A STACK DECODER FOR CONTINOUS SPEECH RECOGNITION[1]

Dean G. Sturtevant
Dragon Systems, Inc.
90 Bridge St.
Newton, MA 02158

ABSTRACT

We describe the structure, preliminary implementation and performance of an algorithm for doing continuous speech recognition. The algorithm, known as a stack decoder, proceeds by continually evaluating one-word extensions of the most promising partial transcriptions of an input utterance. The output is a list of candidate complete transcriptions, ordered by likelihood under a stochastic model. The stochastic model in the current implementation is composed solely of an acoustic component - a linguistic component will soon be added. The acoustic models make use of dictionary phonetic spellings together with models for phonemes in context. The linguistic models will be based on digram statistics.

A key component of the system is a module for quickly evaluating a hypothesized partial transcription of an input utterance to determine how likely it is that it will extend to a complete transcription which is the most likely transcription under the model.

THE SPEECH RECOGNITION PROBLEM

Natural language automatic speech recognition typically proceeds as follows. Human speech is recorded via a microphone, then digitized. This digitized waveform is further processed to extract time- and/or frequency-domain parameters and features. This processed input, which we will refer to as an 'utterance', is then fed to a recognizer, which is a program that uses knowledge about speech and language to present a list of possible transcriptions of the input.

We will discuss a particular algorithm, known as a 'stack decoder' (or sometimes, 'A*-search') for doing continuous speech recognition. We will also discuss an implementation of this algorithm developed at Dragon Systems.

THE STACK DECODER

This section provides a description of the basic structure of the algorithm. The development of the stack decoder idea as applied to speech recognition was first done by Fred Jelinek and his associates at IBM in the early 70's (Jelinek et. al., 1975 [3]), based on earlier work by Jelinek (Jelinek, 1969 [1,2]), who had developed the algorithm as a method of sequential decoding of transmitted information.

The algorithm controls the use of two sub-algorithms, which in this section we will assume as given. Each of the sub-algorithms takes as input a word sequence W and an utterance U. The first, which we will call a 'complete transcription scorer', or CTS, computes the likelihood that W is a complete and correct transcription of the utterance U. The second sub-algorithm, a 'partial transcription evaluator', or PTE, computes a 'priority' for a hypothesized partial transcription. This priority will be used to decide which of a given list of partial transcriptions will be considered first for extension.

---

1. This work was sponsored by the Defense Advanced Research Projects Agency and was monitored by the Space and Naval Warfare Systems Command under contract N000-39-86-C-0307

The 'stack' (which is far from being a stack in the computer science sense of a first-in-first-out list) is a list of partial transcriptions, ordered by the PTE. Initially the stack consists of the empty word sequence (which is certain to extend to a correct transcription of the utterance). The algorithm then iterates the following steps until a stopping criterion, explained below, is satisfied.

- 1. Remove from the stack the partial transcription W having the highest priority.
- 2. For each possible one-word extension W' of W:
    * 3. Apply the PTE to W' and use the result to insert W' in the appropriate place on the stack.
    * 4. Apply the CTS to W' and use the result to insert W' in a 'choice list'.

The stopping criterion will depend on the desired output and performance. If the desired output is a list of the N most likely candidates for the transcription, then run the algorithm until it may be determined that the best partial transcription on the stack cannot extend to a complete transcription which scores better than the Nth best choice so far computed.

REFINEMENTS OF THE BASIC ALGORITHM

One obvious refinement is to save the computational state of an item on the stack. That is, one expects that the computation that the PTE and CTS perform with input W', a one-word extension of W, can make use of the work done in computing with input W. In particular, suppose that the utterance U consists of a sequence of parameter vectors, one vector representing the speech signal at a certain discrete time instant. Then one could save, over a certain time interval, the likelihood that W provides a correct partial transcription of U up to each time in that interval. In other words, save an ending time distribution for the partial transcription W.

Another refinement is called 'thresholding'. This is the process whereby a hypothesized partial transcription is discarded when there is good reason to believe that it will not extend to a choice in the top N. If thresholding is done, there is no guarantee that the most likely complete transcription will appear on the choice list. However, thresholding significantly reduces the amount of computation required. One method of thresholding is to maintain a likelihood score for the best current hypothesis at each time instant, and to discard a partial hypothesis if its score at a particular time is more than a fixed difference from the score of the best hypothesis at that time. Another kind of thresholding results from placing a limit on the number of items on the stack at any one time (such a limit already exists by virtue of memory limitations). When a new item is inserted into a full stack, the item having least priority will be discarded in order to make room.

A third refinement is called 'shared computation'. The acoustic models corresponding to different one-word extensions of a partial transcription may be identical on initial portions. One may compute the acoustic match on those identical portions just once. This may be accomplished by changing the algorithmic assumptions slightly, and allowing the PTE and CTS to operate on all legal one-word extensions of a given partial transcription. A further advantage of doing this is that thresholding may be done sooner if the extensions are processed time-synchronously.

'Rapid match' is the process of pruning the one-word extensions of a partial transcription to produce a shorter list. Rapid match would make use of both acoustic and linguistic information in a crude manner to discard possibilities that are not likely to extend to a correct complete transcription. One method of doing this is to combine simple word frequencies and acoustic matches to crude models for word beginnings to obtain a smaller list of candidates.

ACOUSTIC MODELS

With this section we start discussion of the implementation details of the stack decoder at Dragon Systems. The models we use for words are based on phonetic spellings (from the Random House Unabridged Dictionary [4]) together with stochastic models for phonemes in context.

A phoneme in context model (or PIC) models the acoustics and duration of a phoneme in a given environment which consists of:

- 1. The immediately preceding phoneme.
- 2. The immediately succeeding phoneme.
- 3. The level of stress (primary, secondary, or unstressed).
- 4. Whether or not the phoneme is expected to undergo pre-pausal lengthening.

The fourth component of the environment may need further explanation. Before a pause, a speaker will typically extend speech sounds past their normal durations, and this pre-pausal lengthening is confined mainly (although not strictly) to the syllable immediately preceding the pause.

A PIC is a stochastic network consisting of a sequence of from one to three nodes. Each node carries with it a probability distribution of acoustic parameters and a probability distribution of durations.

For purposes of efficiency in storage, we allow the same PIC to represent the phoneme in many different environments.

A stochastic model for a word is constructed "on the fly" during the recognition process using its phonetic spelling. The PICs that may represent the phonemes in a word (with a given phoneme given as left context) are assembled into a network in the following way.

- 1. Each phoneme that is not included in the final syllable of the word has only one possible PIC representing it.
- 2. The final syllable will have one of two subnetworks representing it (unless the final syllable consists of one phoneme - in that case there will be more, as indicated in 3.). One subnetwork (the 'pre-pausal subnetwork', or PPS) would consider the hypothesis that a pause immediately follows the word, the other subnetwork ('non-pausal subnetwork', or NPS) would consider the opposite hypothesis.
- 3. In the NPS, the final phoneme will be represented by any one of a number of PICs, depending on which phoneme starts the next word in the hypothesis. For the PPS, the final phoneme is assumed to be followed by silence.

LANGUAGE MODELS

At the time of preparation of this paper, a language module was being installed in the system. This language module has been tested in conjunction with Dragon Systems' discrete utterance recognizer.

The language module makes use of word-pair statistics (for common word pairs) combined with one-gram statistics.

THE COMPLETE TRANSCRIPTION SCORER

When we refer to a 'score', we mean a negative log likelihood of a path through a stochastic network. Thus the most likely paths are those with the lowest scores.

Recall that a complete transcription scorer (or CTS) computes an estimate of the likelihood that a given word sequence is a correct transcription of the input utterance. The CTS for the stack decoder at present consists of a single component, the acoustic likelihood module (ALM) (future versions will include a linguistic likelihood module and perhaps more). The output of the ALM is based upon consideration of the following stochastic process. Link together the stochastic networks defined by the words in the proposed transcription in a linear fashion. The right and left contexts for each word are now determined, so the resulting network is a linear sequence of nodes. For each node in this sequence, obtain a sample parameter vector sequence from its distribution. Concatenate these sequences together to produce an utterance. The ALM is a dynamic program to compute the likelihood that this process produces the input utterance.

THE PARTIAL TRANSCRIPTION EVALUATOR

The general description given in the section entitled "THE STACK DECODER" defined a partial transcription evaluator, or PTE, as a module which takes as input an utterance and a hypothesized partial transcription and returns a priority for evaluating extensions of the partial transcription. It is a challenging research problem to design algorithms that compute an effective measure of priority.

An effective priority should satisfy the following requirements.

- 1. There should be an algorithm (i.e., a PTE) which computes it fairly rapidly.
- 2. It should tend to favor extending correct partial hypotheses.

If the PTE computes an estimate of the likelihood of the best complete transcription extending the input partial transcription, then that would satisfy requirement 2. The PTEs that we have developed and tested have this goal in mind.

The importance of the actual implementation of the PTE is two-fold. First of all, the earlier the best scoring transcription is considered, the better the subsequent thresholding will be. Second, a poor PTE may cause the best partial transcription to 'fall off the bottom of the stack'.

So far we have tested several PTEs. The first few of these do not look ahead at the speech data. However, they do (as they should) allow comparison between hypotheses which consume differing amounts of speech data. Following are the descriptions of the PTEs.

The Average Score PTE. The measure returned by this PTE is the average score per time instant for the hypothesis as computed by the ALM.

The Score Difference PTE. The number returned by this PTE is the difference between the score for the hypothesis and the best score of any path at the time which is the best guess for the ending time of the last word in the hypothesis.

The Confidence PTE. As explained above, each node in the stochastic network has associated with it some probability distributions. Hence there is a notion of 'expected score' for a path in the network (averaged over utterances created by drawing samples from these distributions). The 'confidence' of a path through the network obtained by running the acoustic likelihood module on an input utterance is the difference between the expected score for that path and the actual score as computed by the module, divided by a factor designed to reduce the effect of larger variance for longer hypotheses. This confidence is the measure returned by the confidence PTE. It is essentially a 'normalized' total score to allow comparison between paths of different lengths in the network.

MISCELLANEOUS IMPLEMENTATION DETAILS

The code implementing the stack decoder described above has been written using the C programming language, conforming completely to the ANSI standard.

The implementation also was designed with parallelism in mind. In particular, given a parallel programming environment which supports C, the code may be modified so as to allow different partial transcriptions to be extended simultaneously on separate processors.

A separate PC-specific module allows one to view labelled spectrograms of both recorded and fabricated data. More specifically, one can save live utterances (as above - live speech data which has undergone signal processing) or create utterances from the models (string together data based on the means of the probability distributions in the nodes). Using either one of these types of utterances as input, together with acoustic models and a transcription of the utterance, the spectrogram will display (using CGA, EGA, or VGA) a view of the utterance, segmented according to the model (i.e., the module finds the best path through the stochastic network defined by the the model corresponding to the given transcription). The module may be used for (among other things) understanding speech recognition errors and discovering programming errors.

PERFORMANCE AND TESTBED

As the stack decoder is still in the intermediate stages of development, significant performance results are not yet available. We will in this section describe the preliminary results with a caveat that these results should not be taken as any indication of the future performance of the stack decoder, but rather as an indication that the acoustic models that we have described model coarticulation reasonably well, and that they have a chance of performing well on more difficult tasks.

The first test of the stack decoder on speech data used a vocabulary consisting of the ten digits. The models for phonemes in context were trained by one speaker. Two lists of 100 7-digit sequences were constructed with the property that all digits appeared with every possible left and right context in each list. A second speaker provided utterances for every sequence on these lists. Using the 100 utterances from the first list, the PICs were adapted to model the second speaker's voice. The 100 utterances from the second list were used as test data. Two replacement errors resulted.

A second test was based on 100 frequent words appearing in a large (several million words) corpus of radiology reports. Sentences were constructed from these words by considering the most common 8-grams in the corpus. Models for phonemes in context were built, and utterances of the sentences were collected from the speaker who provided the training data for the PICs. So far the stack decoder has not performed well - on a sample of 10 of the sentences, in five instances the correct transcription does not appear on the choice list, even though when the CTE was applied to the thresholded transcriptions, the scores would have placed them at the top or high in the list. In the other five, the correct transcription appears first three times, second once, and third once. Over the ten sentences, there was one insertion error, seven replacement errors, and two deletion errors out of the 85 words in the sentences. We hope that installation of the language model, together with implementation of a superior PTE, will cause the thresholding problem to diminish.

In the future, we will use as the testbed for our continuous speech recognition algorithms a 1000-word vocabulary and language model based on the radiology corpus.

FUTURE RESEARCH

In this section, we sketch some ideas which, when implemented, are expected to improve the performance of the stack decoder.

Installation of rapid match and language model. These imminent developments (described briefly above) are expected to boost the time and accuracy performance of the decoder significantly.

Improvements in the partial transcription evaluator. In the section entitled "THE PARTIAL TRANSCRIPTION EVALUATOR" we have outlined PTE calculations that do not look ahead at speech data not 'consumed' by the hypothesized partial transcription. We are in the process of developing algorithms that do look ahead, and expect them (on a theoretical basis) to be more effective than the ones we have implemented so far.

The results on the 100-word task indicate that in the top choices, the longer words (which are typically content words) are generally recognized correctly, whereas uncertainty as to the correct shorter words (usually function words) exists. This suggests the following two improvements.

Caching of word matches (terminology suggested by Doug Paul). In the present scheme, the scorers are repeatedly computing likelihoods of a word in the same time interval. Caching would save the results of such evaluations for future use.

Special treatment of function words. One possibility is to treat the set of all function words (or even the set of all reasonable function word sequences) as a single model in the initial stages of recognition. As a post-processing step, the best choice for the function words will be determined. There are also certain benefits in this treatment as far as the language model is concerned, and research is currently being done on this aspect.

ACKNOWLEDGEMENTS

The author would like to thank Jim Baker for describing the structure of the stack decoder to him and for contributing many ideas about its implementation. The work on phonemes in context is due to Paul Bamberg. Larry Gillick and Bob Roth contributed the work on rapid match and language modelling.

REFERENCES

1. F. Jelinek, "A fast sequential decoding algorithm using a stack," IBM J. Res. Develop., vol. 13, pp. 675-685, Nov. 1969.

2. F. Jelinek, "A stack algorithm for faster sequential decoding of transmitted information," IBM Res. Center, Yorktown Hts.,N.Y.,
IBM Res. Rep. RC2441, Apr. 1969.

3. F. Jelinek, L.R. Bahl, R.L. Mercer, "Design of a Linguistic Statistical Decoder for the Recognition of Continuous Speech," IEEE Transactions on Info. Theory, Vol. IT-21, No. 3, pp. 250-256, May 1975.

4. The Random House Dictionary of the English Language, Unabridged Edition. Random House, Inc., 1983.