COMPRESSION OF ACOUSTIC FEATURES FOR SPEECH RECOGNITION IN NETWORK ENVIRONMENTS

Ganesh N. Ramaswamy   *Ponani S. Gopalakrishnan*

IBM Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT

In this paper, we describe a new compression algorithm for encoding acoustic features used in typical speech recognition systems. The proposed algorithm uses a combination of simple techniques, such as linear prediction and multi-stage vector quantization, and the current version of the algorithm encodes the acoustic features at a fixed rate of 4.0 Kbit/s. The compression algorithm can be used very effectively for speech recognition in network environments, such as those employing a client-server model, or to reduce storage in general speech recognition applications. The algorithm has also been tuned for practical implementations, so that the computational complexity and memory requirements are modest. We have successfully tested the compression algorithm against many test sets from several different languages, and the algorithm performed very well, with no significant change in the recognition accuracy due to compression.

1. INTRODUCTION

The first step in a speech recognition system involves the computation of a set of acoustic features from sampled speech. One common set of acoustic features is the set of mel-cepstral features, which can be computed as described in [1] or elsewhere. These acoustic features are then submitted to a speech recognition engine where the utterances are recognized. For speech recognition systems in network environments, such as those employing a client-server model, typically the acoustic features are computed on the client system and transmitted to the server system for recognition. In this scenario, it is highly desirable to compress the acoustic features to minimize the bandwidth requirements for the transmission. Compression is also useful in more general speech recognition systems where storage of the acoustic features is needed.

Although the topic of speech compression has been well researched over the years (see [3] for a summary), all of the proposed solutions only address the problem of compressing and reproducing speech that sounds acceptable to a human ear, and not the problem of compressing the acoustic features computed from spoken utterances, for the purpose of subsequent machine recognition of speech. Hence, a new compression algorithm is needed.

There are several constraints that need to be addressed in developing such an algorithm. In addition to reducing the bandwidth required to transmit the data, such that client-server speech recognition is viable using very low-bandwidth wide-area wireless links, it is crucial that the compression does not introduce noise that degrades the recognition accuracy. At the same time, since some of the client devices used may have limited computational resources, it is also highly desirable that the compression algorithm has low computational complexity and low memory requirements.

In this paper, we propose a new compression algorithm to encode the acoustics features at a fixed rate of 4.0 Kbit/s that satisfies the above requirements and constraints. This is the first in the family of compression algorithms that we plan to introduce in the future that addresses the problem of compressing acoustic features. In the version presented in this paper, we use a combination of simple techniques, ranging from linear prediction to multi-stage vector quantization, along with strategies to minimize the computational resources required. We also describe some of the experiments that were done to evaluate the performance and the robustness of the new algorithm, using a number of test sets drawn from several different languages.

The remainder of the paper is divided into four sections. In Section 2, the compression algorithm is described in detail. In Section 3, several practical implementation issues are addressed. Section 4 contains the results of some of the experiments that were conducted to evaluate the algorithm. Section 5 concludes the paper.

2. THE NEW COMPRESSION ALGORITHM

Figure 1 shows a speech recognition system involving compression and decompression of acoustic feature vectors. We will assume that the feature vectors are 13-dimensional vectors consisting of mel-cepstral features, as noted in Section 1. We also assume that the feature vectors are computed every 10 ms, which is the typical frame rate in most speech recognition systems. The proposed compression algorithm can easily be modified for systems employing different a feature set or a different frame rate.

Figure 2 describes the various steps in the compression process. The first step is the *Linear Prediction* step, which is designed to take advantage of the correlation (in time) between subsequent feature vectors. One simple method would be to take the difference between adjacent features vectors, which is a 1-step (scalar) linear prediction, and we found this to work very well in our test cases. Although we could use a multi-step multi-dimensional linear prediction, we found that the additional computational complexity is not worth the marginal improvements in performance. Hence, the version of the compression algorithm presented in this paper uses only a simple 1-step prediction, where the feature vector in the current frame is compared against the *encoded* feature vector from the previous frame, for all vectors except for the very first one. Since there will be no prior encoded vector to compare the very first vector, it will be compared against a *precomputed* mean vector. Therefore, in the terminology of Figure 2, the *data for prediction* is the precomputed mean vector for the first feature vector, and the encoded vector from the previous frame for all other feature vectors.

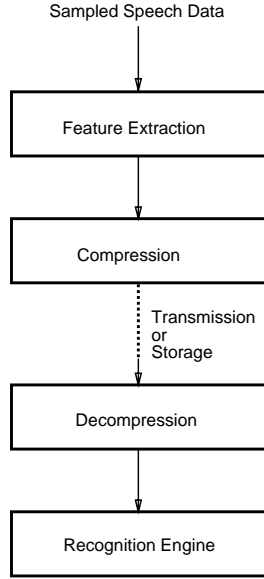The error vector from linear prediction is then subjected to

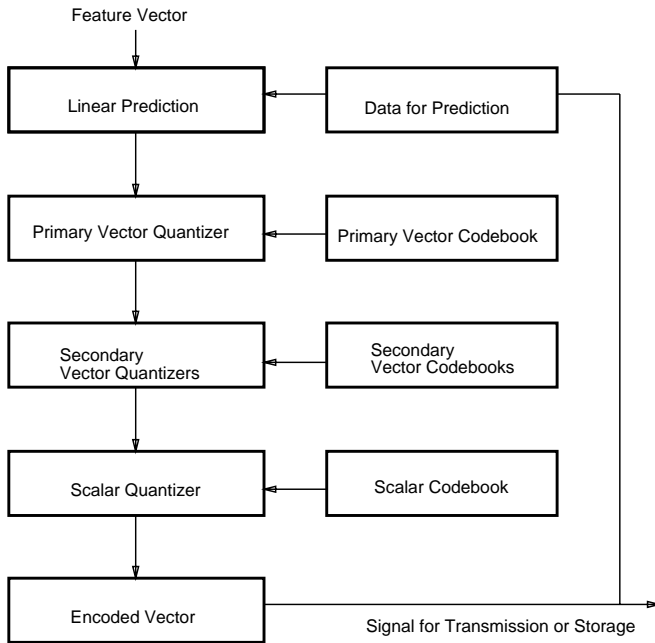Figure 1: **A Speech Recognition System Involving Compression and Decompression of the Acoustic Features.**

Figure 2: **The Compression Process.**

multi-stage vector quantization. In the current version of the compression algorithm, there are two vector quantizers. The *Primary Vector Quantizer*, takes the error vector from linear prediction and compares it to the entries in the *Primary Vector Codebook*, using a distance measure such as the Euclidean distance, and selects the entry in the codebook that best approximates the vector. In the experiments to be described in Section 4, we use a Primary Vector Codebook consisting of 4096 vectors of 13-dimensions each. The construction of the codebook and techniques for searching the codebook quickly will be described in Section 3.

The 13-dimensional residual vector remaining after the Primary Vector Quantization is then sent to the *Secondary Vector Quantizer(s)*, and also to the *Scalar Quantizer*. See Figure 2. Assuming that the original 13-dimensional feature vector contains the C0 (or the energy) coefficient of the mel-cepstral feature set as its 13th dimension, the residual vector from the Primary Vector Quantizer is partitioned into three subvectors of dimensions 6, 6, and 1, where the last subvector (which is a scalar) corresponds to the C0 or energy element. In the present implementation, only the first two 6-dimensional subvectors are used in the Secondary Vector Quantization, and the third 1-dimensional subvector is sent directly to the Scalar Quantizer (in Figure 2, we show the inputs to the Scalar Quantizer as coming from the Secondary Vector Quantizer and this may be slightly misleading, but the figure is intended to describe a more general scenario). In the Secondary Vector Quantization stage, each of the two 6-dimensional subvectors are assigned to the closest entries in the *Secondary Vector Codebooks* (each subvector has a separate codebook). As in the Primary Vector Quantization stage, we can again use the simple Euclidean distance as the metric for comparison. In the current version of the algorithm, we use two Secondary Vector Codebooks containing 4096 vectors of 6-dimensions each. Details concerning the construction of codebooks and searching strategies will be described in Section 3.

The third 1-dimensional subvector of the residual from the Primary Vector Quantizer is the only one to be subjected to Scalar Quantization, where we assign the subvector to the closest entry in the *Scalar Codebook*. In the current version, we use a Scalar Codebook containing 16 entries.

We treat the 13th dimension (the C0 or the energy coefficient) separately because this entry is usually much more sensitive to environmental and other variations, and treating the entry separately provides additional robustness. Although in the present version we start treating the 13th dimension separately only after the Primary Vector Quantization stage, it may be worthwhile to consider treating the the entry separately even before any quantization (for which we may have to take a slight bandwidth and/or performance hit).

After the Scalar Quantization stage, the compression process is essentially complete. All that needs to be done now is to assemble the indices corresponding to the chosen entries in the various codebooks, to form the final *Encoded Vector*. In the current version, we use 40 bits to form the Encoded Vector, with the first 12 bits allocated for the index into the Primary Vector Codebook, the second 12 bits allocated for the index into the first Secondary Vector Codebook, the third 12 bits allocated for the index into the second Secondary Vector Codebook and the last 4 bits allocated for the index into the Scalar Codebook. With a frame duration of 10ms, 100 Encoded Vectors are computed per second, for a fixed data rate of 4.0 Kbit/s. Without any compression, where 13-dimensional feature vectors in floating point representation have to be represented every 10 ms, the required data rate is 41.6 Kbit/s, and hence the

Table 1: **Experiments with WSJ Test Data.** The table below shows the errors rates for the various experiments performed on the WSJ test data. The "Uncompressed" column shows the results of the baseline test, where the data was not compressed. The last three columns show the results for three different compression experiments.

Speaker	Words	Uncompressed	Compressed I	Compressed II	Compressed III
a	696	6.2%	5.7%	5.6%	5.7%
b	731	12.2%	12.3%	12.3%	10.8%
c	660	19.8%	19.1%	19.5%	17.7%
d	704	9.8%	10.2%	9.8%	9.5%
e	714	8.4%	7.1%	7.8%	8.1%
f	673	10.3%	9.8%	10.8%	11.0%
g	665	11.0%	10.5%	10.8%	11.3%
h	691	16.1%	16.8%	16.1%	14.8%
i	610	8.9%	9.7%	9.3%	10.7%
Total	6144	11.4%	11.2%	11.3%	11.0%

compression algorithm provides for a reduction of bandwidth by a factor of more than 10.

The decompression stage is a straightforward process. The indices corresponding to the different codebooks are first extracted from the Encoded Vector, and the corresponding entries in the codebooks are obtained via simple table look-up steps, and added to the Data for Prediction (which is either the precomputed mean vector or the encoded vector from the previous frame) to reverse the Linear Prediction stage of the compression process. The reconstructed vector is then used for recognition.

We have chosen to focus on a fixed rate compression algorithm for the sake of simplicity. However, a variable rate scheme may provide additional enhancements, which may be desirable for some applications, and we intend to pursue such schemes in the future.

3. PRACTICAL CONSIDERATIONS

Since the Linear Prediction uses only a simple 1-step prediction, this stage is not computationally intensive. But the vector quantization stages can be quite intensive and fast searching strategies and other simplifications are necessary to reduce the computational resources required.

A common technique used to speed-up the searching of the codebooks is to use a *tree-structured* search [2], so that we search only a portion of the codebook instead of the whole codebook. For example, since our Primary Vector Codebook contains 4096 entries, we can group these entries into 64 groups of 64 entries each, and construct an intermediate codebook containing 64 entries. The entries in this intermediate codebook may be set to the centroid of the 64 vectors in each of the groups. With this approach, we can first search the intermediate codebook and find the best match, and then search only the corresponding group of 64 entries in the actual codebook for the final match. We call this the "64-64" tree structure for the 4096-entry codebook. Therefore, we only do 128 distance comparisons, as opposed to 4096 distance comparisons without the search strategy. Using such a search strategy may sometimes not provide the optimal results, but the computational savings are well worth the tradeoff.

In the experiments of Section 4, we use the 64-64 tree-structured search strategy for searching the Primary Vector Codebook, and the two Secondary Vector Codebooks which also contain 4096 vectors

each. Note that we have artificially forced each group to contain the same number of entries, which was done for the sake of maintaining simplicity. Additional performance improvements may be obtained by allowing different groups to have different number of entries, and in this case the computational load may vary according the to nature of the data.

One other issue that needs to be addressed in implementing the compression algorithm is the memory required to store the codebooks. The entries in the each of the codebooks would normally be floating point numbers, with each number requiring 4 bytes of storage. However, if we constrain all the entries to be integers, we can store them as short integers, requiring only 2 bytes of storage per integer. With this approach, we can cut the memory requirement by a factor of 2, and we will see in Section 4 that there is no significant performance loss in doing so. In this case, the memory required to store all the codebooks is approximately 200 KB, which is very reasonable for most systems. If additional reduction in memory is needed, we could partition the vectors into more subvectors and use more codebooks with fewer entries (but there may be some performance loss).

With the key issues pertaining to the implementation of the algorithm sorted out, constructing the codebooks becomes a straightforward process. We start with a collection of training data and calculate the feature vectors. The precomputed mean vector for use in the Linear Prediction stage for the first vector can be set to the mean of all the first vectors in each sentence of the training set. After that, the linear prediction step is applied to all the vectors in the training set, and we apply the popular K-means clustering algorithm [2] to the resulting vectors and generate all the codebooks.

4. EXPERIMENTAL EVALUATION

In this section we describe some of the experiments that were conducted to evaluate the proposed compression algorithm. All the codebooks described in the section were developed using a portion of corpus from the ARPA sponsored Wall Street Journal (WSJ) task.

Table 1 shows the results of the first set of experiments, where the test data consisted of a portion of the 1992 development test set from the WSJ corpus (see [1] for some additional information

about the test set). There were a total of 6144 words from 9 different speakers. We used a speaker independent continuous speech recognition system, with all the parameters of the recognition engine fixed for all the experiments (although the system used here is similar to the system described in [1], we used a version that was tuned for speed and not for performance, and therefore the recognition accuracy reported here for the baseline system is not the best). The detailed description of the recognition engine is omitted here since it is not relevant to the specific focus of evaluating the compression algorithm.

The column labeled "Uncompressed" shows the error rates for the baseline system, where we used the original uncompressed feature vectors for the recognition. The average error across all the speakers was 11.4% with the given version of the speech recognition engine.

The column labeled "Compressed I" shows the error rates for the first experiment. In this experiment, the codebooks contained floating point entries and we used the 64-64 tree structure for searching the Primary and Secondary Vector codebooks. We took the test data and subjected it to the compression and decompression, and the resulting feature vectors were used in the recognition process, with all other parameters and components of the recognition engine remaining fixed. The error changed only slightly for most of the speakers, with the average error *dropping* marginally to 11.2% after introducing the compression process. Despite the simplicity of the proposed algorithm and the relatively tight compression being applied, the performance was strikingly remarkable.

The column labeled "Compressed II" shows the error rates for the second experiment, where all entries in all of the codebooks were constrained to be (short) integers, to save on memory as we discussed in Section 3. The additional round-off errors appear to not make any significant difference, since the average error rate for this experiment was 11.3%, which is still lower than the baseline uncompressed case.

The column labeled "Compressed III" shows the error rates for the third experiment. In this experiment, we used the integer-only codebooks from the second experiment to compress and decompress all of the acoustic training data and generated new prototypes. With this step, the engine was be trained with data that closely matches the characteristics of the test data, and not surprisingly, the average error rate now drops further to 11.0%.

In the three experiments described in Table 1, both the test data and the data for generating the codebooks were drawn from the WSJ corpus. Even though we used different portions of the data for testing and for generating the codebooks, the basic characteristics of the data remain the same. Also, the WSJ data is "clean" and gives relatively low error rates. In order to test the robustness of the compression algorithm, we need to test it against data that would otherwise give high error rates. At the same time, we also need to determine if the compression algorithm is language independent.

Therefore, we subjected the compression algorithm to a very difficult test. We used the same codebooks from the WSJ data (which is all American English), and tested the compression algorithm against test sets (continuous dictation, read speech) from 5 different European languages. In particular, we intentionally chose test sets that had high error rates, for reasons noted above. Again, the recognition system used was a preliminary version which was not optimized for accuracy. It should be noted that in addition to the differences in the languages themselves, there are also differences in the data sets due to other reasons, such as the different microphones used to collect the data.

Table 2: **Experiments with High Error Rate Test Data.** The table below shows the error rates for experiments where the test data was drawn from several different languages, and was intentionally chosen such that the baseline test had high error rates, to illustrate the robustness of the algorithm.

Language	Words	Uncompressed	Compressed
French	6840	23.5%	23.6%
German	6764	13.2%	13.7%
Spanish	4730	7.2%	7.1%
Italian	6925	15.6%	15.6%
UK English	6290	24.2%	24.5%
Total	31549	17.3%	17.4%

Once again, the compression algorithm provided very good results, which are shown in Table 2. The high error rate of the baseline (uncompressed) case did not affect the performance of the compression algorithm significantly. Although the average error rate increases slightly due to compression, regenerating the codebooks with data from all the different languages should easily solve that problem. If additional improvements are desired, then we can regenerate the acoustic prototypes with data subjected to compression, as described previously.

5. CONCLUSIONS

In this paper, we described a new compression algorithm for encoding the acoustic features at a fixed rate of 4.0 Kbit/s. This is the first in the family of algorithms that we are developing to address the problem of compressing and encoding acoustic features in speech recognition. Despite its simplicity, modest computational complexity and the tight compression it provides, the algorithm performed very well in all the test sets, including those from different languages, and even when there were mismatches in the characteristics of the data used for generating the codebooks and the data used for testing.

The promising results presented in the paper open several new exciting avenues for further work, some of which were identified in this paper. We plan to develop a new framework for encoding acoustic features in general, incorporating new strategies and enhancements to the compression algorithm, and the results will be reported in the future.

REFERENCES

[1] Bahl, L. R., et. al., "Performance of the IBM Large Vocabulary Continuous Speech Recognition System on the ARPA Wall Street Journal Task," Proceedings of the IEEE ICASSP, Detroit, pp. 41-44, May 1995.

[2] Gersho, A., and Gray, R. M., *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, 1992.

[3] Klein, W. B., and Paliwal, K. K. (ed.), *Speech Coding and Synthesis*, Elsevier, 1995.