Continuous Speech Recognition Using Modified Stack Decoding Algorithm

David C. Lee

Dept. of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
dlee1@ece.cmu.edu

Introduction

This paper attempts to recognize a speech sequence with a series of words. We represent words with HMM models and use a modified version of stack decoding method [1][2] as the primary algorithm to recognize words. This type of method allows us to recognize a streaming sequence of speech signal as we receive it. We introduced a heuristic function that works with stack decoding algorithm. We also observed how model parameters affect the performance of the algorithm.

Word Model

We have used the Bakis model [3] to represent words. The Bakis model is one type of a HMM model that nicely models the time-varying characteristic of a speech signal while it keeps the size of the search space low. Bakis model only allows the state sequence to go in one direction, i.e., the sequence can not move in backward direction. This one directional characteristic is why it nicely models the time-varying nature of a speech signal. Let the transition probability of moving from state i to state j be a_{ij}. Then,

$$a_{ij} = 0, \ j < i$$

We could also add limitations on the model to prevent it from jumping over too many states, i.e.,

$$a_{ij} = 0, \ j > i + \Delta$$

The output of the model is a vector. Each state of the model is assigned a probability function of the output vector. If the output alphabet is discrete we could assign probabilities to each of the alphabets, but if the output is chosen from a range of continuous real numbers, we must assign a probability density function to each state. The most commonly used pdf is the Gaussian distribution

Each word model has one starting state and one ending state. Word to word transition probability is also assigned.

Recognition Algorithm

There are several algorithms that can be used in recognition, such as Viterbi decoding, beam search, token passing, frame synchronous network search and stack decoding search. We have used stack decoding algorithm with some modification as our search method. Stack decoding search is one form of a heuristic A* search. This search method is useful when we have a nice working heuristic function, but finding one is not easy.

Let the input sequence be a series of vectors $\mathbf{x}_1\mathbf{x}_2\cdots\mathbf{x}_n$ and let there be a number of states for each word model. There are also several words in the entire vocabulary. Calculating the cost will be discussed in detail in a while. The recognition process works as follows:

1. Take the first input $\mathbf{x}_1$ and calculate the cost of $\mathbf{x}_1$ being produced from our word model for all the possible combinations of getting $\mathbf{x}_1$ as an output. Then, add the $<\mathbf{x}_1, s, cost(s)>$ tuple to the stack, where s indicates which state of which word the corresponding cost was calculated from.
2. Sort the stack in decreasing order with respect to the last entry of the tuple, i.e., the cost.
3. Pop the first entry in stack, which should be the one with the lowest cost. Let the popped entry be $<\mathbf{x}_1\mathbf{x}_2\cdots\mathbf{x}_p, s_{pop}, cost(s_{pop})>$.
4. If the last state of s_{pop} is a final state of a word, conclude that $\mathbf{x}_1\mathbf{x}_2\cdots\mathbf{x}_p$ is an utterance of that word. Empty the stack. If there are remaining elements in the input sequence, go to step 1 and repeat the process beginning with element $\mathbf{x}_{p+1}$.
5. Find all the possible combinations of $\mathbf{x}_{p+1}$ being produced, given that $\mathbf{x}_1\mathbf{x}_2\cdots\mathbf{x}_p$ was produced from s_{pop}. Add all combinations of $<\mathbf{x}_1\mathbf{x}_2\cdots\mathbf{x}_p\mathbf{x}_{p+1}, s_{pop}s_{pop+1}, cost(s_{pop}s_{pop+1})>$ to stack. Note that $\mathbf{x}_{p+1}$ and $\mathbf{x}_1\mathbf{x}_2\cdots\mathbf{x}_p$ could only have been produced from the same word, since we can assume that only after one finishes a word can he begin the next word.
6. Go to step 2. Repeat this until the end of the input is reached.

Notice that during this process, we do not have to know

the whole input sequence in advance to conclude that a certain part of the input sequence corresponds to certain words.

Cost Function

The cost function takes a sequence of states as the input and gives the cost of that sequence as the output. Since the sequence of states given as input may only be the beginning part of the word, the cost consists of two parts: the cost of the beginning part of the state sequence matching the model and the estimated cost that this state sequence will have when this sequence reaches the end of the word. The latter is where the heuristic function comes into play. Since we should determine the cost of certain sequence of states without knowing what it is, we have no choice but to guess. Fortunately, for the algorithm given above, there is a way of guessing that guarantees that we will not get a wrong answer. This is guaranteed when the heuristic function is admissible. A heuristic is admissible if guessed cost does not exceed the actual cost of the correct answer. As long as this condition is satisfied, it is preferred to have a higher estimated cost, as this will reduce the size of the search space. One extreme case of a choice of heuristic will be to guess that cost is 0. This will give us a correct answer but the search space will be huge.

In our experiment, we have chosen the heuristic to be the minimum cost to reach the goal state, as this will guarantee an admissible heuristic while keeping the search space down to a reasonable amount.

There is one more part to the cost function that has not been mentioned yet. Since cost function is simply the negative log of the probability, the shorter the word is, the smaller the cost will be, so if we just use the plain cost, the algorithm will prefer short words. Therefore, we need to give some penalty to shorter words so that short words and longer words have similar cost. You can also consider this as normalizing the cost function over words. One way to penalize shorter words is to multiply some factor that depends on the word length.

$$P' = P / \gamma^T$$

P' : normalized probability, P : probability,
γ : normalization factor($0 < \gamma < 1$), T : word length

Normalizing this way, the cost function maintains high value just as long as the word we are evaluating is the correct one and we have not gone beyond its end. On the other hand, if the word we are evaluating is the incorrect one or we have already passed its end, the cost increases sharply. We can set the normalization factor γ empirically during training so that $P' = 1$ when we know the correct word sequence. $\gamma = \sqrt[T]{\alpha_{\max}(T)}$, where $\alpha_{\max}(t)$ is the maximum forward probability for $P(x_1^t \mid w_1^k, s_t)$ for all the states of w_k .

Results and Future Work

We have tried different HMM model parameters for the word model to compare speed and accuracy. We have not collected enough data yet, but we have observed some clear evidence of how the model parameters affect the performance of this algorithm. The number of words in the vocabulary only increases the search space by a linear amount in this method, since this method considers one word at a time. The number of states and Δ (the number of states allowed to jump over) in each word model affects the search space more significantly. It affects the search space by the order of $O\left(n^2(\Delta+1)\right)$. The order of output vector and the number of literals is the size of the feature vector, so it does not affect the search space but only affects computation time. If these parameters are too small, there will not be enough information to distinguish between words. The following is one example of model parameters that recognizes 95% of the words correctly in the given sequence under Java2 environment with constraint of Pentium 1.2GHz and 0.5 sec/word recognition rate.

Table 1. HMM Model parameters

Number of words in vocabulary	200
Number of states	12
Δ	3
Output vector order	16
Number of literals	3

The key to the speed of this suggested algorithm is in the heuristic function. We have used one type of heuristic that worked efficiently. To evaluate and improve the performance of this proposed algorithm, we should try using different heuristics and also compare it with other algorithms.

Acknowledgements

This work was done under the guidance of Prof. C. D. Yoo while the author was with Korea Advanced Institute of Science and Technology (KAIST).

References

[1] *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*, X. Huang, A. Acero, H. Won, 2001
[2] *Decoding Algorithm in Statistical Machine Translation*, Ye-Yi Wang and Alex Waibel, 1997
[3] *A tutorial on Hidden Markov Models and selected applications in speech recognition*, L. R. Rabiner, 1989
[4] *The HTK Book v.2.1*, Steve Young, et. Al