EECS225d Audio Signal Processing in Humans and Machines **Spring 1995**

Lecture 35: April 17, 1995

Lecturer: Dan Jurafsky *Scribe: Jeff Bilmes*

The Architecture of Speech Understanding

In this lecture, we will discuss the architecture of a particular speech understanding system, the Berkeley Restaurant Project or BeRP[BRP94] that was developed at ICSI. Berp is a limited vocabulary and limited domain speaker independent continuous-speech recognition system.

35.1 Overview

A general speech recognizer often consists of four stages as described below:

feature extraction *"What acoustic features are there in the input?"*. Examples include RASTA or PLP processing, cepstral analysis, etc.

phonetic likelihood estimation *"What phonetic sounds are there in these features?"*. Examples include MLP estimation of phonetic probabilities, etc. This perhaps should be called *subword likelihood estimation* since we do not necessarily operate at the phoneme level.

decoding *"What is the string of words which the speaker intended to utter via those sounds?"* Examples include DTW, Forward or Viterbi HMM approaches to find the best word for a string of phonemes.

understanding *"What is the meaning of that string of words?* Examples include high level AI approaches to natural language understanding, such as stochastic grammars, cognitive models, etc.

.

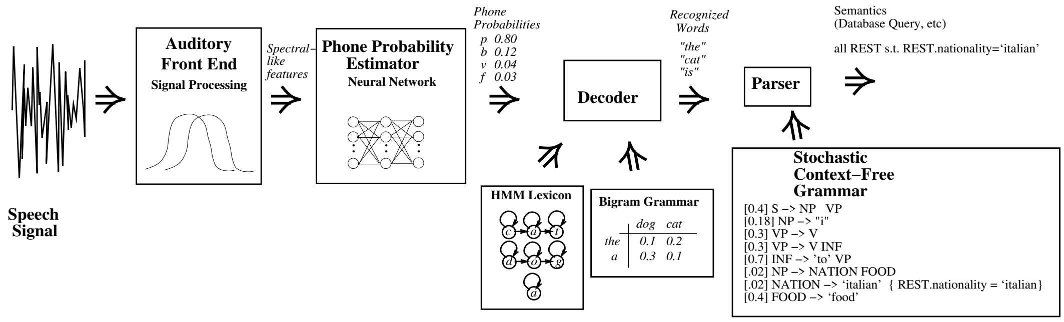

Figure 35.1: Speech Understanding Architecture

Such a speech understanding system is depicted in Figure 35.1. In this figure, we see (moving from right to left) that the process transforms an audio signal into semantics. First, an audio front end (such as PLP) transforms the audio signal into spectral-like features. The audio signal is broken into frames (often 10ms in length) and the output of the front end is a vector of features for every 10ms of input audio signal.

Second, a multi-layered perceptron network transforms each frame of features into a distribution over a set of phonemes. The MLP acts as a phonetic probability estimator.

Third, a decoder (Viterbi, Stack, or something else) transforms the sequence of phoneme distributions into a sequence of words. The Viterbi decoder uses Hidden Markov Model (HMM) models of each word for intra-word processing, and a Bigram grammar for inter-word processing. The result is the most likely (in a probabilistic sense) sequence of words.

Finally, the sequence of words is given to a parser which uses a stochastic context-free grammar to fill in slots of a predetermined template. The template has slots that are necessary for a data-base. Once the slots of the template have been filled in by the parser, the query is submitted and the request is processed.

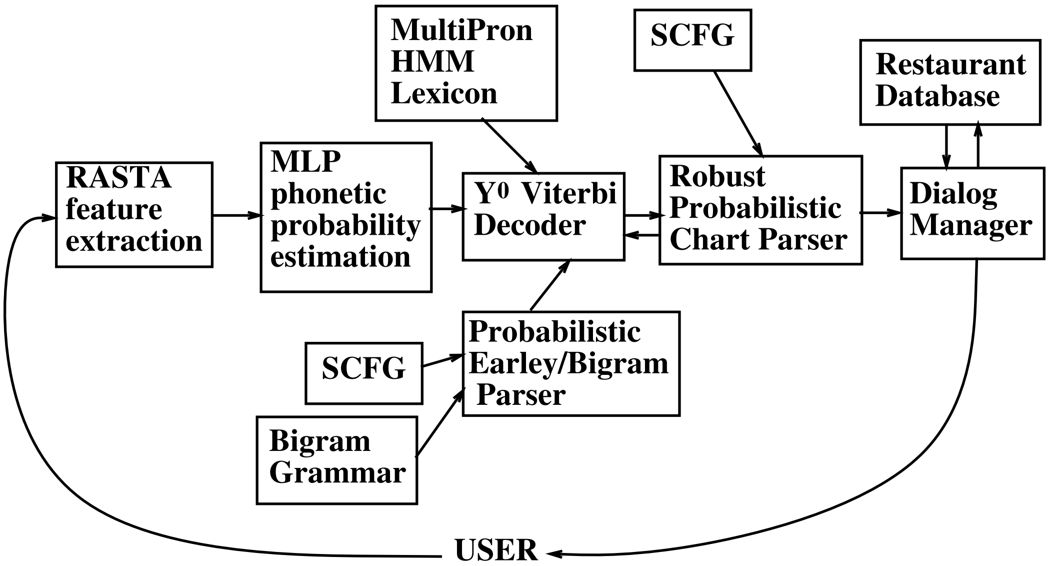

Figure 35.2: The BeRP Architecture

Figure 35.2 is a flow diagram of the Berp system in particular. The most notable changes between Figure 35.2 and Figure 35.1 is that Berp uses 1) RASTA feature extraction, 2) a Viterbi Decoder, and 3) a Dialog manager that manages the dialog between computer and user. The feedback loop involving the "USER" in Figure 35.2 is not feedback to enhance RASTA processing; this link depicts the way in which the dialog manager directs the user when to speak.

Figure 35.3 is a table that gives the size/type of data that BERP uses and the current error rates. As can be seen, there is still room for improvement.

35.2 Estimating Phonetic Probabilities with a MLP

Figure 35.4 shows the structure of the MLP used to estimate the phonetic probability distributions. Note that there are up to 61 output phones, a potentially very large hidden layer, and an input layer consisting of 9 frames where each frame consists of 20 RASTA or PLP features. This leads to a very large network and fast hardware or algorithms are necessary to make training or running such a network feasible. This type of

Training Corpus	4786 sentences + TIMIT	
Test Corpus	563 sentences	
Vocabulary	1274 words	
Data Base	1 database table, 150 restaurants	
Bigram	Perplexity 10.7 with 77% coverage	
Grammar	1389 handwritten SCFG rules	
Implementation	18,000 lines of C++	
Performance	Recognition	32.1% error
	Parsing	63% training 61% test
	Understanding	34% error

Figure 35.3: BeRP Status in August 1994

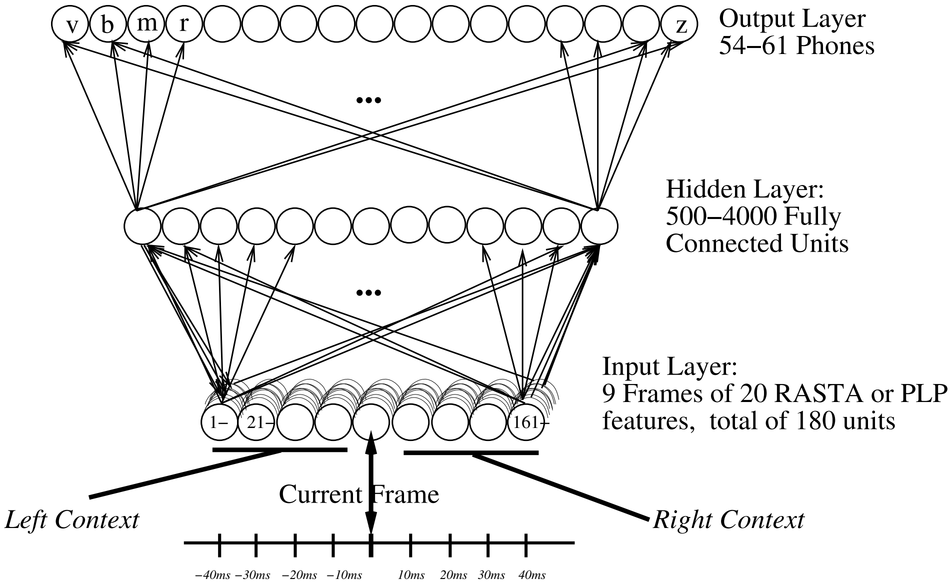

Figure 35.4: Estimating Phonetic Probabilities with a MLP

processing was described in class before, however, so we shall not delve into it any further.

35.3 The Lexicon: Word Pronunciations

BERP uses HMMs to model the intra-word phonetic probabilities. Figure 35.5 gives a simple model of the pronunciation of the word "of". We see that in this case, the HMM is a simple finite state automata, where there is one way to move from start state to end state.

Figure 35.6 shows a multiple pronunciation model of the word "of". In this case, each transition has an associated probability which corresponds to the "chance" that we will move from one state to another. For example, $P(q_b|q_a) = 0.7$ whereas $P(q_c|q_a) = 0.3$.

The goal of HMMs is to model the probabilities of a collection words given the acoustic input. HMM

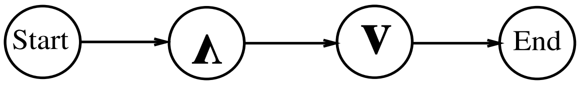

Figure 35.5: A simple model of the pronunciation of "of"

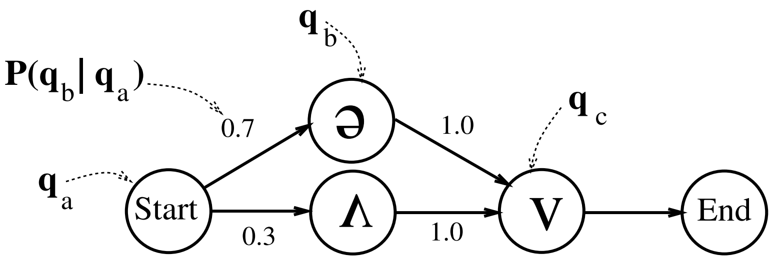

Figure 35.6: A multiple pronunciation model of "of"

modeling is similar to DTW in that we use one HMM model for word, and use some form of dynamic programming algorithm to infer the best word string. HMM modeling is different than DTW in that the paths are probabilistic and that they don't need to follow a fixed and predetermined set of path constraints. Again, we covered HMMs in class, so we will refrain from further description.

35.4 The Speech Recognition Problem

The speech recognition problem can be stated roughly as follows: *"What is the most likely string of words w in a language $\mathcal{L}$ given some acoustic input y"*, or in other words, we want to find w^* where

$$w^* = \underset{w \in \mathcal{L}}{\operatorname{argmax}} \; P(w|y)$$

Figure 35.7 gives an example, where the speech input is used to determine that w^* = *"If music be the food of love"*.

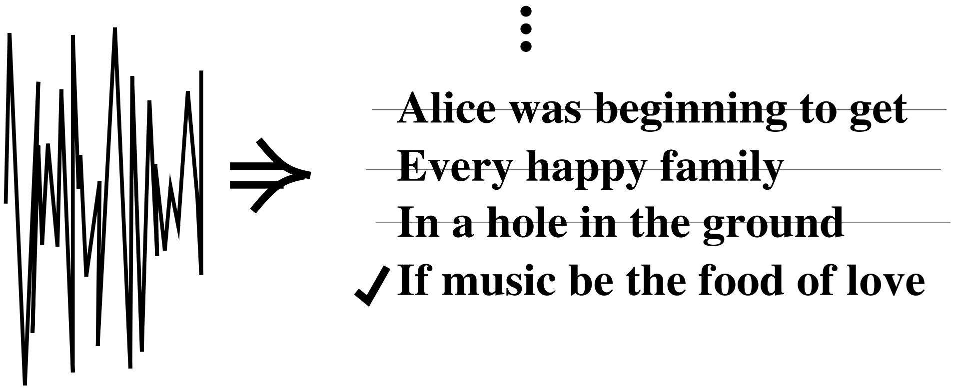

Figure 35.7: The Speech Recognition Problem

The problem can be restated as follows: given an acoustic string, return the most likely string of words. More formally:

Given:	y = string of acoustic events
Find:	The string of words 'w' which maximizes $P(w \mid y)$

Recall that Bayes rule says the following:

$$P(w \mid y) = \frac{P(y \mid w)P(w)}{P(y)} \tag{35.1}$$

Since P(y) constant relative to w, we can ignore it in a maximization problem. So we just need to find w^* where:

$$w^* = \underset{w \in \mathcal{L}}{\text{argmax}} \; P(y \mid w)P(w) \tag{35.2}$$

So, now that we have an idea of what the problem is, lets be a bit more specific about which module does what function (i.e., who does what). Remember, we need to find the word string which maximizes

$$P(y \mid w) \; P(w) \tag{35.3}$$

MLP (or Gaussians) computes $P(y_i \mid q)$ (The *state (phone or sub-phone) likelihood*). y_i indicates the i^{th} acoustic feature vector, and q is an HMM state that represents a particular phoneme (or other subword unit). As was said before, the MLP estimates the distribution of the phonemes for each feature vector. That means, the output of the MLP is actually $P(q \mid y_i)$. Bayes rule says that $P(y_i \mid q) = P(q \mid y_i)P(y_i)/P(q)$, so we approximate $P(y_i \mid q)$ by $P(q \mid y_i)/P(q)$ ignoring $P(y_i)$ for now. We get $P(q)$ by "doing a histogram or something."

pronunciation dictionary : For each word in our language, we'll need to know how to pronounce it. This means we need an HMM which tells us the probability of moving from one phone to another inside a word, or the intra-word phoneme probabilities $P(q_{i+1} \mid q_i)$. These are the *state ((sub)phone) transition probabilities* of each HMM for each word and indicate the probability of moving to state q_{i+1} given that we're in state q_i.

word likelihood algorithm : An algorithm for taking such an HMM for a word (i.e., all of the $P(y_i \mid q)$'s and $P(q_{i+1} \mid q_i)$'s) and gives the likelihood of part of the acoustic input given that word, i.e., $P(y \mid w)$, the *state (phone) string likelihood.*

language model : Since we're maximizing over all strings of the language, we need to know what it means to be a string of the language. The Language Model (LM) tells us the probability of these strings, i.e., it tells us P(w), the prior probability of word string w.

search algorithm/decoding : An efficient (possibly suboptimal) algorithm for intelligently searching through all possible strings of words in the language, given the probabilities $P(y \mid w)$ and $P(w)$. That is, combine $P(y \mid w)$ and $P(w)$ to find optimal word string w^*.

Why is decoding hard? Consider the following phone sequence.

wɛðərʔɪznoblərɾəsʌfərðəslɪŋzændærozv

In other words, *whether tis nobler to suffer the slings and arrows of.* There are several reasons why converting this phone string into an optimal list of words is difficult.

dinner	0.122	some	0.089	lunch	0.068
a	0.061	on	0.059	breakfast	0.041
today	0.039	Italian	0.037	any	0.026
Indian	0.024	Chinese	0.023	for	0.020
in	0.018	American	0.017	Mexican	0.016
Thai	0.013	pizza	0.012	vegetarian	0.01
this	0.012	tonight	0.011	Saturdays	0.011

Figure 35.8: The Language Model: Bigram for 'eat'

- No word-boundaries, we don't know the number of words.
- Coarticulation effects where the pronunciation of phrases or sub-word units can be different depending on context.
- The string of phones is not error-free.
- The string of phones is not a string of phones but a list of probability mass functions over phone sets.

35.5 The Language Model: N-gram

The language model provides us with inter-word transition probabilities. In general, we represent the probability of a string of words as follows:

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)\ldots P(w_n \mid w_1^{n-1}) \\ &= \prod_{k=1}^{n} P(w_k \mid w_1^{k-1}) \end{aligned} \tag{35.4}$$

where $P(w_1^n)$ is the probability of w_1 followed by w_2 followed by ...followed by w_n (note that this result follows by a multiple applications of the definition of conditional probability). The N-gram language model assumption says that we only consider probabilities of sequences of N words. That is we assume,

$$P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-N+1}^{k-1}) \tag{35.5}$$

. For $N = 2$ (a Bigram and therefore a Markovian dependency between words), we get:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1}) \tag{35.6}$$

.

Figure 35.8 gives an example of a bigram language model for the word "eat". Each number in the table gives the probability of the corresponding word following the word "eat". So the most likely word to follow "eat" is "dinner".

35.6 Word Probabilities

But how do we compute the word likelihoods themselves? That is, we need to solve the following problem:

Given: y = string of acoustic events x_i
w_i = one simple HMM word model:
Find: $P(y \mid w_i)$

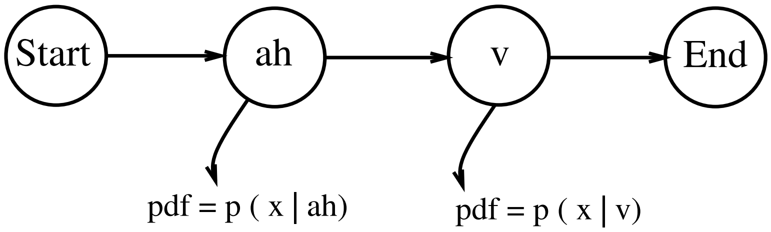

Figure 35.9: Simple HMM word model

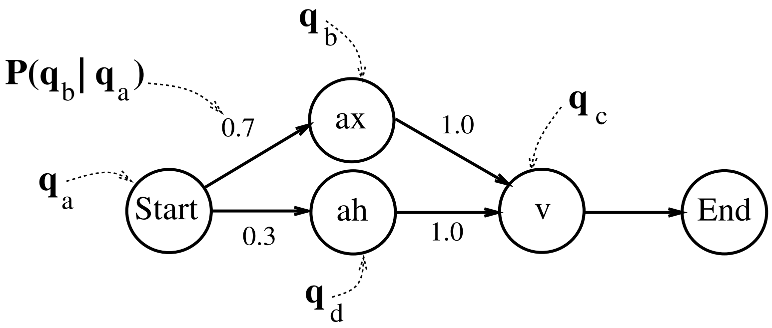

Figure 35.10: More complex HMM word model

Figure 35.9 shows a simple HMM word model. In this model, there are no alternate paths between states, so all the path transition probabilities are unity. For a particular word, the probability is simply the product of the probabilities of seeing the particular phonemes in each state. That is, we get: [1]

$$P(y \mid w_i) = P(x_0 \mid ah)P(x_1 \mid v) \tag{35.7}$$

Figure 35.10 shows a more complex HMM word model. In this case, we need to consider the alternate paths in the model in the computation of the total probability. Assuming that the phone string consists of phones $y = [x_0 x_1]$, then $P(y \mid w_i)$ is given by:

$$\begin{aligned} P(y \mid w_i) = \; & P(q_b \mid q_a)P(x_0 \mid q_b)P(q_c \mid q_b)P(x_1 \mid q_c) \\ & + P(q_d \mid q_a)P(x_0 \mid q_d)P(q_c \mid q_d)P(x_1 \mid q_c) \end{aligned} \tag{35.8}$$

That is, we have two paths. The first path moves from state q_a to q_b with probability $P(q_b \mid q_a)$. The probability of seeing x_0 if we're in state q_b is $P(x_0 \mid q_b)$. Then there is a transition $P(q_c \mid q_b)$ (which in this case is unity because there is only one transition), and a final probability if seeing x_1 given that we're in q_c. The lower path is similar, and the total probability is the sum of each of these.

35.7 Given a word string w, Computing $P(y|w)$

There are several ways of doing this. A first approach might be to search the space of all possible sequences of states as listed in Figure 35.11. This is not done, however, because of the intractability of the algorithm (i.e., it's $O(N^T)$).

[1]Note that in this lecture, we use x_i and y_i, i.e., a particular acoustic feature vector, interchangeably

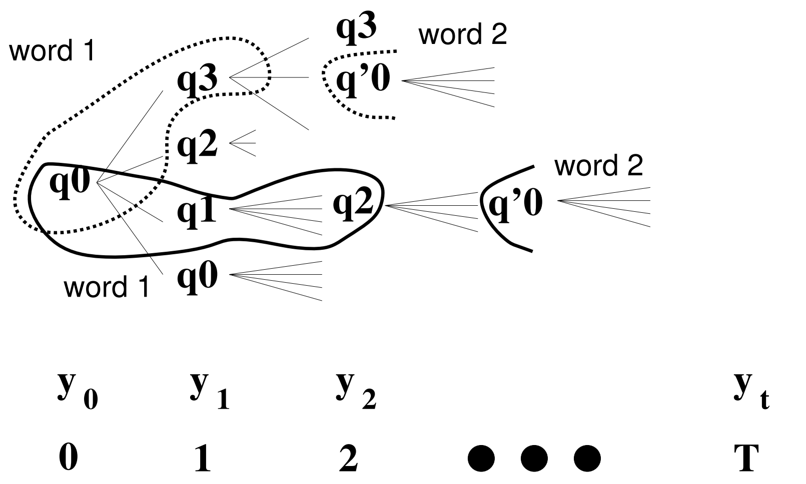

Figure 35.11: Searching the space of all possible sequences of states

Instead, we do the *The Forward Algorithm* which is $O(N^2T)$ [2]. The idea is as follows:

- Fold these exponential paths into a simple trellis, so that all possible paths will merge into N states at every time slice.
- We define the *forward probability* as follows: $\alpha_t(i) = P(y_0 y_1 \cdots y_t, q_t = S_i|w)$ which is the probability, given word string w, of acoustics y_0 through y_t and that the t^{th} state is S_i.
- We compute it by induction:
  - Initialization: $\alpha_1(i) = \pi_i P(y_1|q_i), 1 \leq i \leq N$ where π_i is the initial distribution over states, i.e., $\pi_i = P(q_i)$.
  - Induction:

$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) P(q_j|q_i)] P(y_{t+1}|q_j), \quad 1 \leq t \leq T-1, 1 \leq j \leq N \tag{35.9}$$

  - Termination: $P(y|w) = \sum_{i=1}^{N} \alpha_T(i)$ where T is the number of acoustic frames.

For example, consider Figure 35.12. Here, the computation of the forward variable $\alpha_t(i)$ in terms of a trellis of observations t and states i is done by summing, i.e.,:

$$\alpha_1(3) = \sum_{i=1}^{4} \alpha_0(i) P(q_3|q_i) P(x_1|q_3)$$

Note that the forward algorithm efficiently computes $P(y \mid w)$. But we would have to run the forward algorithm for each sentence w in the language, and for any reasonably sized language this cost is prohibitive. There are two approximate solutions to this problems.

[2] The Forward algorithm described here is just the forward part of the more commonly known Forward/Backward algorithm which computes the probability of an observation sequence given an HMM model

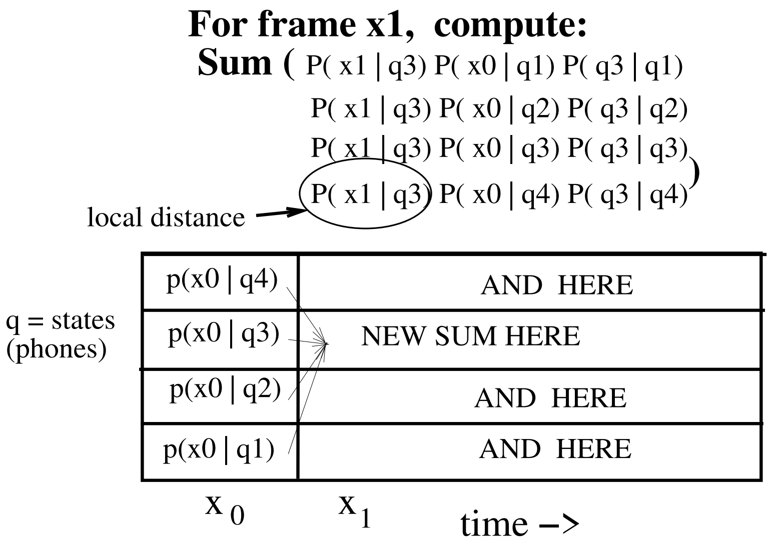

Figure 35.12: Forward Algorithm Example

Stack Decoding : Use smart graph search algorithms so that we only run the forward algorithm for sentences w which look likely to match the acoustic input y. We won't go into this today.

Viterbi Decoding : Use a dynamic programming algorithm called the Viterbi algorithm, a modification of the forward algorithm, which *simultaneously* computes $P(y|w)$ and finds the w (best path) which more-or-less maximizes $P(y|w)$.

In the Viterbi algorithm, we approximate $P(y \mid w_i)$ by the best path. Going back to our "more" complex HMM model (see Figure 35.10), we get:

$$P(y \mid w_i) = P(q_b \mid q_a)P(x_0 \mid q_b)P(q_c \mid q_b)P(x_1 \mid q_c) \tag{35.10}$$

assuming that this indeed is the maximum path into q_c. So in this algorithm, at each time-slice, we take the MAX instead of the SUM.

For example, consider Figure 35.13. Here, we see that

$$\alpha_1(3) = \max_i \alpha_0(i)P(q_3|q_i)P(x_1|q_3)$$

But how do we handle the transitions between words? That is, when we get to the end of a HMM word model, and need to use the word transition probabilities provided by the language model, what do we do? In fact, we use the same equations but the language model, whether Bigrams or stochastic context free grammar, provides the transition probability rather than using a HMM state transition probability. For example, in Figure 35.14, we see the transition from word 1 to word 2. In this case, we are still under the Viterbi assumption, so we compute the maximum of the previous frame, but in the word transition case, we use $P(w_2|w_1)$ as the transition probability. Obtaining the word transition probabilities from a bigram language model is obvious. Obtaining the word transition probabilities from a stochastic context-free grammar is done by top-down parsing the string of words seen so far according to the grammar, and computing the probability of seeing each next word.

Figure 35.15 shows a more global picture of the best path through this frame space. As can be seen, at the end of word one, the best path involves a word transition (using the word transition probabilities), so the path continues at word W.

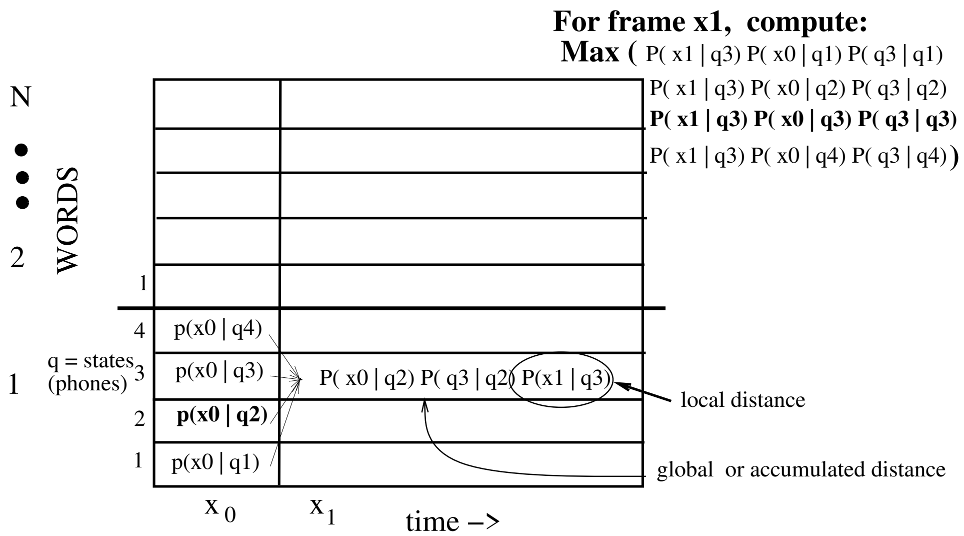

Figure 35.13: The Viterbi Algorithm: Example

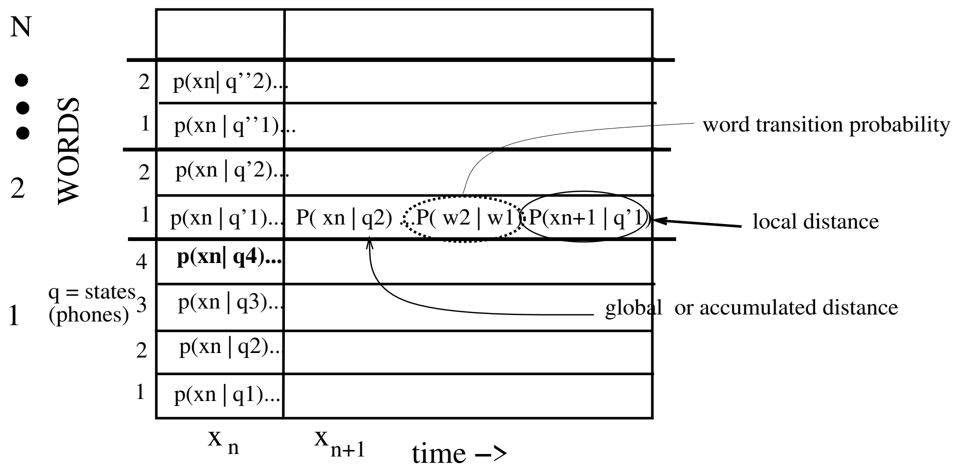

Figure 35.14: Viterbi Algorithm: Intra-word transition probability. example

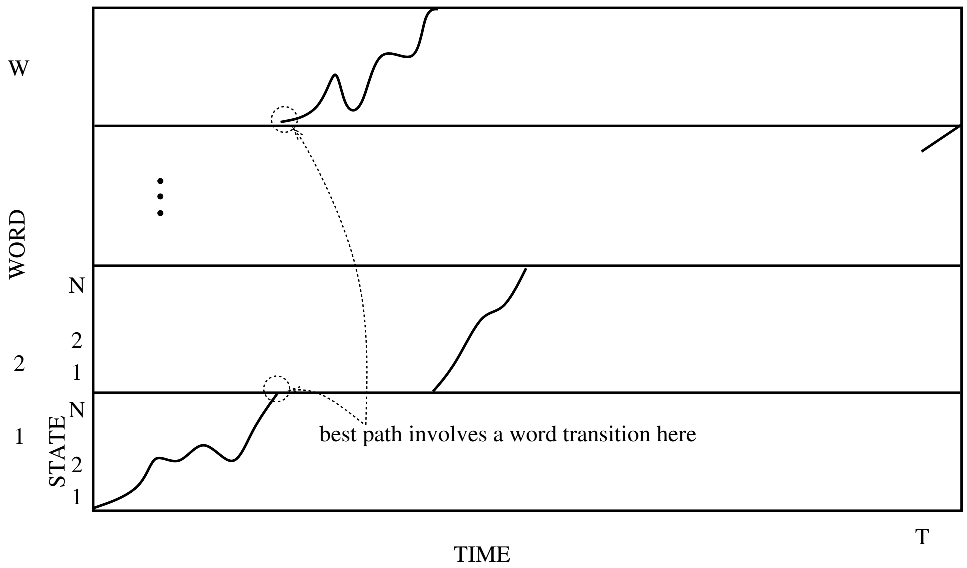

Figure 35.15: The Viterbi Algorithm III: A view of the search

Like in DTW, we are not really interested in computing the optimal path through phonemes, but rather the optimal list of words. Therefore, we need not compute the entire large array, but instead can use arrays to keep a back-trace of the most likely path at each time frame. We will not discuss it now, but in the same way as DTW, we can compute the optimal word string once we reach the final column. This is illustrated in Figure 35.16.

So, the efficiency optimizations we can do for the viterbi algorithm include:

- Storage: We just keep a column array of accumulated distances. This means we'll need to keep back-pointers for each state as described above.
- Simplifying assumptions: We make HMM transitions only to the same or higher-numbered states.
- Pruning at the state level.
- Pruning at the word level.

35.8 The Natural Language Back-end

As was stated before, the language model is is used to compute the inter-word probabilities in the decoding phase. When we use a stochastic context free grammar as the language mode, we in fact use the same grammar in the natural language back-end.

The goal of the NL back-end is to obtain information from the collection of sentences and fill in slots or fields of a template. This is done by having the user interact with a *dialog manager*, Each sentence hopefully contains information useful to this task. The template contains fields that are pertinent to making a restaurant query, such as food type, cost, distance, time, etc. Once the NL back-end has filled in all the template slots, it is submitted to the data-base manager which performs the query.

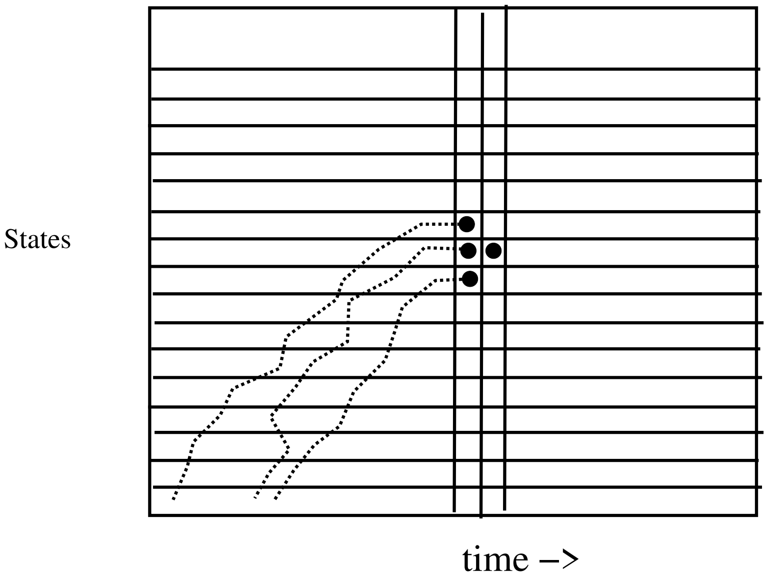

Figure 35.16: The Viterbi Algorithm IIIb: Back-tracing. When advancing to next column, we extend the back-trace pointer to most likely path.

Figure 35.17 shows the structure of the NL back-end. The output from the decoder, a complete string of words, is given to the probabilistic bottom-up chart parser. The grammar is a stochastic context free grammar, a normal context free grammar with probabilities associated with each production rule (see Figure 35.18). The rules themselves are handwritten and the rule probabilities are determined by training on a corpus of sentences (so the probabilities are determined experimentally).

Note that the string of words may be ambiguous relative to the grammar (i.e., have more than one parse) or might not have a parse at all. If there is ambiguity, we choose the parse with the highest probability (this is the reason for using a stochastic grammar). If there is no parse, then parse fragments (a partition of the word string with a best possible parse of each segment) are produced.

The Greedy Parse-Fragment Combination module tries to infer information from the parse (or parsed fragments). That is, just because a sentence doesn't have a parse according to a context free grammar doesn't mean that the sentence isn't rich with semantics. This module, therefore, attempts to combine the parse fragments in such a way that the semantics of the entire sentence can be inferred.

The Context Module fills, as its name implies, fills in contextual information. That is, when the sentences contains words such as "today", or "the fourth one", the context module determines today's date, or the "what" that "the fourth one" refers to.

Similar to a computer language compiler, the three preceding modules work by creating and modifying an annotated parse tree. That is, a parse tree with different slots in each node enables the overall system to infer some "meaning" from the original list of words. In this context, the parsing is called probabilistic chart parsing, since the parsing can be though of as occuring on a chart as in Figure 35.19. We can see the initial word string is "i would like to eat american food or mexican food". The chart shows how the relevant information (in this case nationality) is obtained from the sentence, and ultimately used to determine the restaurant type.

The probabilistic chart parse can also be used to infer partial information from an incomplete sentence. Figure 35.20 shows that the query "I want British" can in fact considerably narrow down the selection of

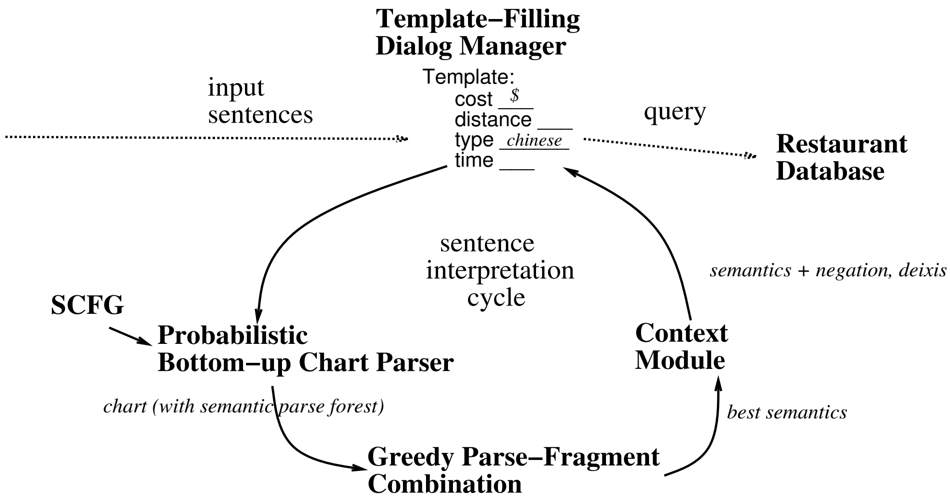

Figure 35.17: The Natural Language Backend

- 1389 hand-written rules
- rules trained on 4786-sentence corpus using EM
- rule sample:

[.35]	s $\rightarrow$ WANTING-S
[.32]	WANTING-S $\rightarrow$ i WANT-V WANT-OBJ
[.97]	WANT-V $\rightarrow$ WANT
[.54]	WANT-OBJ $\rightarrow$ NATIONALITY FOOD-RESTAURANT
[0.6]	FOOD-RESTAURANT $\rightarrow$ FOOD
[0.4]	FOOD-RESTAURANT $\rightarrow$ RESTAURANTS
[0.06]	RESTAURANTS $\rightarrow$ places
[0.1]	RESTAURANTS $\rightarrow$ restaurants
[0.04]	FOOD $\rightarrow$ cuisine
[0.8]	FOOD $\rightarrow$ food

Figure 35.18: Stochastic Context-Free Grammar

possible restaurants for the subsequent database query.

References

[BRP94] Daniel Jurafsky, Chuck Wooters, Gary Tajchman, Jonathan Segal, Andreas Stolcke, Eric Fosler, and Nelson Morgan, (1994) "The Berkeley Restaurant Project" ICSLP94 Yokohama, Japan, pp. 2139-2142.

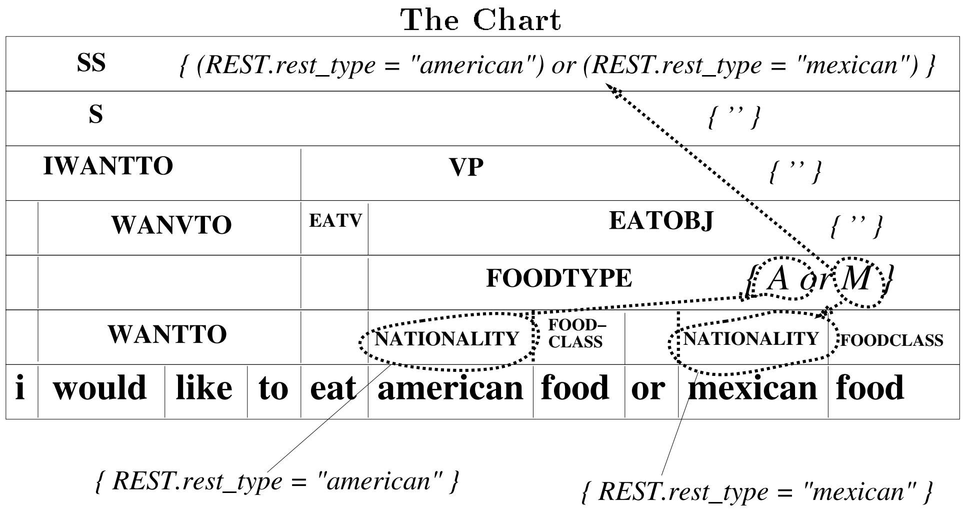

Figure 35.19: The Chart

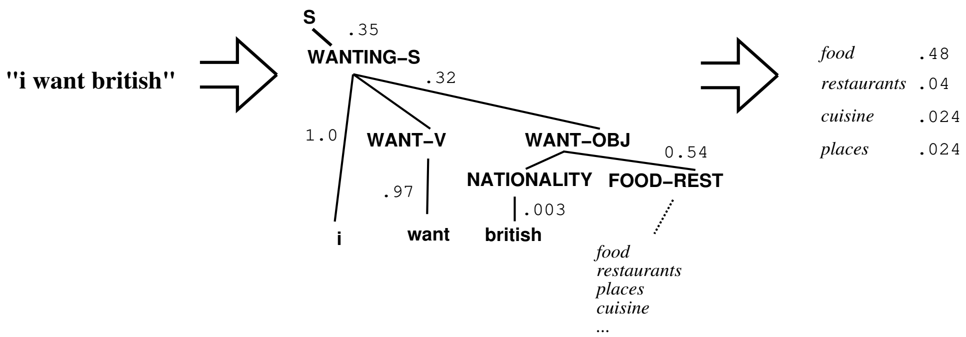

Figure 35.20: Probabilistic Chart Parser