AN $\mathcal{O}(N\sqrt{\overline{E}})$ VITERBI ALGORITHM

Sarvar Patel

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

ABSTRACT

In continuous speech recognition, a significant amount of time is used every frame to evaluate interword transitions. In fact, if N is the size of the vocabulary and each word transitions on average to $\overline{E}$ other words then $\mathcal{O}(N\overline{E})$ operations are required. Similarly when evaluating a partially connected HMM, the Viterbi algorithm requires $\mathcal{O}(N\overline{E})$ operations.

This paper presents the first algorithm to break the $\mathcal{O}(N\overline{E})$ complexity requirement. The new algorithm has an average complexity of $\mathcal{O}(N\sqrt{\overline{E}})$. An algorithm was previously presented by the author for the special case of fully connected models, however, the new algorithm is general. It speeds up evaluations of both partial and fully connected HMM and language models. Unlike pruning, this paper does not use any heuristics which may sacrifice optimality, 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 observations 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. In a partially-connected grammar, a word on average could be succeeded by $\overline{E}$ other words, hence there are $\mathcal{O}(N\overline{E})$ operations per frame for interword transitions (where N is the size of the vocabulary). Partially-connected grammar or model may occur because of the syntax of a chosen language or simply because we have not 'seen' enough bigram values and are forced to use some null values for those unseen bigrams[2]. Pruning [1] can be used to reduce computation, but at the cost of sacrificing optimality. If, however, we are interested in the optimal answer, then we need to perform $\mathcal{O}(N\overline{E})$ operations per frame using the standard Viterbi algorithm. This paper reports the first improvement on that bound by introducing an algorithm with average complexity $\mathcal{O}(N\sqrt{\overline{E}})$ operations per frame. As an example, with a vocabulary size $N = 5000$ and average branching factor $\overline{E} = 400$, $N\overline{E}$ equals 2,000,000 while $N\sqrt{\overline{E}}$ equals 100,000! Since the constants involved are small this gives a reasonable view of the magnitude of the speed-ups that are possible.

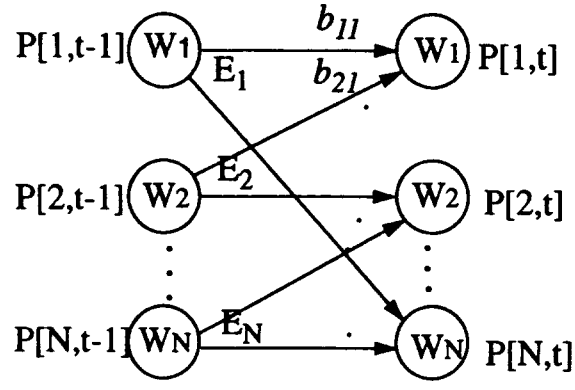

Figure 1: Partially connected bigram grammar, with $\overline{E}$ transitions on average to and from other words.

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. Similarly, in an HMM we are searching for the best preceding state using probability scores at time $t-1$ and the interstate transition matrix. The standard evaluation of the best state path of an HMM with N states and on average $\overline{E}$ transitions per state requires $\mathcal{O}(N\overline{E})$ operations per frame[3]. Hence, the algorithm also lowers the bound to evaluate a partially-connected HMM.

Copyright 1997 IEEE

When $\overline{E}$ equals N then all words can transition to all other words, and we get the special case of a fully connected model and the algorithm requires $\mathcal{O}(N\sqrt{N})$ operations. The author had previously presented [5] an $\mathcal{O}(N\sqrt{N})$ algorithm to speed up this special case of a fully connected model. The current algorithm is completely general and speeds-up evaluations for all sizes of $\overline{E}$ and not just when $\overline{E}$ equals N. This algorithm does not use heuristics but fundamentally reduces the computations involved in the Viterbi evaluation. The rest of the paper will present the improved algorithm in terms of reducing the number of operations for evaluating interword transitions.

Previous Algorithm

In order to understand the algorithm the reader needs to be familiar with how maximum of N expressions can be evaluated in $\mathcal{O}(\sqrt{N})$ operations if we have two pre-sorted arrays (actually one pre-sorted and one pre-ordered). We see this is useful in reducing required operations for fully connected models from $\mathcal{O}(N^2)$ to $\mathcal{O}(N\sqrt{N})$ operations. If the reader is already familiar with [5] then this section can be skipped.

In standard Viterbi, for each word w at time t we have to find the best preceding word at time $t-1$. 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. The rest of the section shows how this can be done (see [5] for details).

An array sb[pw,w] indexes b[pw,w] in sorted order. The array sb[ ] is created once and remains static throughout the recognition phase. 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 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 requires $\mathcal{O}(N)$ operations.

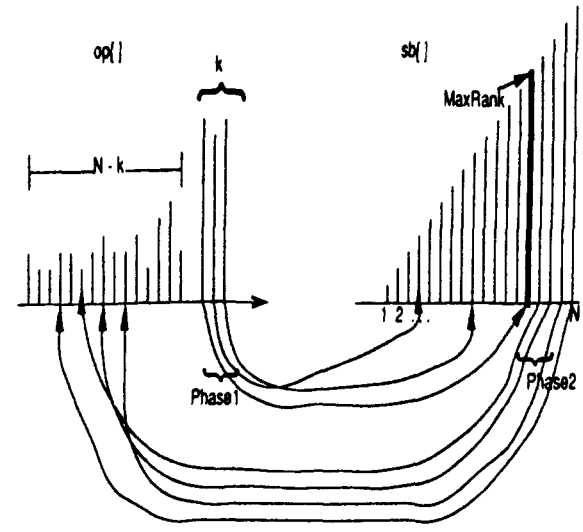

Figure 2: Two-phase algorithm for efficiently finding max(sb[]*op[]).

After the selection algorithm for op[ ] is performed, for each word w, 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[ ] is evaluated while the maximum product is updated, as well as the Maxrank, the index of the highest sb[ ] value used in one of the k op[ ]*sb[ ] expressions. It is clear from Figure 2 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, 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 p[ ]*b[ ] evaluated and the maximum value updated if necessary. There are N-Maxrank expressions to be evaluated. The algorithm will always terminate with the correct maximum product, and k + (N - Maxrank) expressions are evaluated. In [5] the best value for k is

Copyright 1997 IEEE

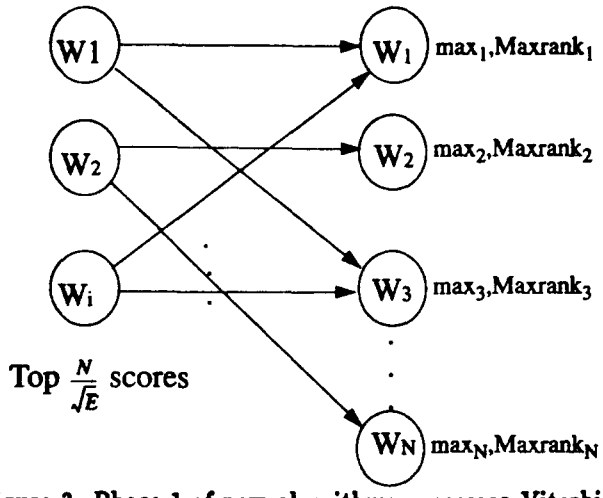

Figure 3: Phase 1 of new algorithm: successor Viterbi is performed on top words at $t-1$.

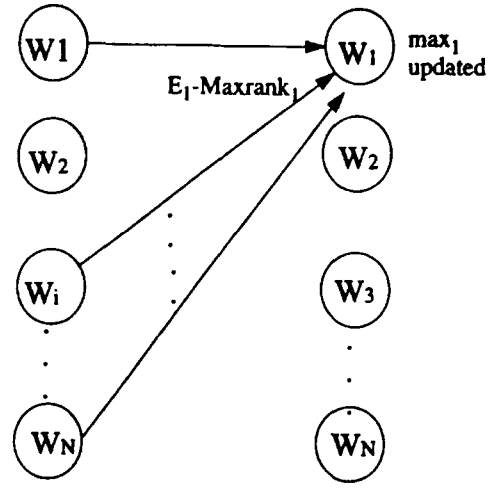

Figure 4: Phase 2 of new algorithm: predecessor Viterbi is performed on E_w-Maxrank expressions for every word w (only first word is shown).

derived to be $\sqrt{N+1}-1$, expected value of Maxrank is $k(N+1)/(k+1)$ and thus the total number of expressions evaluated on average are $\mathcal{O}(N\sqrt{N})$.

New Algorithm

The previous algorithm is efficient in evaluating fully connected models, and even in a partially-connected model with a large $\overline{E}$ it may still be more efficient than standard Viterbi. However, we present a new algorithm which is more efficient than standard Viterbi and the previous algorithm for partially-connected models in general. Lets say word w at time t has E_w edges or transitions coming into it. If we can somehow access the top $\sqrt{E_w}$ p[pw,t-1] values with transitions to w and evaluate the associated p[pw,t-1]*b[pw,w] expressions then on average we would only need to do $\sqrt{E_w}$ more expressions as presented in previous section, thus requiring a total of $2\sqrt{E_w}$ or $\mathcal{O}(\sqrt{E_w})$ expressions evaluated. Although we can find top $\sqrt{E_w}$ p[pw,t-1] values, not all will transition into word w. Thus we would have to check more than top $\sqrt{E_w}$ p[pw,t-1], and now we cannot make the claim that the algorithm only requires $\mathcal{O}(\sqrt{E_w})$ expressions. This complicates the situation and requires a modified algorithm. The modified algorithm is still simple, but its slightly more complicated to prove bounds on the average running time of the algorithm.

There are actually two ways to perform a standard Viterbi, and we use both ways in the two phases of our algorithm. First, the standard Viterbi can be evaluated by expanding successor words from t-1 to time t. At t many p[w,t] values are simultaneously updated as some word pw expands into many words at t. Each w reached from pw updates its value by keeping the maximum of its previous value and the new p[pw,t-1]*b[pw,w] value. After all pw words at t-1 are expanded then at time t all words will have the proper p[w,t]. We will call this the 'successor Viterbi.' Secondly the standard Viterbi can be evaluated in the way we did in the preceding section. At time t for word w, we evaluate p[pw,t-1]*b[pw,w] for all words pw which can precede word w and keep updating p[w,t] value to be the maximum of the expressions. Once we are finished with word w then we move onto the next word w+1 and do the same. We will call this, the 'predecessor Viterbi.'

The new algorithm works as such: First the top $N/\sqrt{\overline{E}}$ values of array p[] at time t-1 are found and stored in op[] for ordered p[]; In phase 1 (see Figure 3), 'successor Viterbi' is performed on the top $N/\sqrt{\overline{E}}$ words; i.e. all successor words to the top $N/\sqrt{\overline{E}}$ words at t-1 are expanded and the appropriate words w at t have their scores updated. However, along with the maximum value being kept as the score, Maxrank value is also updated. Maxrank for word w at t indicates the index of the highest sb[] value used in one of the k op[]*sb[] expressions. Although $N/\sqrt{\overline{E}}$ words at t-1 were expanded, we don't know how many transitioned into a given word w at t, hence, we said k op[]*sb[] expressions evaluated for w and not $N/\sqrt{\overline{E}}$. In Phase 2 (see Figure 4), we will now perform 'preceding Viterbi' on all words w at t. That is for word w at t, we will only evaluate the remaining N-Maxrank (E_w-Maxrank to be precise) expressions associated with sb[]'s to the right of Maxrank for w. We know from previous section and figure 2 that expressions associated to the left of sb[]'s cannot possibly result in a larger value. We do

Copyright 1997 IEEE

this for all w at t and thus all words at t will have their correct p[w,t] values.

Complexity of the Algorithm

At first, there doesn't seem to be much hope that we can get any meaningful bounds on the complexity of the algorithm, but we will see how this is possible. Lets just consider one word w: In phase 1 we assumed that k words reached word w with some probability prob[k]. And in phase 2, $E_w - Maxrank$ further expressions were evaluated. The expected value for Maxrank is $k(E_w + 1)/(k + 1)$ and is derived in [5]. E_w is the total number of edges or transitions into word w. So altogether for word w we have to evaluate k expressions in phase 1 plus $E_w - E[Maxrank]$ phase 2 expressions. So f(k), the total number of expressions evaluated for a particular k is $k + E_w - k(E_w + 1)/(k + 1)$ or after simplification $(k^2+E_w)/(k+1) = k^2/(k+1)+E_w/(k+1)$.

Prob[k] is actually a hypergeometric distribution, but since $N/\sqrt{E}$ is large, we can approximate it well with the binomial distribution. Binomial distribution b(k;n,p) is $\binom{n}{k}p^k(1-p)^{n-k}$. That is probability of k successes out of n where a success occurs with probability p. In our algorithm there are E_w words out of N words which will transition into word w. So the probability of success that a given word picked in phase 1 will transition into w at t is $p = E_w/N$; $N/\sqrt{E}$ is n; Now we can calculate Expected$[f(k)]$ for w.

$$E[f(k)] = \sum_{k=0}^{n} f(k)prob(k)$$
$$= \sum_{k=0}^{n}\left(\frac{k^2}{k+1} + \frac{E_w}{k+1}\right)\binom{n}{k}p^k q^{n-k}.$$

$$\sum_{k=0}^{n}\frac{k^2}{k+1}\binom{n}{k}p^k q^{n-k} \le \sum_{k=0}^{n} k\binom{n}{k}p^k q^{n-k}$$
$$= np = \frac{N}{\sqrt{E}}\frac{E_w}{N} = \frac{E_w}{\sqrt{E}}.$$

$$\sum_{k=0}^{n}\frac{E_w}{k+1}\binom{n}{k}p^k q^{n-k} =$$
$$\frac{E_w}{(n+1)p}\sum_{k=0}^{n}\binom{n+1}{k+1}p^{k+1}q^{(n+1)-(k+1)}$$
$$\frac{E_w}{(n+1)p}\sum_{u=1}^{n+1}\binom{n+1}{u}p^u q^{(n+1)-(u)}$$
$$\frac{E_w}{(n+1)p}(1 - b(0;n+1,p))$$
$$\frac{E_w}{(n+1)p}(1 - q^{n+1}) \le E_w/(np)$$
$$\frac{E_w}{((N/\sqrt{E}) * (E_w/N))} = \sqrt{E}$$

Therefore $E[f(k)] \le E_w/\sqrt{E} + \sqrt{E}$. Since there are N words, the total number of operations $= N * E[f(k)] \le N * (E_w/\sqrt{E} + \sqrt{E})$. E_w the number of transitions in to a word is distributed by some prob$[E_w]$ with mean $\sqrt{E}$ over the N words. Therefore

$$\sum_{E_w=0}^{N} N\left(\frac{E_w}{\sqrt{E}} + \sqrt{E}\right) * prob[E_w] =$$
$$\frac{N}{\sqrt{E}}\sum_{E_w=0}^{N} E_w * prob[E_w] + N\sqrt{E} * 1 =$$
$$\frac{N}{\sqrt{E}}\overline{E} + N\sqrt{E} = 2N\sqrt{E}$$

Hence the total number of operations required is $\mathcal{O}(N\sqrt{E})$.

Conclusion

We have presented a faster algorithm for performing Viterbi evaluations on partially connected models. This algorithm has average complexity $\mathcal{O}(N\sqrt{E})$ and is theoretical interesting because until now it was assumed that $\mathcal{O}(N\overline{E})$ operations are necessary. Since the constants involved in the complexity bounds are small, this algorithm is also highly practical.

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] S. Austin, P. Peterson, P. Placeway, R. Schwarts, J. Vandergrift, "Toward a Real-Time Spoken Language System Using Commercial Hardware," *Proc. DARPA Speech and Natural Language Workshop*, pp 72-77, 6/90.

[3] Lawrence Rabiner, "A Tutorial on HMMs and Selected Applications in Speech Recognition," *Proc. of the IEEE*, Vol 77, No 2, pp 257-286, Feb 1989.

[4] R. Floyd, R. Rivest, "Expected Time Bounds for Selection,' *Communications of the ACM*, Vol 18, No 3, pp 165–173, March 1975.

[5] Sarvar Patel, "A Lower-Complexity Viterbi Algorithm", *Proc. ICASSP 95*, pp 592-595, 1995.

Copyright 1997 IEEE