A LOWER-COMPLEXITY VITERBI ALGORITHM

Sarvar Patel

Bellcore
445 South St.
Morristown NJ 07960
sarvar@bellcore.com

ABSTRACT

In continuous speech recognition, when using statistical language models (e.g. bigrams) a significant amount of time is used every frame to evaluate interword transitions. In fact, if N is the size of vocabulary, $\mathcal{O}(N^2)$ operations are required per frame. Also, when evaluating fully connected HMM with N states, the Viterbi algorithm requires $\mathcal{O}(N^2)$ operations per frame.

This paper presents the first algorithm to break the $\mathcal{O}(N^2)$ complexity requirement in the Viterbi algorithm, whether evaluating interword transitions or evaluating a fully connected HMM. The algorithm presented has an average complexity of $\mathcal{O}(N\sqrt{N})$. Previous speed-ups of the evaluations of interword transitions used heuristics, like pruning, or relied upon unavailability of many of the bigram values. However, this paper does not rely on any heuristics but fundamentally improves the basic evaluation of the time synchronous Viterbi algorithm.

Introduction

In continuous speech recognition, the Viterbi algorithm is often used to find the most likely path given the observation sequence derived from the speech signal. When statistical language models (e.g. bigrams) are used, a significant amount of computation is performed every frame to evaluate the interword transitions. Since every word could be preceded by every other word in the vocabulary, there are $\mathcal{O}(N^2)$ operations per frame for interword transitions (where N is the size of the vocabulary). Pruning [1] can be used to reduce computation, but at the cost of sacrificing optimality. Also, computation can be reduced if one has not 'seen' all of the possible bigrams [2].

If, however, we have complete bigram values and we are interested in the optimal answer, then we need to perform $\mathcal{O}(N^2)$ operations per frame using the standard algorithm. This paper reports the first improvement on that bound by introducing an algorithm with average complexity of $\mathcal{O}(N\sqrt{N})$ or $\mathcal{O}(N^{1.5})$ operations per frame. The evaluation of interword transitions is similar to the evaluation of interstate transitions of an HMM. While evaluating interword transitions we are searching for the best preceding word using the probability scores at time $t-1$ and the bigram values (see Figure 1). Similarly, in an HMM we are searching for the best preceding state using the probability scores at time $t-1$ and the interstate transition matrix. The standard evaluation of the best state path of a fully connected HMM with N states requires $\mathcal{O}(N^2)$ operations per frame [3]. Hence, the algorithm presented here also lowers the bound to evaluate a fully-connected HMM. The improved Viterbi algorithm requires on the average $\mathcal{O}(N\sqrt{N})$ operations per frame. We are not using better heuristics or exploiting incomplete information, but fundamentally reducing the computations involved in the Viterbi algorithm. The rest of the paper will present the algorithm in terms of reducing the number of operations for evaluating the interword transitions.

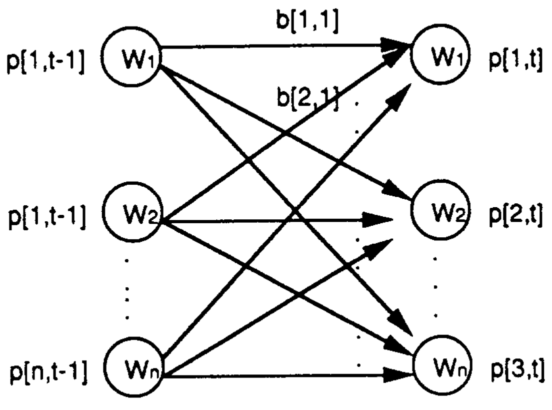

Figure 1: Standard bigram grammar, with N-squared interword transitions to be evaluated each frame.

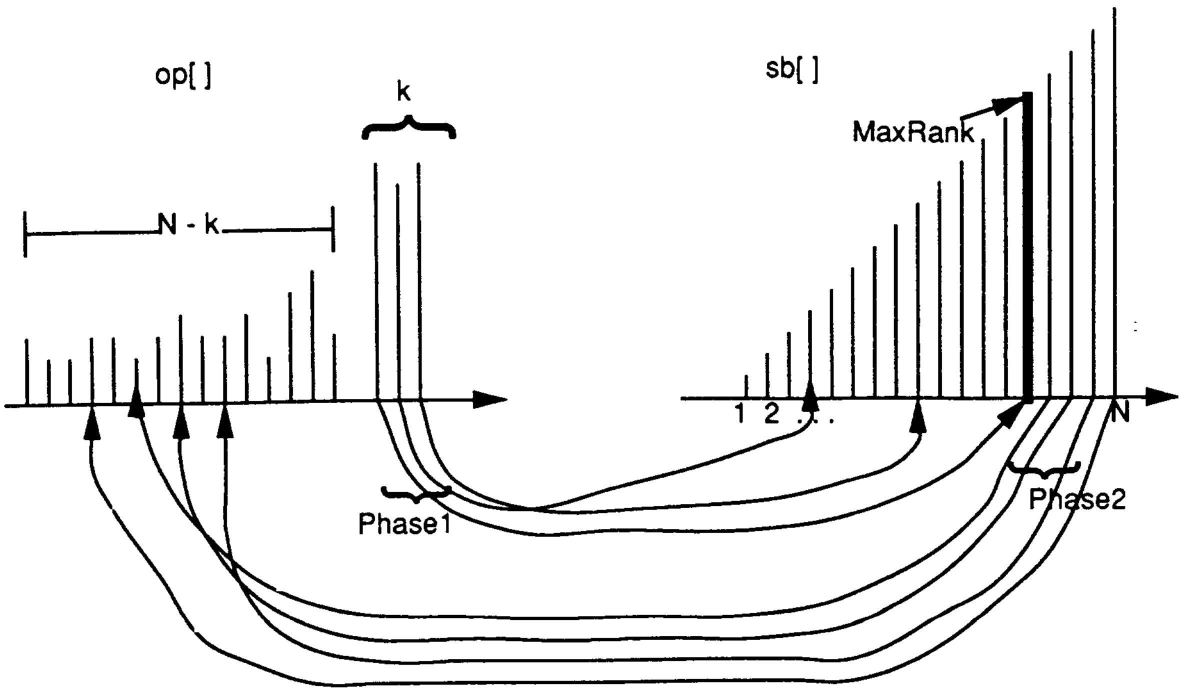

Figure 2: Two-phase algorithm for efficiently finding max(sb[ ] * op[ ]).

Motivation

For each word w at time t we have to find the best preceding word at time $t-1$ (see Figure 1). That is:

for w=1,2,..N,
p[w,t] = max(p[pw,$t-1$] * b[pw,w]) for pw=1,2,..N

> w is the current word; pw is the previous word;
> p[w,t] is the probability score of best path ending at word w at time t.
> b[pw,w] is the bigram weight of word w preceded by word pw.

For each word w, N multiplications and $N-1$ comparisons are needed to find the maximum value of the products p[pw,$t-1$] * b[pw,w]. Since there are N words, $\mathcal{O}(N^2)$ operations are required per frame. The number of operations can be reduced if we can find the best previous word in less than N operations; specifically if we can calculate max (p[1,$t-1$]*b[1,w], p[2,$t-1$]*b[2,w], ..., p[N,$t-1$]*b[N,w]) without having to multiply all of the p[ ]*b[ ] products and comparing them. This reduction will in fact result if, when evaluating the N p[ ]*b[ ] expressions, we find out, before all expressions are evaluated, that the remaining p[ ]'s and b[ ]'s all have lower probabilities than the p[ ] and b[ ] of one of the p[ ]*b[ ] expressions calculated thus far. In that case, there is no reason to evaluate remaining p[ ]*b[ ]'s, because they will only have lower probabilities than the current maximum p[ ]*b[ ].

This is the insight that we will exploit to answer how exactly one knows that no more expressions need to be evaluated, and how to determine the best order to evaluate these expressions so that a minimum number of them are evaluated. Furthermore, we will specify the complexity of the resulting algorithm.

Algorithm

An array sb[pw,w] retrieves bigram probabilities from b[pw,w] in sorted order. The array sb[ ] is created once and remains static throughout the recognition phase, since the bigram probabilities are fixed. For every frame, the top k values of array p[ ] at time $t-1$ are found and stored in an array op[ ] (for ordered p[ ]); the value for k will be specified later. We see in Figure 2 [left side] that op[ ] has k values which are all larger than the remaining $N-k$ scores. There is no sorting done among the top k values, however. The top k values can be found for each frame using the selection algorithm for order statistics [4] which surprisingly requires $\mathcal{O}(N)$ operations [1].

[1] Typically, in order to find the largest k values, we use sorting which requires $N \log N$ operations, but this is not necessary for

After the selection algorithm for op[ ] is performed, for each word w, at time t the best preceding word is found in two phases. Figure 2 will be used to illustrate this algorithm.

In phase 1, each of the k expressions, op[ ]*sb[ ], (see footnote [2]) is evaluated and, if necessary the maximum value is updated, along with *Maxrank*, the index of the highest sb[ ] value used in one of the k op[ ]*sb[ ] expressions. It is clear from Figure 2 [right side] that none of the remaining sb[ ]'s to the left of *Maxrank* can result in the maximum value of op[ ]*sb[ ] because none will have op[ ] larger than the op[ ] associated with sb[*Maxrank*,*w*]. That is, the op[ ] will have to come from the remaining $N-k$ op[ ]'s which all have lower values. The sb[ ]'s to the left by definition have lower values than sb[*Maxrank*,*w*] because the array is sorted. Therefore, the only possible way of getting a larger maximum value is to evaluate the expressions involving sb[ ]'s which are to the right of the *Maxrank* in Figure 2.

This is done in phase 2, where the sb[ ]'s to the right of *Maxrank* have the appropriate expression op[ ]*sb[ ] evaluated and the maximum value updated if necessary. There are *N*-*Maxrank* expressions evaluated in phase 2. The algorithm will always terminate with the correct maximum value, and k + (N - *Maxrank*) expressions are evaluated in total. But what value should we use for k and what is the resulting number of p[ ]*b[ ] expressions that are evaluated? If we knew the expected value of *Maxrank*, then we could discover the optimum value for k and the average number of expressions evaluated.

Complexity Of The Algorithm

The average number, $f(k)$, of expressions evaluated for a specific k is: $f(k) = k + (N - E[Maxrank])$. The expected value of *Maxrank*, $E[Maxrank]$, is $k(N+1)/(k+1)$ and is derived in Appendix A. Therefore $f(k) = k + (N - k(N+1)/(k+1))$. In order to find the best value for discrete k, the 1st derivative of the continuous function $f(x)$ is taken with respect to the continuous variable x and set equal to 0. The function $f(x)$ has exactly one minimum between 1 and N. The x which solves the equation $f'(x) = 0$ will be the value which minimizes $f(x)$, the number of operations. The optimal value, x_{opt}, is then $\sqrt{N+1} - 1$, and the average number of expressions evaluated is $2(\sqrt{N+1} - 1)$, which is $\mathcal{O}(\sqrt{N})$ operations. The integer k is then chosen as the ceiling or floor of, x_{opt}, whichever has a lower $f()$ value. Since there are N words for which we have to find out the best previous word to propagate the recognized word sequence from, we require $\mathcal{O}(N\sqrt{N})$ operations per frame. The selection done in phase 1 requires only $\mathcal{O}(N)$ operations and does not change the overall $\mathcal{O}(N\sqrt{N})$ nature of the algorithm. For N equal to 1000, the standard Viterbi would require on the order of N^2 or 1,000,000 operations to be evaluated, while our algorithm requires on average on the order of $N^{1.5}$ or 32,000 operations to be evaluated. Since the constants involved in the complexity of the algorithm are small, this gives a reasonable view of the magnitude of speed up possible.

our task. What we need is the separation of the largest k values from the lowest $N-k$ values, but we don't need the the k values sorted among themselves nor do we need the $N-k$ values sorted among themselves. The constant in the selection algorithm is small enough to make the algorithm practical [4].

[2] The jth sb[j] associated with the ith op[i] is found by mapping arrays which would map op[i] as the value associated with the word h, and then another mapping array would find the sb[j] associated with word h.

Conclusion

We have presented a new algorithm which performs the time-synchronous Viterbi evaluation. We also proved that the new algorithm requires on average $\mathcal{O}(N\sqrt{N})$ number of operations per frame as oppose to the $\mathcal{O}(N^2)$ required by the standard Viterbi algorithm. The selection algorithm, requiring $\mathcal{O}(N)$ operations, is used to find the top k values every frame. However, it should be clear that it is not the fast nature of the selection algorithm which gives us the overall $\mathcal{O}(N\sqrt{N})$ number. Even if we sorted the N numbers to find out the top k values for every frame, $\mathcal{O}(N \log N)$ operations would be required. The overall number of operations will still be dominated by $\mathcal{O}(N\sqrt{N})$.

Heuristics, like pruning, have been successfully used in the standard Viterbi algorithm because they increased speed by sacrificing little in accuracy. Partly due to the success of these heuristics, and the acceptance of the $\mathcal{O}(N^2)$ barrier, there has been little effort in improving the speed of the time synchronous Viterbi. Although this paper does not show how to break the $\mathcal{O}(N^2)$ barrier in worst case, it does show how to break the barrier in the average case with appropriate pre-processing.

This opens up a whole host of exciting theoretical and practical questions. It is very likely that there are algorithms faster than $\mathcal{O}(N\sqrt{N})$ to be discovered. What is a non-trivial lower bound for the average number of operations required for the time synchronous Viterbi? Is an $\mathcal{O}(N \log N)$ algorithm possible? Although the worst case number of operations per frame is $\mathcal{O}(N^2)$, would an amortized worst case analysis of the algorithm in this paper yield less than $\mathcal{O}(N^2)$? [An amortized analysis looks at number of operations required over a sequence of frames as oppose to a single frame]. And finally, perhaps ideas similar to those in the paper can be mixed with pruning and other heuristics to create novel algorithms.

Acknowledgement

I would like to thank David Pepper, Rashid Ansari, Candace Kamm, Sharad Singhal, and Fure-ching Jeng for comments on the paper.

Appendix A

Proof: E[$Maxrank$] = $k(N+1)/(k+1)$.

P($Maxrank$=i) : probability that the maximum value is i when k values from 1..N are picked without repetition. That is, one value is i and the remaining $k-1$ values are less than i.

$$P(Maxrank{=}i) = \text{\# of sets of size } k \text{ containing } i \text{ as max } / \text{\# of sets with } k \text{ values.}$$

$$= \frac{\binom{i-1}{k-1}}{\binom{N}{k}}$$

since, the i element is fixed in the numerator.

Thus,

$$E[Maxrank] = \sum_{i=0}^{N} i\frac{\binom{i-1}{k-1}}{\binom{N}{k}} = \sum_{i=0}^{N} k\frac{\binom{i}{k}}{\binom{N}{k}}$$

$$= \frac{k}{\binom{N}{k}}\sum_{i=0}^{N}\binom{i}{k}$$

$$= \frac{k}{\binom{N}{k}}\binom{N+1}{k+1}$$

$$= \frac{k}{k+1}(N+1).$$

See Reference [5] for summation identity.

References

[1] B. Lowerre, D. R. Reddy, "The HARPY speech understanding system", in *Trends in Speech Recognition* (Lea, W. ed.), pp 340–346. Prentice–Hall, Englewood Cliffs NJ, 1980.
[2] Steve Austin, Pat Peterson, Paul Placeway, Richard Schwartz, Jeff Vandergrift, "Toward a Real–Time Spoken Language System Using Commercial Hardware", *Proceedings DARPA Speech and Natural Language Workshop*, pp 72–77, June 1990.
[3] Lawrence R. Rabiner, "A Tutorial on HMMs and Selected Applications in Speech Recognition," *Proceedings of the IEEE*, Vol 77, No 2, pp 257-286, Feb 1989.
[4] Robert W. Floyd, Ronald L. Rivest, "Expected Time Bounds for Selection,' *Communications of the ACM*, Vol 18, No 3, pp 165–173, March 1975.
[5] Ronald Graham, Donald Knuth, Oren Patashnik, *Concrete Mathematics*, Addisson–Wesley, New York, 1994.