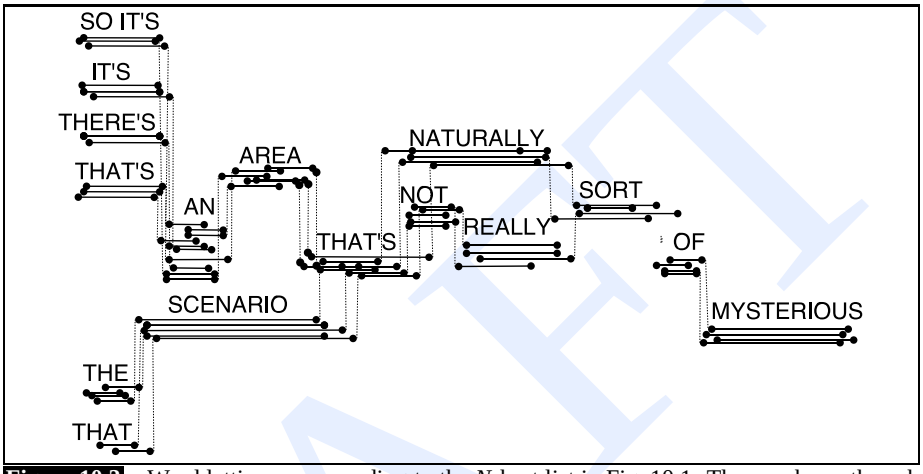

Figure 10.3 Word lattice corresponding to the *N*-best list in Fig. 10.1. The arcs beneath each word show the different start and end times for each word hypothesis in the lattice; for some of these we've shown schematically how each word hypothesis must start at the end of a previous hypothesis. Not shown in this figure are the acoustic and language model probabilities that decorate each arc.

The fact that each word hypothesis in a lattice is augmented separately with its acoustic model likelihood and language model probability allows us to rescore any path through the lattice, using either a more sophisticated language model or a more sophisticated acoustic model. As with *N*-best lists, the goal of this rescoring is to replace the **1-best utterance** with a different utterance that perhaps had a lower score on the first decoding pass. For this second-pass knowledge source to get perfect word error rate, the actual correct sentence would have to be in the lattice or *N*-best list. If the correct sentence isn't there, the rescoring knowledge source can't find it. Thus it is important when working with a lattice or *N*-best list to consider the baseline **lattice error rate** (Woodland et al., 1995; Ortmanns et al., 1997): the lower bound word error rate from the lattice. The lattice error rate is the word error rate we get if we chose the lattice path (the sentence) that has the lowest word error rate. Because it relies on perfect knowledge of which path to pick, we call this an **oracle** error rate, since we need some oracle to tell us which sentence/path to pick.

Lattice error rate

Oracle error rate

Lattice density

Another important lattice concept is the **lattice density**, which is the number of edges in a lattice divided by the number of words in the reference transcript. As we saw schematically in Fig. 10.3, real lattices are often extremely dense, with many copies of individual word hypotheses at slightly different start and end times. Because of this density, lattices are often pruned.

Besides pruning, lattices are often simplified into a different, more schematic kind

Word graph

of lattice that is sometimes called a **word graph** or **finite-state machine**, although often it's still just referred to as a word lattice. In these word graphs, the timing information is removed and multiple overlapping copies of the same word are merged. The timing of the words is left implicit in the structure of the graph. In addition, the acoustic model likelihood information is removed, leaving only the language model probabilities. The resulting graph is a weighted FSA, which is a natural extension of an N-gram language model; the word graph corresponding to Fig. 10.3 is shown in Fig. 10.4. This word graph can in fact be used as the language model for another decoding pass. Since such a wordgraph language model vastly restricts the search space, it can make it possible to use a complicated acoustic model which is too slow to use in first-pass decoding.

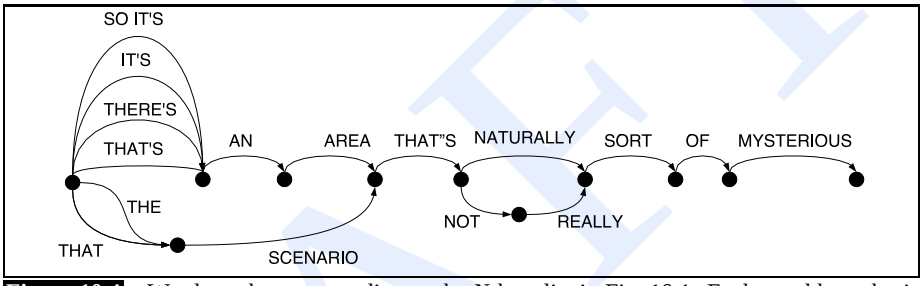

Figure 10.4 Word graph corresponding to the N-best list in Fig. 10.1. Each word hypothesis in the lattice also has language model probabilities (not shown in this figure).

A final type of lattice is used when we need to represent the posterior probability of individual words in a lattice. It turns out that in speech recognition, we almost never see the true posterior probability of anything, despite the fact that the goal of speech recognition is to compute the sentence with the maximum a posteriori probability. This is because in the fundamental equation of speech recognition we ignore the denominator in our maximization:

$$\hat{W} = \operatorname*{argmax}_{W \in \mathscr{L}} \frac{P(O|W)P(W)}{P(O)} = \operatorname*{argmax}_{W \in \mathscr{L}} P(O|W)\,P(W) \tag{10.3}$$

The product of the likelihood and the prior is **not** the posterior probability of the utterance. It is not even a probability, since it doesn't necessarily lie between 0 and 1. It's just a score. Why does it matter that we don't have a true probability? The reason is that without having true probability, we can choose the best hypothesis, but we can't know how good it is. Perhaps the best hypothesis is still really bad, and we need to ask the user to repeat themselves. If we had the posterior probability of a word it could be used as a **confidence** metric, since the posterior is an absolute rather than relative measure. A confidence metric is a metric that the speech recognizer can give to a higher-level process (like dialogue) to indicate how confident the recognizer is that the word string that it returns is a good one. We'll return to the use of confidence in Ch. 24.

In order to compute the posterior probability of a word, we'll need to normalize over all the different word hypotheses available at a particular point in the utterances.

Confusion network
Mesh
Sausage
Pinched lattice

At each point we'll need to know which words are competing or confusable. The lattices that show these sequences of word confusions are called **confusion networks**, **meshes**, **sausages**, or **pinched lattices**. A confusion network consists of a sequence of word positions. At each position is a set of mutually exclusive word hypotheses. The network represents the set of sentences that can be created by choosing one word from each position.

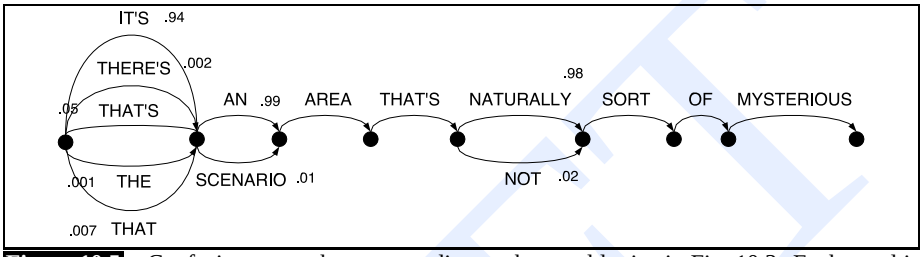

Figure 10.5 Confusion network corresponding to the word lattice in Fig. 10.3. Each word is associated with a posterior probability. Note that some of the words from the lattice have been pruned away. (Probabilities computed by the SRI-LM toolkit).

Note that unlike lattices or word graphs, the process of constructing a confusion network actually adds paths that were not in the original lattice. Confusion networks have other uses besides computing confidence. They were originally proposed for use in minimizing word error rate, by focusing on maximizing improving the word posterior probability rather than the sentence likelihood. Recently confusion networks have been used to train discriminative classifiers that distinguish between words.

Roughly speaking, confusion networks are built by taking the different hypothesis paths in the lattice and aligning them with each other. The posterior probability for each word is computing by first summing over all paths passing through a word, and then normalizing by the sum of the probabilities of all competing words. For further details see Mangu et al. (2000), Evermann and Woodland (2000), Kumar and Byrne (2002), Doumpiotis et al. (2003b).

Standard publicly available language modeling toolkits like SRI-LM (Stolcke, 2002) (`http://www.speech.sri.com/projects/srilm/`) and the HTK language modeling toolkit (Young et al., 2005) (`http://htk.eng.cam.ac.uk/`) can be used to generate and manipulate lattices, *N*-best lists, and confusion networks.

forward-backward

There are many other kinds of multiple-stage search, such as the **forward-backward** search algorithm (not to be confused with the **forward-backward** algorithm for HMM parameter setting) (Austin et al., 1991) which performs a simple forward search followed by a detailed backward (i.e., time-reversed) search.

10.2 A* ('Stack') Decoding

Recall that the Viterbi algorithm approximated the forward computation, computing the likelihood of the single best (MAX) path through the HMM, while the forward algorithm computes the likelihood of the total (SUM) of all the paths through the HMM.

The A* decoding algorithm allows us to use the complete forward probability, avoiding the Viterbi approximation. A* decoding also allows us to use any arbitrary language model. Thus A* is a one-pass alternative to multi-pass decoding.

The A* decoding algorithm is a best-first search of the tree that implicitly defines the sequence of allowable words in a language. Consider the tree in Fig. 10.6, rooted in the START node on the left. Each leaf of this tree defines one sentence of the language; the one formed by concatenating all the words along the path from START to the leaf. We don't represent this tree explicitly, but the stack decoding algorithm uses the tree implicitly as a way to structure the decoding search.

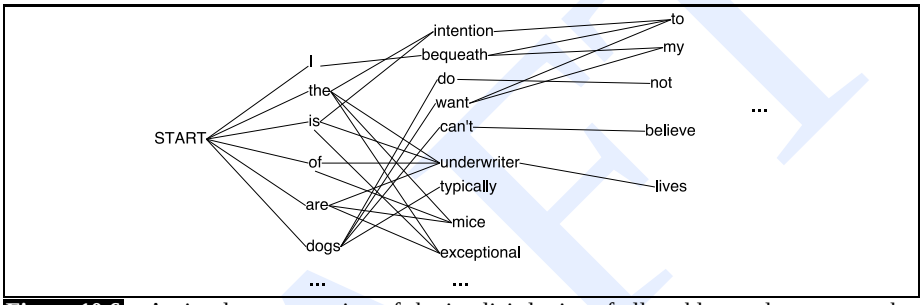

Figure 10.6 A visual representation of the implicit lattice of allowable word sequences that defines a language. The set of sentences of a language is far too large to represent explicitly, but the lattice gives a metaphor for exploring prefixes.

The algorithm performs a search from the root of the tree toward the leaves, looking for the highest probability path, and hence the highest probability sentence. As we proceed from root toward the leaves, each branch leaving a given word node represents a word which may follow the current word. Each of these branches has a probability, which expresses the conditional probability of this next word given the part of the sentence we've seen so far. In addition, we will use the forward algorithm to assign each word a likelihood of producing some part of the observed acoustic data. The A* decoder must thus find the path (word sequence) from the root to a leaf which has the highest probability, where a path probability is defined as the product of its language model probability (prior) and its acoustic match to the data (likelihood). It does this by keeping a **priority queue** of partial paths (i.e., prefixes of sentences, each annotated with a score). In a priority queue each element has a score, and the *pop* operation returns the element with the highest score. The A* decoding algorithm iteratively chooses the best prefix-so-far, computes all the possible next words for that prefix, and adds these extended sentences to the queue. Fig. 10.7 shows the complete algorithm.

Priority queue

Let's consider a stylized example of an A* decoder working on a waveform for which the correct transcription is *If music be the food of love*. Fig. 10.8 shows the search space after the decoder has examined paths of length one from the root. A **fast match** is used to select the likely next words. A fast match is one of a class of heuristics designed to efficiently winnow down the number of possible following words, often by computing some approximation to the forward probability (see below for further discussion of fast matching).

Fast match

At this point in our example, we've done the fast match, selected a subset of the

function STACK-DECODING() **returns** *min-distance*

Initialize the priority queue with a null sentence.
Pop the best (highest score) sentence s off the queue.
If (s is marked end-of-sentence (EOS)) output s and terminate.
Get list of candidate next words by doing fast matches.
For each candidate next word w:
    Create a new candidate sentence $s+w$.
    Use forward algorithm to compute acoustic likelihood L of $s+w$
    Compute language model probability P of extended sentence $s+w$
    Compute "score" for $s+w$ (a function of L, P, and ???)
    if (end-of-sentence) set EOS flag for $s+w$.
    Insert $s+w$ into the queue together with its score and EOS flag

Figure 10.7 The A* decoding algorithm (modified from Paul (1991) and Jelinek (1997)). The evaluation function that is used to compute the score for a sentence is not completely defined here; possible evaluation functions are discussed below.

possible next words, and assigned each of them a score. The word *Alice* has the highest score. We haven't yet said exactly how the scoring works.

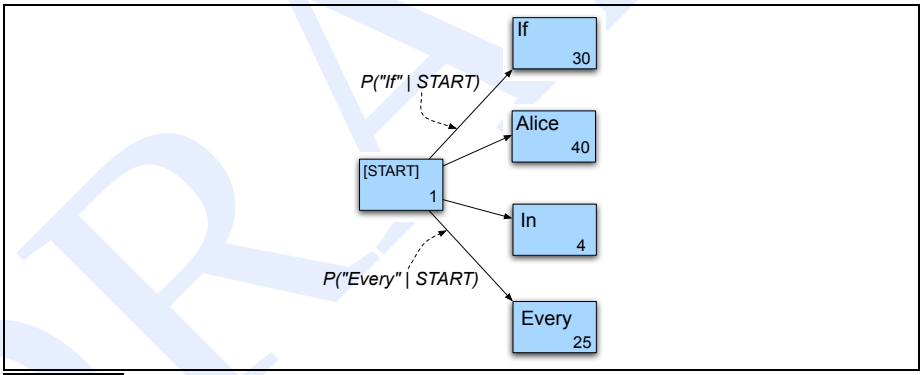

Figure 10.8 The beginning of the search for the sentence *If music be the food of love.* At this early stage *Alice* is the most likely hypothesis. (It has a higher score than the other hypotheses.)

Fig. 10.9a show the next stage in the search. We have expanded the *Alice* node. This means that the *Alice* node is no longer on the queue, but its children are. Note that now the node labeled *if* actually has a higher score than any of the children of *Alice*. Fig. 10.9b shows the state of the search after expanding the *if* node, removing it, and adding *if music*, *if muscle*, and *if messy* on to the queue.

We clearly want the scoring criterion for a hypothesis to be related to its probability. Indeed it might seem that the score for a string of words w_1^i given an acoustic string y_1^j should be the product of the prior and the likelihood:

$$P(y_1^j|w_1^i)P(w_1^i)$$

Alas, the score cannot be this probability because the probability will be much smaller for a longer path than a shorter one. This is due to a simple fact about prob-

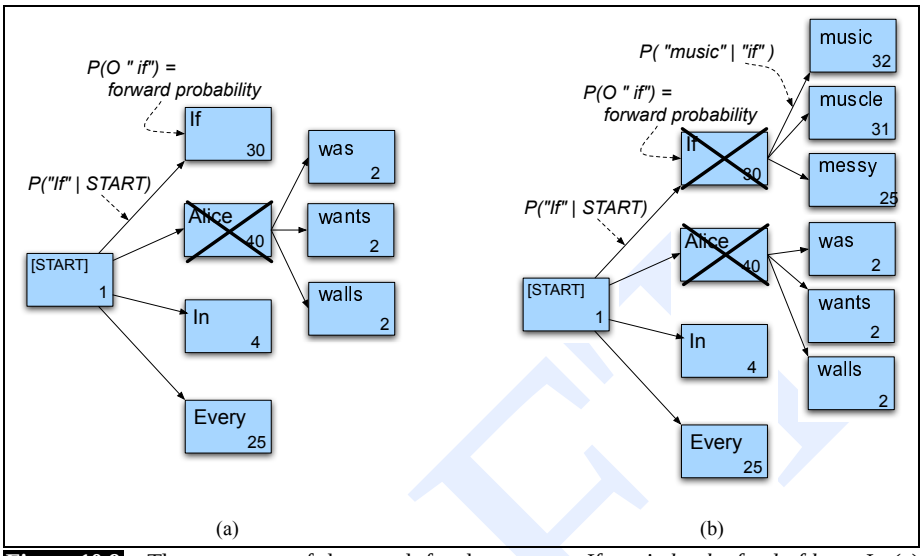

Figure 10.9 The next steps of the search for the sentence *If music be the food of love.* In (a) we've now expanded the *Alice* node and added three extensions which have a relatively high score; the highest-scoring node is *START if*, which is not along the *START Alice* path at all. In (b) we've expanded the *if* node. The hypothesis *START if music* then has the highest score.

abilities and substrings; any prefix of a string must have a higher probability than the string itself (e.g., P(START the ...) will be greater than P(START the book)). Thus if we used probability as the score, the A* decoding algorithm would get stuck on the single-word hypotheses.

Instead, we use the A* evaluation function (Nilsson, 1980; Pearl, 1984) $f^*(p)$, given a partial path p:

$$f^*(p) = g(p) + h^*(p)$$

$f^*(p)$ is the *estimated* score of the best complete path (complete sentence) which starts with the partial path p. In other words, it is an estimate of how well this path would do if we let it continue through the sentence. The A* algorithm builds this estimate from two components:

- $g(p)$ is the score from the beginning of utterance to the end of the partial path p. This g function can be nicely estimated by the probability of p given the acoustics so far (i.e., as $P(O|W)P(W)$ for the word string W constituting p).
- $h^*(p)$ is an estimate of the best scoring extension of the partial path to the end of the utterance.

Coming up with a good estimate of h^* is an unsolved and interesting problem. A very simple approach is to chose an h^* estimate which correlates with the number of words remaining in the sentence (Paul, 1991). Slightly smarter is to estimate the expected likelihood per frame for the remaining frames, and multiple this by the estimate of the remaining time. This expected likelihood can be computed by averaging the likelihood per frame in the training set. See Jelinek (1997) for further discussion.

Tree Structured Lexicons

Tree-structured lexicon

We mentioned above that both the A* and various other two-stage decoding algorithms require the use of a **fast match** for quickly finding which words in the lexicon are likely candidates for matching some portion of the acoustic input. Many fast match algorithms are based on the use of a **tree-structured lexicon**, which stores the pronunciations of all the words in such a way that the computation of the forward probability can be shared for words which start with the same sequence of phones. The tree-structured lexicon was first suggested by Klovstad and Mondshein (1975); fast match algorithms which make use of it include Gupta et al. (1988), Bahl et al. (1992) in the context of A* decoding, and Ney et al. (1992) and Nguyen and Schwartz (1999) in the context of Viterbi decoding. Fig. 10.10 shows an example of a tree-structured lexicon from the Sphinx-II recognizer (Ravishankar, 1996). Each tree root represents the first phone of all words beginning with that context dependent phone (phone context may or may not be preserved across word boundaries), and each leaf is associated with a word.

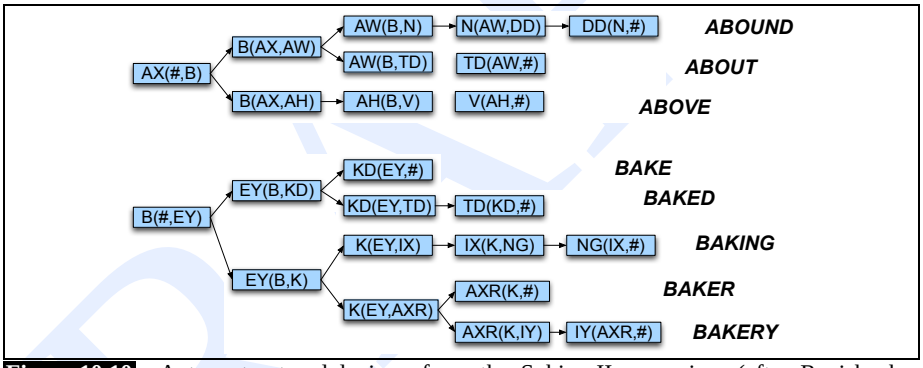

Figure 10.10 A tree-structured lexicon from the Sphinx-II recognizer (after Ravishankar (1996)). Each node corresponds to a particular triphone in the slightly modified version of the ARPAbet used by Sphinx-II. Thus EY(B,KD) means the phone EY preceded by a B and followed by the closure of a K.

10.3 Context-Dependent Acoustic Models: Triphones

In our discussion in Sec. 9.4 of how the HMM architecture is applied to ASR, we showed how an HMM could be created for each phone, with its three emitting states corresponding to subphones at the beginning, middle, and end of the phone. We thus represent each subphone ("beginning of [eh]", "beginning of [t]", "middle of [ae]") with its own GMM.

There is a problem with using a fixed GMM for a subphone like "beginning of [eh]". The problem is that phones vary enormously based on the phones on either side. This is because the movement of the articulators (tongue, lips, velum) during speech production is continuous and is subject to physical constraints like momentum. Thus

Coarticulation

an articulator may start moving during one phone to get into place in time for the next phone. In Ch. 7 we defined the word **coarticulation** as the movement of articulators to anticipate the next sound, or perseverating movement from the last sound. Fig. 10.11 shows coarticulation due to neighboring phone contexts for the vowel [eh].

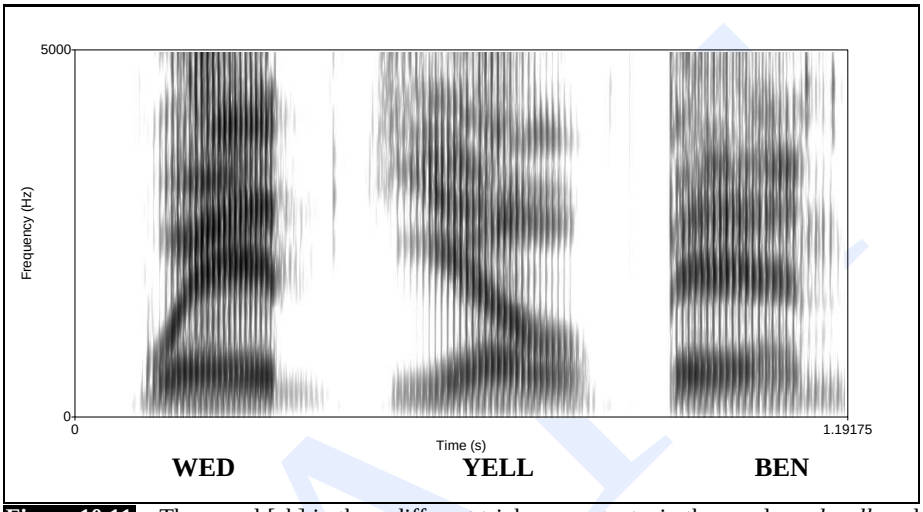

Figure 10.11 The vowel [eh] in three different triphone contexts, in the words *wed*, *yell*, and *Ben*. Notice the marked differences in the second formant (F2) at the beginning and end of the [eh] in all three cases.

CI phone
CD phones
Triphone

In order to model the marked variation that a phone exhibits in different contexts, most LVCSR systems replace the idea of a context-independent (**CI phone**) HMM with a context-dependent or **CD phones**. The most common kind of context-dependent model is a **triphone** HMM (Schwartz et al., 1985; Deng et al., 1990). A triphone model represents a phone in a particular left and right context. For example the triphone *[y-eh+l]* means "[eh] preceded by [y] and followed by [l]". In general, [a-b+c] will mean "[b] preceded by [a] and followed by [c]". In situations where we don't have a full triphone context, we'll use [a-b] to mean "[b] preceded by [a]" and [b+c] to mean "[b] followed by [c]".

Context-dependent phones capture an important source of variation, and are a key part of modern ASR systems. But unbridled context-dependency also introduces the same problem we saw in language modeling: training data sparsity. The more complex the model we try to train, the less likely we are to have seen enough observations of each phone-type to train on. For a phoneset with 50 phones, in principle we would need 50^3 or 125,000 triphones. In practice not every sequence of three phones is possible (English doesn't seem to allow triphone sequences like [ae-eh+ow] or [m-j+t]). Young et al. (1994) found that 55,000 triphones are needed in the 20K Wall Street Journal task. But they found that only 18,500 of these triphones, i.e. less than half, actually occurred in the SI84 section of the WSJ training data.

Tied states

Because of the problem of data sparsity, we must reduce the number of triphone parameters that we need to train. The most common way to do this is by clustering some of the contexts together and **tying** subphones whose contexts fall into the same

cluster (Young and Woodland, 1994). For example, the beginning of a phone with an [n] on its left may look much like the beginning of a phone with an [m] on its left. We can therefore tie together the first (beginning) subphone of, say, the [m-eh+d] and [n-eh+d] triphones. Tying two states together means that they share the same Gaussians. So we only train a single Gaussian model for the first subphone of the [m-eh+d] and [n-eh+d] triphones. Likewise, it turns out that the left context phones [r] and [w] produce a similar effect on the initial subphone of following phones.

Fig. 10.12 shows, for example the vowel [iy] preceded by the consonants [w], [r], [m], and [n]. Notice that the beginning of [iy] has a similar rise in F2 after [w] and [r]. And notice the similarity of the beginning of [m] and [n]; as Ch. 7 noted, the position of nasal formants varies strongly across speakers, but this speaker (the first author) has a nasal formant (N2) around 1000 Hz.

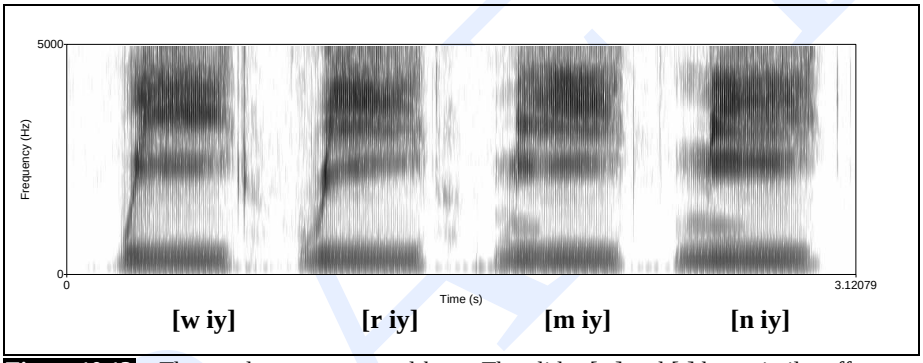

Figure 10.12 The words *we*, *re*, *me*, and *knee*. The glides [w] and [r] have similar effects on the beginning of the vowel [iy], as do the two nasals [n] and [m].

Fig. 10.13 shows an example of the kind of triphone tying learned by the clustering algorithm. Each mixture Gaussian model is shared by the subphone states of various triphone HMMs.

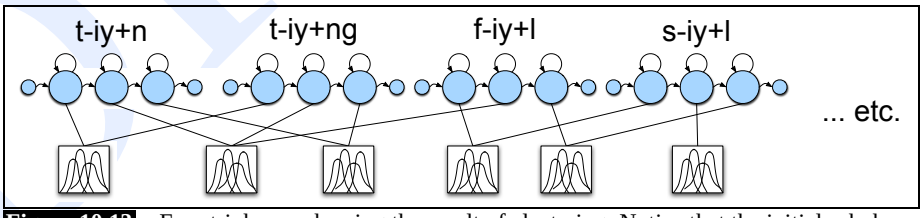

Figure 10.13 Four triphones showing the result of clustering. Notice that the initial subphone of [t-iy+n] and [t-iy+ng] is tied together, i.e. shares the same Gaussian mixture acoustic model. After Young et al. (1994).

How do we decide what contexts to cluster together? The most common method is to use a decision tree. For each state (subphone) of each phone, a separate tree is built. Fig. 10.14 shows a sample tree from the first (beginning) state of the phone /ih/, modified from Odell (1995). We begin at the root node of the tree with a single large cluster containing (the beginning state of) all triphones centered on /ih/. At each node

in the tree, we split the current cluster into two smaller clusters by asking questions about the context. For example the tree in Fig. 10.14 first splits the initial cluster into two clusters, one with nasal phone on the left, and one without. As we descend the tree from the root, each of these clusters is progressively split. The tree in Fig. 10.14 would split all beginning-state /ih/ triphones into 5 clusters, labeled A-E in the figure.

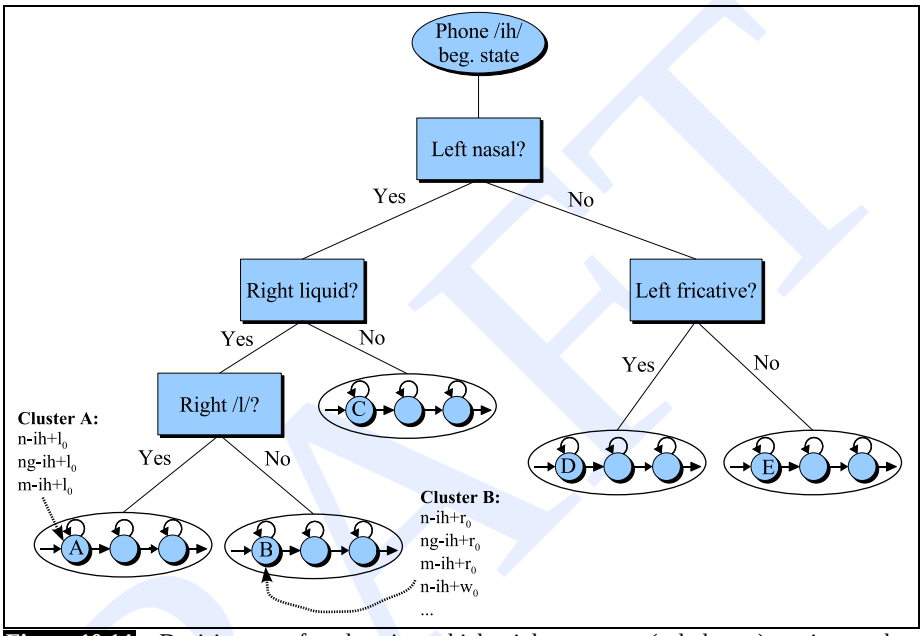

Figure 10.14 Decision tree for choosing which triphone states (subphones) to tie together. This particular tree will cluster state 0 (the beginning state) of the triphones /n-ih+l/, /ng-ih+l/, /m-ih+l/, into cluster class A, and various other triphones into classes B-E. Adapted from Odell (1995).

The questions used in the decision tree ask whether the phone to the left or right has a certain **phonetic feature**, of the type introduced in Ch. 7. Fig. 10.3 shows a few decision tree questions; note that there are separate questions for vowels and consonants. Real trees would have many more questions.

How are decision trees like the one in Fig. 10.14 trained? The trees are grown top down from the root. At each iteration, the algorithm considers each possible question q and each node n in the tree. For each question, it considers how the new split would impact the acoustic likelihood of the training data. The algorithm computes the difference between the current acoustic likelihood of the training data, and the new likelihood if the models were tied based on splitting via question q. The algorithm picks the node n and question q that give the maximum likelihood. The procedure then iterates, stopping when each leaf node has a minimum threshold number of examples.

We also need to modify the embedded training algorithm we saw in Sec. 9.7 to deal with context-dependent phones and also to handle mixture Gaussians. In both cases we use a more complex process that involves **cloning** and using extra iterations of EM, as described in Young et al. (1994).

Cloning

Feature	Phones
Stop	b d g k p t
Nasal	m n ng
Fricative	ch dh f jh s sh th v z zh
Liquid	l r w y
Vowel	aa ae ah ao aw ax axr ay eh er ey ih ix iy ow oy uh uw
Front Vowel	ae eh ih ix iy
Central Vowel	aa ah ao axr er
Back Vowel	ax ow uh uw
High Vowel	ih ix iy uh uw
Rounded	ao ow oy uh uw w
Reduced	ax axr ix
Unvoiced	ch f hh k p s sh t th
Coronal	ch d dh jh l n r s sh t th z zh

Figure 10.15 Sample decision tree questions on phonetic features used by Odell (1995).

To train context-dependent models, for example, we first use the standard embedded training procedure to train context-independent models, using multiple passes of EM and resulting in separate single-Gaussians models for each subphone of each monophone /aa/, /ae/, etc. We then **clone** each monophone model, i.e. make identical copies of the model with its 3 substates of Gaussians, one clone for each potential triphone. The *A* transition matrices are not cloned, but tied together for all the triphone clones of a monophone. We then run an iteration of EM again and retrain the triphone Gaussians. Now for each monophone we cluster all the context-dependent triphones using the clustering algorithm described on page 350 to get a set of tied state clusters. One typical state is chosen as the exemplar for this cluster and the rest are tied to it.

We use this same cloning procedure to learn Gaussian mixtures. We first use embedded training with multiple iterations of EM to learn single-mixture Gaussian models for each tied triphone state as described above. We then clone (split) each state into 2 identical Gaussians, perturb the values of each by some epsilon, and run EM again to retrain these values. We then split each of the two mixtures, resulting in four, perturb them, retrain. We continue until we have an appropriate number of mixtures for the amount of observations in each state.

A full context-depending GMM triphone model is thus created by applying these two cloning-and-retraining procedures in series, as shown schematically in Fig. 10.16.

10.4 Discriminative Training

MLE
Discriminative training

The Baum-Welch and embedded training models we have presented for training the HMM parameters (the *A* and *B* matrices) are based on maximizing the likelihood of the training data. An alternative to this **maximum likelihood estimation** (**MLE**) is to focus not on fitting the best model to the data, but rather on **discriminating** the best model from all the other models. Such training procedures include Maximum Mutual Information Estimation (MMIE) (Woodland and Povey, 2002) the use of neural net/SVM classifiers (Bourlard and Morgan, 1994) as well as other techniques like Minimum Classification Error training (Chou et al., 1993; McDermott and Hazen, 2004) or Minimum Bayes Risk estimation (Doumpiotis et al., 2003a). We summarize the first

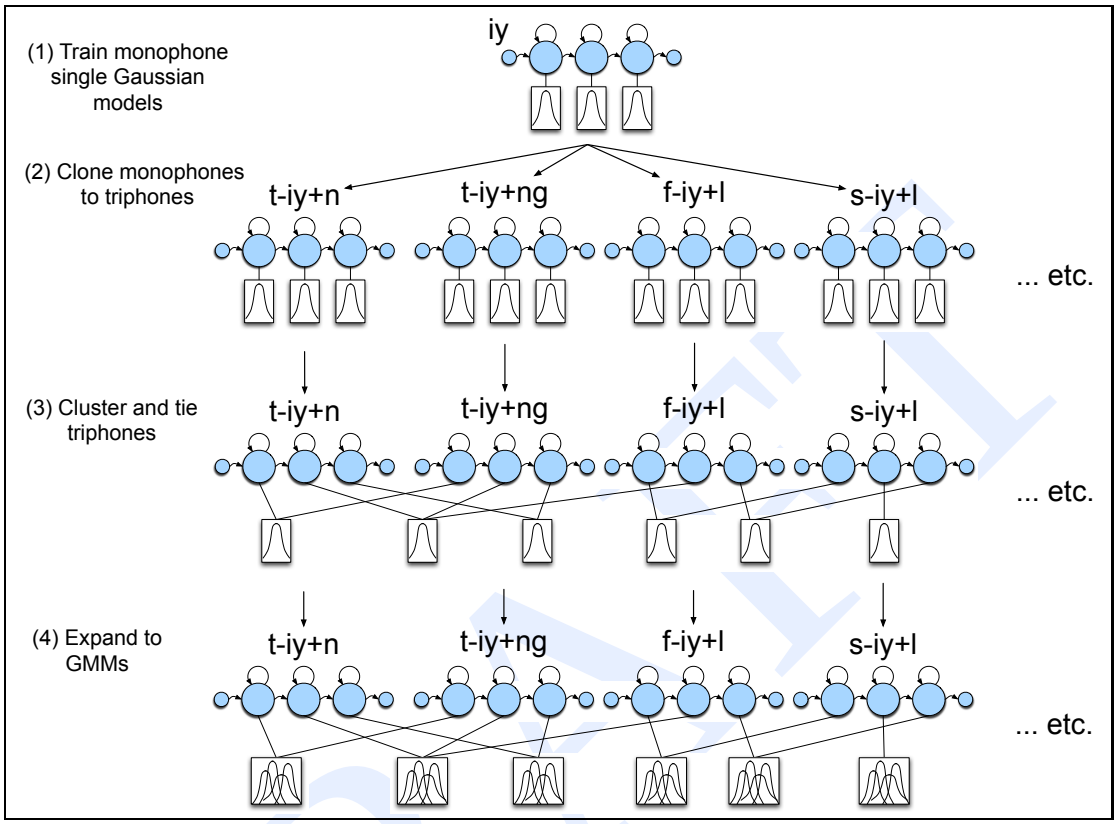

Figure 10.16 The four stages in training a tied-mixture triphone acoustic model. After Young et al. (1994).

two of these in the next two subsections.

10.4.1 Maximum Mutual Information Estimation

Recall that in Maximum Likelihood Estimation (MLE), we train our acoustic model parameters (A and B) so as to maximize the likelihood of the training data. Consider a particular observation sequence O, and a particular HMM model M_k corresponding to word sequence W_k, out of all the possible sentences $W' \in \mathscr{L}$. The MLE criterion thus maximizes

$$\mathscr{F}_{\text{MLE}}(\lambda) = P_\lambda(O|M_k) \tag{10.4}$$

Since our goal in speech recognition is to have the correct transcription for the largest number of sentences, we'd like on average for the probability of the **correct** word string W_k to be high; certainly higher than the probability of all the **wrong** word strings $W_j s.t. j \neq k$. But the MLE criterion above does not guarantee this. Thus we'd like to pick some other criterion which will let us chose the model λ which assigns the highest probability to the correct model, i.e. maximizes $P_\lambda(M_k|O)$. Maximizing the probability of the word string rather than the probability of the observation sequence is called **conditional maximum likelihood estimation** or **CMLE**:

CMLE

$$\mathcal{F}_{\text{CMLE}}(\lambda) = P_\lambda(M_k|O) \tag{10.5}$$

Using Bayes Law, we can express this as

$$\mathcal{F}_{\text{CMLE}}(\lambda) = P_\lambda(M_k|O) = \frac{P_\lambda(O|M_k)P(M_k)}{P_\lambda(O)} \tag{10.6}$$

Let's now expand $P_\lambda(O)$ by marginalizing (summing over all sequences which could have produced it). The total probability of the observation sequence is the weighted sum over all word strings of the observation likelihood given that word string:

$$P(O) = \sum_{W \in \mathcal{L}} P(O|W)P(W) \tag{10.7}$$

So a complete expansion of Eq. 10.6 is:

$$\mathcal{F}_{\text{CMLE}}(\lambda) = P_\lambda(M_k|O) = \frac{P_\lambda(O|M_k)P(M_k)}{\sum_{M \in \mathcal{L}} P_\lambda(O|M)P(M)} \tag{10.8}$$

In a slightly confusing bit of standard nomenclature, CMLE is generally referred to instead as Maximum Mutual Information Estimation (MMIE). This is because it turns out that maximizing the posterior $P(W|O)$ and maximizing the mutual information $I(W,O)$ are equivalent if we assume that the language model probability of each sentence W is constant (fixed) during acoustic training, an assumption we usually make. Thus from here on we will refer to this criterion as the MMIE criterion rather than the CMLE criterion, and so here is Eq. 10.8 restated:

$$\mathcal{F}_{\text{MMIE}}(\lambda) = P_\lambda(M_k|O) = \frac{P_\lambda(O|M_k)P(M_k)}{\sum_{M \in \mathcal{L}} P_\lambda(O|M)P(M)} \tag{10.9}$$

In a nutshell, then, the goal of MMIE estimation is to maximize (10.9) rather than (10.4). Now if our goal is to maximize $P_\lambda(M_k|O)$, we not only need to maximize the numerator of (10.9), but also minimize the denominator. Notice that we can rewrite the denominator to make it clear that it includes a term equal to the model we are trying to maximize and a term for all other models:

$$P_\lambda(M_k|O) = \frac{P_\lambda(O|M_k)P(M_k)}{P_\lambda(O|M_k)P(M_k) + \sum_{i \neq k} P_\lambda(O|M_i)P(M_i)} \tag{10.10}$$

Thus in order to maximize $P_\lambda(M_k|O)$, we will need to incrementally change λ so that it increases the probability of the correct model, while simultaneously decreasing the probability of each of the incorrect models. Thus training with MMIE clearly fulfills the important goal of **discriminating** between the correct sequence and all other sequences.

The implementation of MMIE is quite complex, and we don't discuss it here except to mention that it relies on a variant of Baum-Welch training called Extended Baum-Welch that maximizes (10.9) instead of (10.4). Briefly, we can view this as a two step algorithm; we first use standard MLE Baum-Welch to compute the forward-backward counts for the training utterances. Then we compute another forward-backward pass

using all other possible utterances and subtract these from the counts. Of course it turns out that computing this full denominator is computationally extremely expensive, because it requires running a full recognition pass on all the training data. Recall that in normal EM, we don't need to run decoding on the training data, since we are only trying to maximize the likelihood of the *correct* word sequence; in MMIE, we need to compute the probabilities of *all* possible word sequences. Decoding is very time-consuming because of complex language models. Thus in practice MMIE algorithms estimate the denominator by summing over only the paths that occur in a word lattice, as an approximation to the full set of possible paths.

CMLE was first proposed by Nádas (1983) and MMIE by Bahl et al. (1986), but practical implementations that actually reduced word error rate came much later; see Woodland and Povey (2002) or Normandin (1996) for details.

10.4.2 Acoustic Models based on Posterior Classifiers

Another way to think about discriminative training is to choose a classifier at the frame level which is discriminant. Thus while the Gaussian classifier is by far the most commonly used acoustic likelihood classifier, it is possible to instead use classifiers that are naturally discriminative or posterior estimators, such as neural networks or SVMs (support vector machines).

The posterior classifier (neural net or SVM) is generally integrated with an HMM architecture, is often called a **HMM-SVM** or **HMM-MLP hybrid** approach (Bourlard and Morgan, 1994).

The SVM or MLP approaches, like the Gaussian model, estimate the probability with respect to a cepstral feature vector at a single time t. Unlike the Gaussian model, the posterior approaches often uses a larger window of acoustic information, relying on cepstral feature vectors from neighboring time periods as well. Thus the input to a typical acoustic MLP or SVM might be feature vectors for the current frame plus the four previous and four following frames, i.e. a total of 9 cepstral feature vectors instead of the single one that the Gaussian model uses. Because they have such a wide context, SVM or MLP models generally use phones rather than subphones or triphones, and compute a posterior for each phone.

The SVM or MLP classifiers are thus computing the posterior probability of a state j given the observation vectors, i.e. $P(q_j|o_t)$. (also conditioned on the context, but let's ignore that for the moment). But the observation likelihood we need for the HMM, $b_j(o_t)$, is $P(o_t|q_j)$. The Bayes rule can help us see how to compute one from the other. The net is computing:

$$p(q_j|o_t) = \frac{P(o_t|q_j)p(q_j)}{p(o_t)} \tag{10.11}$$

We can rearrange the terms as follows:

$$\frac{p(o_t|q_j)}{p(o_t)} = \frac{P(q_j|o_t)}{p(q_j)} \tag{10.12}$$

The two terms on the right-hand side of (10.12) can be directly computed from the posterior classifier; the numerator is the output of the SVM or MLP, and the denomi-

nator is the total probability of a given state, summing over all observations (i.e., the sum over all t of $\xi_j(t)$). Thus although we cannot directly compute $P(o_t|q_j)$, we *can* use (10.12) to compute $\frac{p(o_t|q_j)}{p(o_t)}$, which is known as a **scaled likelihood** (the likelihood divided by the probability of the observation). In fact, the scaled likelihood is just as good as the regular likelihood, since the probability of the observation $p(o_t)$ is a constant during recognition and doesn't hurt us to have in the equation.

Scaled likelihood

The supervised training algorithms for training a SVM or MLP posterior phone classifiers require that we know the correct phone label q_j for each observation o_t. We can use the same **embedded training** algorithm that we saw for Gaussians; we start with some initial version of our classifier and a word transcript for the training sentences. We run a forced alignment of the training data, producing a phone string, and now we retrain the classifier, and iterate.

10.5 Modeling Variation

As we noted at the beginning of this chapter, variation is one of the largest obstacles to successful speech recognition. We mentioned variation due to speaker differences from vocal characteristics or dialect, due to genre (such as spontaneous versus read speech), and due to the environment (such as noisy versus quiet environments). Handling this kind of variation is a major subject of modern research.

10.5.1 Environmental Variation and Noise

Environmental variation has received the most attention from the speech literature, and a number of techniques have been suggested for dealing with environmental noise. **Spectral subtraction**, for example, is used to combat **additive noise**. Additive noise is noise from external sound sources like engines or wind or fridges that is relatively constant and can be modeled as a noise signal that is just added in the time domain to the speech waveform to produce the observed signal. In spectral subtraction, we estimate the average noise during non-speech regions and then subtract this average value from the speech signal. Interestingly, speakers often compensate for high background noise levels by increasing their amplitude, F0, and formant frequencies. This change in speech production due to noise is called the **Lombard effect**, named for Etienne Lombard who first described it in 1911 (Junqua, 1993).

Spectral subtraction
Additive noise
Lombard effect

Other noise robustness techniques like **cepstral mean normalization** are used to deal with **convolutional noise**, noise introduced by channel characteristics like different microphones. Here we compute the average of the cepstrum over time and subtract it from each frame; the average cepstrum models the fixed spectral characteristics of the microphone and the room acoustics (Atal, 1974).

Cepstral mean normalization
Convolutional noise

Finally, some kinds of short non-verbal sounds like coughs, loud breathing, and throat clearing, or environmental sounds like beeps, telephone rings, and door slams, can be modeled explicitly. For each of these non-verbal sounds, we create a special phone and add to the lexicon a word consisting only of that phone. We can then use

normal Baum-Welch training to train these phones just by modifying the training data transcripts to include labels for these new non-verbal 'words' (Ward, 1989). These words also need to be added to the language model; often by just allowing them to appear in between any word.

10.5.2 Speaker and Dialect Adaptation: Variation due to speaker differences

Speech recognition systems are generally designed to be speaker-independent, since it's rarely practical to collect sufficient training data to build a system for a single user. But in cases where we have enough data to build speaker-dependent systems, they function better than speaker-independent systems. This only makes sense; we can reduce the variability and increase the precision of our models if we are guaranteed that the test data will look more like the training data.

While it is rare to have enough data to train on an individual speaker, we do have enough data to train separate models for two important groups of speakers: men versus women. Since women and men have different vocal tracts and other acoustic and phonetic characteristics, we can split the training data by gender, and train separate acoustic models for men and for women. Then when a test sentence comes in, we use a gender detector to decide if it is male or female, and switch to those acoustic models. Gender detectors can be built out of binary GMM classifiers based on cepstral features. Such **gender-dependent acoustic modeling** is used in most LVCSR systems.

MLLR

Speaker adaptation

Although we rarely have enough data to train on a specific speaker, there are techniques that work quite well at adapting the acoustic models to a new speaker very quickly. For example the **MLLR** (**Maximum Likelihood Linear Regression**) technique (Leggetter and Woodland, 1995) is used to adapt Gaussian acoustic models to a small amount of data from a new speaker. The idea is to use the small amount of data to train a linear transform to warp the means of the Gaussians. MLLR and other such techniques for **speaker adaptation** have been one of the largest sources of improvement in ASR performance in recent years.

The MLLR algorithm begins with a trained acoustic model and a small adaptation dataset from a new speaker. The adaptation set can be as small as 3 sentences or 10 seconds of speech. The idea is to learn a linear transform matrix (W) and a bias vector (ω) to transform the means of the acoustic model Gaussians. If the old mean of a Gaussian is μ, the equation for the new mean $\hat{\mu}$ is thus:

$$\hat{\mu} = W\mu + \omega \tag{10.13}$$

In the simplest case, we can learn a single global transform and apply it to each Gaussian models. The resulting equation for the acoustic likelihood is thus only very slightly modified:

$$b_j(o_t) = \frac{1}{\sqrt{2\pi|\Sigma j|}} \exp\left(-\frac{1}{2}(o_t - (W\mu_j + \omega))^T \Sigma_j^{-1}(o_t - (W\mu_j + \omega))\right) \tag{10.14}$$

The transform is learned by using linear regression to maximize the likelihood of the adaptation dataset. We first run forward-backward alignment on the adaptation set

to compute the state occupation probabilities $\xi_j(t)$. We then compute W by solving a system of simultaneous equations involving $\xi_j(t)$. If enough data is available, it's also possible to learn a larger number of transforms.

MLLR is an example of the **linear transform** approach to speaker adaptation, one of the three major classes of speaker adaptation methods; the other two are **MAP adaptation** and **Speaker Clustering/Speaker Space** approaches. See Woodland (2001) for a comprehensive survey of speaker adaptation which covers all three families.

MLLR and other speaker adaptation algorithms can also be used to address another large source of error in LVCSR, the problem of foreign or dialect accented speakers. Word error rates go up when the test set speaker speaks a dialect or accent (such as Spanish-accented English or southern accented Mandarin Chinese) that differs from the (usually standard) training set, Here we can take an adaptation set of a few sentences from say 10 speakers, and adapt to them as a group, creating an MLLR transform that addresses whatever characteristics are present in the dialect or accent (Huang et al., 2000; Tomokiyo and Waibel, 2001; Wang et al., 2003; Zheng et al., 2005).

Another useful speaker adaptation technique is to control for the differing vocal tract lengths of speakers. Cues to the speaker's vocal tract length are present in the signal; for example speakers with longer vocal tracts tend to have lower formants. *VTLN* Vocal tract length can therefore be detected and normalized, in a process called **VTLN** (**Vocal Tract Length Normalization**); see the end notes for details.

10.5.3 Pronunciation Modeling: Variation due to Genre

We said at the beginning of the chapter that recognizing conversational speech is harder for ASR systems than recognizing read speech. What are the causes of this difference? Is it the difference in vocabulary? Grammar? Something about the speaker themselves? Perhaps it's a fact about the microphones or telephone used in the experiment.

None of these seems to be the cause. In a well-known experiment, Weintraub et al. (1996) compared ASR performance on natural conversational speech versus performance on read speech, controlling for the influence of possible causal factors. Pairs of subjects in the lab had spontaneous conversations on the telephone. Weintraub et al. (1996) then hand-transcribed the conversations, and invited the participants back into the lab to read their own transcripts to each other over the same phone lines as if they were dictating. Both the natural and read conversations were recorded. Now Weintraub et al. (1996) had two speech corpora from identical transcripts; one original natural conversation, and one read speech. In both cases the speaker, the actual words, and the microphone were identical; the only difference was the naturalness or fluency of the speech. They found that read speech was much easier (WER=29%) than conversational speech (WER=53%). Since the speakers, words, and channel were controlled for, this difference must be modelable somewhere in the acoustic model or pronunciation lexicon.

Saraclar et al. (2000) tested the hypothesis that this difficulty with conversational speech was due to changed pronunciations, i.e., to a mismatch between the phone strings in the lexicon and what people actually said. Recall from Ch. 7 that conversational corpora like Switchboard contain many different pronunciations for words, (such as 12 different pronunciations for *because* and hundreds for *the*). Saraclar et al.

(2000) showed in an oracle experiment that if a Switchboard recognizer is told which pronunciations to use for each word, the word error rate drops from 47% to 27%.

If knowing which pronunciation to use improves accuracy, could we improve recognition by simply adding more pronunciations for each word to the lexicon?

Alas, it turns out that adding multiple pronunciations doesn't work well, even if the list of pronunciation is represented as an efficient pronunciation HMM (Cohen, 1989). Adding extra pronunciations adds more confusability; if a common pronunciation of the word "of" is the single vowel [ax], it is now very confusable with the word "a". Another problem with multiple pronunciations is the use of Viterbi decoding. As we said on page 338, the Viterbi decoder finds the best phone string, rather than the best word string, hence biasing against words with many pronunciations. Finally, using multiple pronunciations to model coarticulatory effects may be unnecessary because CD phones (triphones) are already quite good at modeling the contextual effects in phones due to neighboring phones like flapping and vowel-reduction (Jurafsky et al., 2001).

Instead, most current LVCSR systems use a very small number of pronunciations per word. What is commonly done is to start with a multiple pronunciation lexicon, where the pronunciations are found in dictionaries or are generated via phonological rules of the type described in Ch. 7. A forced Viterbi phone alignment is then run of the training set, using this dictionary. The result of the alignment is a phonetic transcription of the training corpus, showing which pronunciation was used, and the frequency of each pronunciation. We can then collapse similar pronunciations (for example if two pronunciations differ only in a single phone substitution we chose the more frequent pronunciation). We then chose the maximum likelihood pronunciation for each word. For frequent words which have multiple high-frequency pronunciations, some systems chose multiple pronunciations, and annotate the dictionary with the probability of these pronunciations; the probabilities are used in computing the acoustic likelihood (Cohen, 1989; Hain et al., 2001; Hain, 2002).

Finding a better method to deal with pronunciation variation remains an unsolved research problem. One promising avenue is to focus on non-phonetic factors that affect pronunciation. For example words which are highly predictable, or at the beginning or end of intonation phrases, or are followed by disfluencies, are pronounced very differently (Jurafsky et al., 1998; Fosler-Lussier and Morgan, 1999; Bell et al., 2003). Fosler-Lussier (1999) shows reductions in word errors by using these sorts of factors to predict which pronunciation to use. Another exciting line of research in pronunciation modeling uses a dynamic Bayesian network to model the complex overlap in articulators that produces phonetic reduction (Livescu and Glass, 2004b, 2004a).

Another important issue in pronunciation modeling is dealing with unseen words. In web-based applications such as telephone-based interfaces to the Web, the recognizer lexicon must be automatically augmented with pronunciations for the millions of unseen words, particularly names, that occur on the Web. Grapheme-to-phoneme techniques like those described in Sec. 8.2.3 are used to solve this problem.

10.6 Metadata: Boundaries, Punctuation, and Disfluencies

The output of the speech recognition process as we have described it so far is just a string of raw words. Consider the following sample gold-standard transcript (i.e., assuming perfect word recognition) of part of a dialogue (Jones et al., 2003):

> yeah actually um i belong to a gym down here a gold's gym uh-huh and uh exercise i try to exercise five days a week um and i usually do that uh what type of exercising do you do in the gym

Compare the difficult transcript above with the following much clearer version:

A: Yeah I belong to a gym down here. Gold's Gym. And I try to exercise five days a week. And I usually do that.
B: What type of exercising do you do in the gym?

The raw transcript is not divided up among speakers, there is no punctuation or capitalization, and disfluencies are scattered among the words. A number of studies have shown that such raw transcripts are harder for people to read Jones et al. (2003, 2005) and that adding, for example, commas back into the transcript improve sthe accuracy of information extraction algorithms on the transcribed text (Makhoul et al., 2005; Hillard et al., 2006). Post-processing ASR output involves tasks including the following:

diarization

diarization: Many speech tasks have multiple speakers, such as telephone conversations, business meetings, and news reports (with multiple broadcasters). Diarization is the task of breaking up a speech file by speaker assigning parts of the transcript to the relevant speakers, like the **A:** and **B:** labels above.

sentence segmentation

sentence boundary detection: We discussed the task of breaking speech into sentences (sentence segmentation) in Ch. 3 and Ch. 8. But for those tasks we already add punctuation like periods to help us; from speech we don't already have punctuation, just words. Sentence segmentation from speech has the added difficulty that the transcribed words will be errorful, but has the advantage that prosodic features like pauses and sentence-final intontation can be used as cues.

Truecasing

truecasing: Words in a clean transcript need to have sentence-initial words starting with an upper-case letter, acronyms all in capitals, and so on. Truecasing is the task of assigning the correct case for a word, and is often addressed as a HMM classification task like part-of-speech tagging, with hidden states like ALL-LOWER CASE, UPPER-CASE-INITIAL, *all-caps*, and so on.

Punctuation detection

punctuation detection: In addition to segmenting sentences, we need to choose sentence-final punctuation (period, question mark, exclamation mark), and insert commas and quotation marks and so on.

Disfluency detection

disfluency detection: Disfluencies can be removed from a transcript for readability, or at least marked off with commas or font changes. Since standard recognizers don't actually include disfluencies (like word fragments) in their transcripts, disfluency detection algorithms can also play an important role in avoiding the misrecognized words that may result.

Metadata
Rich transcription

Marking these features (punctuation, boundaries, diarization) in the text output is often called **metadata** or sometimes **rich transcription**. Let's look at a couple of these tasks in slightly more detail.

Sentence segmentation can be modeled as a binary classification task, in which each boundary between two words is judged as a sentence boundary or as sentence-internal. Such classifiers can use similar features to the sentence segmentation discussed in Sec. 8.3.1, such as words and part-of-speech tags around each candidate boundary, or length features such as the distance from the previously found boundary. We can also make use of prosodic features, especially pause duration, word duration (recall that sentence-final words are lengthened), and pitch movements.

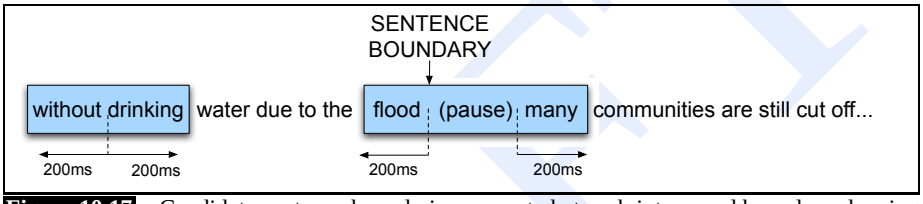

Figure 10.17 Candidate sentence boundaries computed at each inter-word boundary showing prosodic feature extraction regions from the Shriberg et al. (2000) algorithm.

Fig. 10.17 shows the candidate boundary locations in a sample sentence. Commonly extracted features include:

pause features: duration of the interword pause at the candidate boundary.

duration features: durations of the phone and rime (nucleus plus coda) preceding the candidate boundary. Since some phones are inherently longer than others, each phone is normalized to the mean duration for that phone.

F0 features: the **change in pitch** across the boundary; sentence boundaries often have **pitch reset** (an abrupt change in pitch), while non-boundaries are more likely to have continuous pitch across the boundary. Another useful F0 feature is the **pitch range** of the preboundary word; sentences often end with a **final fall** (Sec. 8.3.3) which is close to the speaker's F0 baseline.

For **punctuation detection**, similar features are used as for sentence boundary detection, but with multiple hidden classes (comma, sentence-final question mark, quotation mark, no punctuation). instead of just two.

For both of these tasks, instead of a simple binary classifier, sequence information can be incorporated by modeling sentence segmentation as an HMM in which the hidden states correspond to sentence boundary or non-boundary decisions. We will describe methods for combining prosodic and lexical features in more detail when we introduce dialogue act detection in Sec. 24.5.2.

Disfluencies
Repair

Recall from Sec. 12.8.1 that **disfluencies** or **repair** in conversation include phenomena like the following:

Disfluency type	Example
fillers (or **filled pauses**):	But, *uh*, that was absurd
word fragments	A guy went to a *d-*, a landfill
repetitions:	it was just a *change of*, *change of* location
restarts	it's – I find it very strange

Interruption point

The ATIS sentence in Fig. 10.18 shows examples of a restart and the filler *uh*, showing the **interruption point** that starts the **editing phase**.

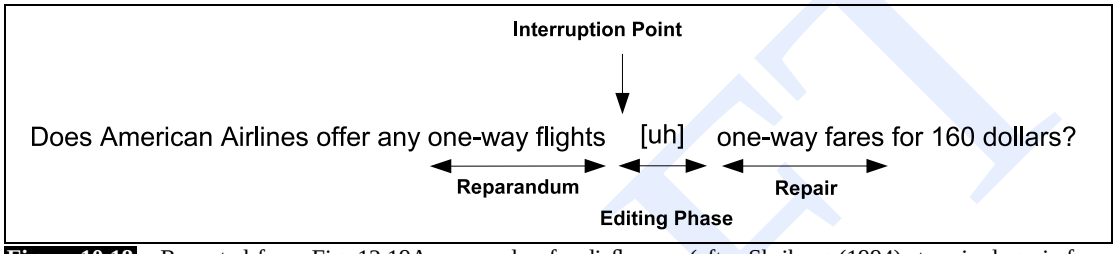

Figure 10.18 Repeated from Fig. 12.19An example of a disfluency (after Shriberg (1994); terminology is from Levelt (1983)).

Detection methods for disfluencies are very similar to detecting sentence boundaries; a classifier is trained to make a decision at each word boundary, using both text and prosodic features. HMM and CRF classifiers are commonly used, and features are quite similar to the features for boundary detection, including neighoring words and part-of-speech tags, the duration of pauses at the word boundary, the duration of the word and phones preceding the boundary, the difference in pitch values across the boundary, and so on.

Jitter
Spectral tilt
Open quotient

For detecting fragments, features for detecting voice quality are used (Liu, 2004), such as **jitter**, a measure of perturbation in the pitch period (Rosenberg, 1971), **spectral tilt**, the slope of the spectrum, (see Sec. 9.3.1), and **open quotient**, the percentage of the glottal cycle in which the vocal folds are open (Fant, 1997).

10.7 Speech Recognition by Humans

Humans are of course much better at speech recognition than machines; current machines are roughly about five times worse than humans on clean speech, and the gap seems to increase with noisy speech.

Lexical access

Speech recognition in humans shares some features with ASR algorithms. We mentioned above that signal processing algorithms like PLP analysis (Hermansky, 1990) were in fact inspired by properties of the human auditory system. In addition, three properties of human **lexical access** (the process of retrieving a word from the mental lexicon) are also true of ASR models: **frequency**, **parallelism**, and **cue-based processing**. For example, as in ASR with its N-gram language models, human lexical access is sensitive to word **frequency**. High-frequency spoken words are accessed faster or with less information than low-frequency words. They are successfully rec-

ognized in noisier environments than low frequency words, or when only parts of the words are presented (Howes, 1957; Grosjean, 1980; Tyler, 1984, inter alia). Like ASR models, human lexical access is **parallel**: multiple words are active at the same time (Marslen-Wilson and Welsh, 1978; Salasoo and Pisoni, 1985, inter alia).

Finally, human speech perception is **cue based**: speech input is interpreted by integrating cues at many different levels. Human phone perception combines acoustic cues, such as formant structure or the exact timing of voicing, (Oden and Massaro, 1978; Miller, 1994) visual cues, such as lip movement (McGurk and Macdonald, 1976; Massaro and Cohen, 1983; Massaro, 1998) and lexical cues such as the identity of the word in which the phone is placed (Warren, 1970; Samuel, 1981; Connine and Clifton, 1987; Connine, 1990). For example, in what is often called the **phoneme restoration effect**, Warren (1970) took a speech sample and replaced one phone (e.g. the [s] in *legislature*) with a cough. Warren found that subjects listening to the resulting tape typically heard the entire word *legislature* including the [s], and perceived the cough as background. In the **McGurk effect**, (McGurk and Macdonald, 1976) showed that visual input can interfere with phone perception, causing us to perceive a completely different phone. They showed subjects a video of someone saying the syllable *ga* in which the audio signal was dubbed instead with someone saying the syllable *ba*. Subjects reported hearing something like *da* instead. It is definitely worth trying this out yourself from video demos on the web; see for example `http://www.haskins.yale.edu/featured/heads/mcgurk.html`.

Phoneme restoration effect

McGurk effect

Word association

Repetition priming

Other cues in human speech perception include semantic **word association** (words are accessed more quickly if a semantically related word has been heard recently) and **repetition priming** (words are accessed more quickly if they themselves have just been heard). The intuitions of both these results are incorporated into recent language models discussed in Ch. 4, such as the cache model of Kuhn and De Mori (1990), which models repetition priming, or the trigger model of Rosenfeld (1996) and the LSA models of Coccaro and Jurafsky (1998) and Bellegarda (1999), which model word association. In a fascinating reminder that good ideas are never discovered only once, Cole and Rudnicky (1983) point out that many of these insights about context effects on word and phone processing were actually discovered by William Bagley (1901). Bagley achieved his results, including an early version of the phoneme restoration effect, by recording speech on Edison phonograph cylinders, modifying it, and presenting it to subjects. Bagley's results were forgotten and only rediscovered much later.[2]

One difference between current ASR models and human speech recognition is the time-course of the model. It is important for the performance of the ASR algorithm that the decoding search optimizes over the entire utterance. This means that the best sentence hypothesis returned by a decoder at the end of the sentence may be very different than the current-best hypothesis, halfway into the sentence. By contrast, there is extensive evidence that human processing is **on-line**: people incrementally segment and utterance into words and assign it an interpretation as they hear it. For example, Marslen-Wilson (1973) studied **close shadowers**: people who are able to shadow (repeat back) a passage as they hear it with lags as short as 250 ms. Marslen-Wilson found that when these shadowers made errors, they were syntactically and semanti-

On-line processing

[2] Recall the discussion on page 13 of multiple independent discovery in science.

cally appropriate with the context, indicating that word segmentation, parsing, and interpretation took place within these 250 ms. Cole (1973) and Cole and Jakimik (1980) found similar effects in their work on the detection of mispronunciations. These results have led psychological models of human speech perception (such as the Cohort model (Marslen-Wilson and Welsh, 1978) and the computational TRACE model (McClelland and Elman, 1986)) to focus on the time-course of word selection and segmentation. The TRACE model, for example, is a connectionist interactive-activation model, based on independent computational units organized into three levels: feature, phoneme, and word. Each unit represents a hypothesis about its presence in the input. Units are activated in parallel by the input, and activation flows between units; connections between units on different levels are excitatory, while connections between units on single level are inhibitatory. Thus the activation of a word slightly inhibits all other words.

We have focused on the similarities between human and machine speech recognition; there are also many differences. In particular, many other cues have been shown to play a role in human speech recognition but have yet to be successfully integrated into ASR. The most important class of these missing cues is prosody. To give only one example, Cutler and Norris (1988), Cutler and Carter (1987) note that most multisyllabic English word tokens have stress on the initial syllable, suggesting in their metrical segmentation strategy (MSS) that stress should be used as a cue for word segmentation. Another difference is that human lexical access exhibits **neighborhood effects** (the neighborhood of a word is the set of words which closely resemble it). Words with large frequency-weighted neighborhoods are accessed slower than words with less neighbors (Luce et al., 1990). Current models of ASR don't generally focus on this word-level competition.

10.8 Summary

- We introduced two advanced decoding algorithms: The multipass (N-best or lattice) decoding algorithm, and **stack** or $\mathbf{A}^*$ decoding.
- Advanced acoustic models are based on context-dependent **triphones** rather than phones. Because the complete set of triphones would be too large, we use a smaller number of automatically clustered triphones instead.
- Acoustic models can be **adapted** to new speakers.
- Pronunciation variation is a source of errors in human-human speech recognition, but one that is not successfully handled by current technology.

Bibliographical and Historical Notes

A^* search

See the previous chapter for most of the relevant speech recognition history. Note that although stack decoding is equivalent to the $\mathbf{A}^*$ **search** developed in artificial intelligence, the stack decoding algorithm was developed independently in the information

theory literature and the link with AI best-first search was noticed only later (Jelinek, 1976). Useful references on vocal tract length normalization include (Cohen et al., 1995; Wegmann et al., 1996; Eide and Gish, 1996; Lee and Rose, 1996; Welling et al., 2002; Kim et al., 2004).

There are many new directions in current speech recognition research involving alternatives to the HMM model. Such new architectures include new **graphical models** (dynamic bayes nets, factorial HMMs, etc) (Zweig, 1998; Bilmes, 2003; Livescu et al., 2003; Bilmes and Bartels, 2005; Frankel et al., 2007), as well as attempts to replace the **frame-based** HMM acoustic model (that make a decision about each frame) with **segment-based recognizers** that attempt to detect variable-length segments (phones) (Digilakis, 1992; Ostendorf et al., 1996; Glass, 2003). New **landmark-based** recognizers and articulatory phonology-based recognizers focus on the use of distinctive features, defined acoustically or articulatorily (respectively) (Niyogi et al., 1998; Livescu, 2005; Hasegawa-Johnson and et al, 2005; Juneja and Espy-Wilson, 2003).

Frame-based
Segment-based

See Shriberg (2005) for an overview of metadata research. Shriberg (2002) and Nakatani and Hirschberg (1994) are computationally-focused corpus studies of the acoustic and lexical properties of disfluencies. Early papers on sentence segmentation from speech include Wang and Hirschberg (1992), Ostendorf and Ross (1997) See Shriberg et al. (2000), Liu et al. (2006a) for recent work on sentence segmentation, Kim and Woodland (2001), Hillard et al. (2006) on punctuation detection, Nakatani and Hirschberg (1994), Honal and Schultz (2003, 2005), Lease et al. (2006), and a number of papers that jointly address multiple metadata extraction tasks (Heeman and Allen, 1999; Liu et al., 2005, 2006b).

Exercises

10.1 Implement the Stack decoding algorithm of Fig. 10.7 on page 345. Pick a very simple h^* function like an estimate of the number of words remaining in the sentence.

10.2 Modify the forward algorithm of Fig. 9.23 from Ch. 9 to use the tree-structured lexicon of Fig. 10.10 on page 347.

10.3 Many ASR systems, including the Sonic and HTK systems, use a different algorithm for Viterbi called the **token-passing Viterbi** algorithm (Young et al., 1989). Read this paper and implement this algorithm.