A simple scheme for compressing sparse representation of melodic music

Laura Rebollo-Neira
Mathematics Department
Aston University
B3 7ET, Birmingham, UK

Ivandro Sanches
Centro Universitario FEI
São Bernardo do Campo, São Paulo, Brazil

A simple scheme for compressing sparse representation of melodic music is outlined. The method is designed to store the output of sparse approximation of that class of signals. Sparsity is achieved using a trigonometric dictionary and a dedicated greedy strategy aiming at approximating the signal at good point-wise quality. Comparisons with the popular MP3 standard illustrate the suitability of the scheme for storing, in a small file, sparse approximation of melodic music.

Introduction: Sparse representation of music refers to techniques for approximating a music signal as a superposition of as few elements as possible. These elements, frequently called 'atoms', 'codebooks', or 'codewords', are selected from a redundant set, called a 'dictionary'. In the area of music information retrieval a number of different tasks have been shown to benefit by the sparsity of a representation [1–3]. This letter presents the proof of concept demonstrating that the outputs of sparse representation of melodic music can be stored in file of competitive size with respect to the popular MP3 format. The procedure is illustrated using a particular dictionary and a particular mathematical method for the selection of atoms. Nevertheless, the same scheme could be used with different dictionaries and/or other selection methods.

Let $\mathbb{R}^{N_b}$ be the space of real vectors of length N_b and $\mathcal{D}=\{\mathbf{d}_n \in \mathbb{R}^{N_b}; \|\mathbf{d}_n\|=1\}_{n=1}^{M}$, with $M > N_b$, a *dictionary* for $\mathbb{R}^{N_b}$. Given a signal $\mathbf{f} \in \mathbb{R}^{N_b}$ partitioned into Q frames $\mathbf{f}_q \in \mathbb{R}^{N_b}$, $q=1,\ldots,Q$, the k_q-term approximation $\mathbf{f}_q^{k_q}$ of each frame $\mathbf{f}_q$ is an atomic decomposition of the form

$$\mathbf{f}_q^{k_q} = \sum_{n=1}^{k_q} c^q(n)\mathbf{d}_{\ell_n^q}, \quad q=1,\ldots,Q. \tag{1}$$

The particular dictionary atoms $\mathbf{d}_{\ell_n^q}$, $n=1,\ldots,k_q$ in (1) will be selected through the Optimized Hierarchized Block Wise Optimized Orthogonal Matching Pursuit (OHBW-OOMP) method [4]. Hereafter the model (1) will be termed sparse approximation (SA) of a given signal partitioned into Q frames. The dictionary we consider is the union of the sine and cosine redundant dictionaries, which has been already shown to be suitable for approximating melodic music [4, 5]. In particular, a local measure of sparsity rendered by this dictionary has been shown to give meaningful information about the signal variation along time [5]. We follow on the previous work by presenting an effective procedure to save the outputs of the method.

A simple coding strategy: The coefficients resulting from approximating a signal by partitioning, as in (1), are converted into integer numbers as follows:

$$c_q^{\Delta}(n) = \lfloor \frac{|c_q(n)|}{\Delta} + \frac{1}{2} \rfloor, \tag{2}$$

where $\lfloor x \rfloor$ indicates the largest integer number smaller or equal to x and Δ is the quantization parameter. The signs of the coefficients, represented as $\mathbf{s}_q$, $q=1,\ldots,Q$, are encoded separately using a binary alphabet.

The indices of the atom in the atomic decompositions of each frame $\mathbf{f}_q$ are firstly sorted in ascending order $\ell_i^q \to \tilde{\ell}_i^q$, $i=1,\ldots,k_q$, which guarantees that, for each q value, $\tilde{\ell}_i^q < \tilde{\ell}_{i+1}^q$, $i=1,\ldots,k_q-1$. This order of the indices induces an order in the coefficients, $\mathbf{c}_q^{\Delta} \to \tilde{\mathbf{c}}_q^{\Delta}$ and in the corresponding signs $\mathbf{s}_q \to \tilde{\mathbf{s}}_q$. By defining $\delta_i^q = \tilde{\ell}_i^q - \tilde{\ell}_{i-1}^q$, $i=2,\ldots,k_q$ the follow string stores the indices for frame q with unique recovery $\tilde{\ell}_1^q, \delta_2^q, \ldots, \delta_{k_q}^q$. The number '0' is then used to separate the string corresponding to different frames and entropy code a long string, $st_{\rm ind}$, which is built as

$$st_{\rm ind} = [\tilde{\ell}_1^1, \ldots, \delta_{k_1}^1, 0, \tilde{\ell}_1^2, \ldots, \delta_{k_2}^2, 0, \cdots, \tilde{\ell}_1^{k_Q}, \ldots, \delta_{k_Q}^Q]. \tag{3}$$

The corresponding quantized magnitude of the coefficients are concatenated in the strings $st_{\rm cf}$ as follows:

$$st_{\rm cf} = [\tilde{c}_1^{\Delta}(1), \ldots, \tilde{c}_1^{\Delta}(k_1), \cdots, \tilde{c}_{k_Q}^{\Delta}(1), \ldots, \tilde{c}_{k_Q}^{\Delta}(k_Q)]. \tag{4}$$

Using '0' to store a positive sign and '1' to store negative one, the signs are placed in the string, $st_{\rm sg}$ as

$$st_{\rm sg} = [\tilde{s}_1(1), \ldots, \tilde{s}_1(k_1), \cdots, \tilde{s}_{k_Q}(1), \ldots, \tilde{s}_{k_Q}(k_Q)]. \tag{5}$$

The next encoding/decoding scheme summarizes the above described procedure.

Encoding

- Given a partition $\mathbf{f}_q \in \mathbb{R}^{N_b}$, $q=1,\ldots,Q$ of a signal, approximate it with $K=\sum_{q=1}^{Q} k_q$ atoms by the atomic decompositions:

$$\mathbf{f}_q^{k_q} = \sum_{n=1}^{k_q} c_q(n)\mathbf{d}_{\ell_n^q}, \quad q=1\ldots,Q. \tag{6}$$

- Quantize the absolute value coefficients in the above equation to obtain $c_q^{\Delta}(n)$, $n=1,\ldots,k_q$, $q=1,\ldots,Q$.
- For each q, sort the indices $\ell_1^q,\ldots,\ell_{k_q}$ in ascending oder to have a new order $\tilde{\ell}_1^q,\ldots,\tilde{\ell}_{k_q}$ and the re-ordered sets $\tilde{s}_q(1),\ldots,\tilde{s}_q(k_q)$, and $\tilde{c}_q(1),\ldots,\tilde{c}_q(k_q)$, to create the strings: $st_{\rm ind}$, as in (3), and $st_{\rm cf}$, and $st_{\rm sg}$ as in (4) and (5), respectively. All these strings are encoded, separately, using adaptive arithmetic coding.

Decoding

- Reverse the arithmetic coding to recover strings $st_{\rm ind}, st_{\rm cf}, st_{\rm sg}$.
- Invert the quantization step as $|\tilde{c}_q^{\rm r}(n)| = \Delta\tilde{c}_q^{\Delta}(n)$.
- Recover the approximated partition through the liner combination

$$\mathbf{f}_q^{{\rm r},k_q} = \sum_{n=1}^{k_q} \tilde{s}_q(n)|\tilde{c}_q^{\rm r}(n)|\mathbf{d}_{\tilde{\ell}_n^q}.$$

- Assemble the recovered signal

Note: In the numerical examples of the next section, the entropy coding step was realized by the MATLAB functions Arith06 available on [6].

The quality of the recovered signal is assessed by the Signal-to-Noise-Ratio (SNR) with respect to the whole signal and also with respect to every frame in the partition snr(q), $q=1,\ldots Q$. Both, the mean value ($\overline{\rm snr}$) and standard deviation (std) of this local quantities are relevant to comparison of point-wise quality recovery.

Numerical Example: The comparison of the proposed scheme is realized with respect to signals compressed by the MP3 format at the bit-rates of 128 kilobits per second (kbps), 96 kbps, and 64 kbps. In order to use the SNR and $\overline{\rm snr}$ as measures of quality for comparison, the MP3 signal has to be optimized in relation to those quantities. This is carried out by the following process.

- Shifting: Since MP3 introduces a shift with respect to the original signal, to compute a meaningful SNR the samples should be shifted back. Denoting by $\mathbf{f}_{\rm M}$ the numerical signal retrieved from the MP3 file, the optimal time shift $\hat{\tau}$ is determined as that maximizing the cross-correlation with the original signal, i.e.

$$\hat{\tau} = \underset{\tau=-N/2,\ldots,N/2}{\arg\max} \sum_{n=1}^{N} f(n) f_{\rm M}(n+\tau).$$

- Scaling: In order to neutralize the effect of any multiplicative and/or additive constant, we allow for such two constants and adjust them to maximize the SNR as follows: Denoting by $\hat{\mathbf{f}}_{\rm M}$ the MP3 signal after the shifting operation, we consider the linear form $a\hat{\mathbf{f}}_{\rm M} + b$ and fix the values of a and b for which $\|\mathbf{f} - (a\hat{\mathbf{f}}_{\rm M} + b)\|^2$ takes the minimum value.

The test clips, originally in WAV format and sampled at 44100 Hz, are listed in Table 1. All are short clips, the mean value length is 7 s , which are partitioned into frames of length $N_b = 1024$. The upper half of the table corresponds to single instruments and the lower half to orchestrated clips. Since MP3 produces a $\overline{\text{snr}}$ which is in general higher than the total SNR, we compare the compression rate to produce the same $\overline{\text{snr}}$ as MP3. The order in which the clips appear in the tables is according to the following criterion: The first ten solo instrument clips are listed in ascending order of compression rate by the SA approach. The last ten orchestrated clips are also listed in ascending order of compression rate by the same approach. The second column displays the MP3 compression rate, fixed to be 128 kbps for all the clips. The third column is the compression rate achieved by compressing the SA parameters which retrieve a signal producing the same mean value of the local snr as MP3 ($\overline{\text{snr}}_1$) and listed in the fourth column (the fifth column gives the corresponding standard deviation). The sixth and seventh columns are the values of mean local snr and standard deviation corresponding to the SA approach. The last two columns show the corresponding global SNR values.

Table 2 and 3 have an equivalent description to Table 1, but the MP3 compression rate is fixed to be 96 kbps and 64 kbps, respectively. Notice that the order of the clips varies from one to another table.

Table 1: The clips in the first column are: Classic Guitar (C1), Chopin Piano (C2), Bach Piano (C3), Pop Piano (C4), Rock Piano (C5), Harmonics Guitar (C6) Rock Ballad (C7), Medieval(C8), Oboe (C9), Electric Guitar (C10), Sextet (C11), Orchestra Horns (C12), Violins (C13), Orchestrated (14), Quartet (15), Bach Fugue (16), Classic Orchestra(C17), Opera Male (18), Opera Female (19), and Orchestra (20). cr_1 and cr_2 are the compression rate for the MP3 and SA files, respectively. $\overline{\text{snr}}_1$ and SNR_1 stand for the corresponding MP3 values and $\overline{\text{snr}}_2$ and SNR_2 for the SA ones.

Clip	cr_1	cr_2	$\overline{\text{snr}}_1$	std	$\overline{\text{snr}}_2$	std	SNR_1	SNR_2
C1	128	59	50.1	5.2	50.1	6.4	50.1	54.6
C2	128	59	43.5	6.7	43.5	5.9	48.1	49.9
C3	128	76	53.7	4.6	53.7	3.2	48.7	55.0
C4	128	83	52.3	4.8	52.3	6.7	52.1	56.0
C5	128	90	47.7	5.9	47.7	6.3	40.7	50.9
C6	128	92	50.5	5.9	50.5	4.2	47.5	52.6
C7	128	102	48.2	3.6	48.2	1.3	44.3	48.5
C8	128	102	57.9	4.4	57.9	4.8	57.4	59.6
C9	128	109	56.1	3.2	56.1	3.2	56.2	57.0
C10	128	111	44.0	6.3	44.0	2.6	37.8	45.0
C11	128	91	40.9	4.0	41.0	4.0	39.0	42.5
C12	128	94	54.4	4.5	54.4	4.8	53.0	56.5
C13	128	95	45.1	2.9	45.1	3.3	44.7	47.0
C14	128	95	55.9	5.0	56.1	6.5	51.7	60.9
C15	128	96	41.5	6.3	41.5	5.4	35.0	45.4
C16	128	96	43.8	4.7	43.8	2.4	41.6	44.3
C17	128	98	40.2	3.1	40.2	4.8	40.0	42.4
C18	128	106	40.3	4.3	40.3	2.9	36.5	41.0
C19	128	110	37.2	5.7	37.1	4.6	32.1	39.1
C20	128	111	40.2	3.5	40.2	2.7	40.0	40.9

Conclusions: Sparse approximation of music signals is relevant, in its own right, to music information retrieval and auditory tasks. This work demonstrates that, by means of a simple encoding scheme, sparse approximation of melodic music can be stored in a file of competitive size with respect to the MP3 compression standard. Actually, for the set of 20 test clips which have been approximated, the overall gain in compression rate, with respect to the 128 kbps corresponding to MP3, is 27%. The gain increases up to 30% with respect to the 96 kbps produced by MP3, and it further increases up to 47% with respect to the 64 kbps of MP3. The latter improvement is in line with the fact that for 64 kbps the reconstruction quality corresponds to values of SNR in the rage of (30dB, 34dB), for most solo instruments, and of (20dB, 30dB) for most orchestrated clips. These are the values of SNR for which the trigonometric dictionary considered here had been previously shown to render approximations of high sparsity [5].

Note: The scripts and MATLAB functions for reproducing the Tables 1, 2, and 3, can be downloaded on [7]. The original clips, as well as the compressed and recovered files have been made available on that site.

Table 2: Equivalent description to Table 1 but fixing the compression ration of MP3 equal to 96 kbps.

Clip	cr_1	cr_2	$\overline{\text{snr}}_1$	std	$\overline{\text{snr}}_2$	std	SNR_1	SNR_2
C1	96	49	47.3	4.7	47.3	6	45.7	51.6
C9	96	49	45.6	5.4	45.6	3.2	39.8	45.6
C2	96	53	40.6	5.8	40.6	5.7	44.1	43.3
C3	96	63	48.3	5.1	48.3	3.1	45.0	49.5
C5	96	65	39.6	5.1	39.6	5.7	39.6	42.7
C4	96	67	47.4	4.9	47.4	5.9	44.4	51.5
C10	96	73	35.7	6.3	35.7	2.7	29.7	36.7
C6	96	75	44.6	5.7	44.6	4.1	42.8	46.7
C7	96	77	36.7	4.0	40.0	1.4	36.7	40.3
C8	96	82	52.9	4.2	52.9	4.2	53.3	54.7
C12	96	64	47.6	4.5	47.6	4.6	46.1	49.5
C11	96	65	36.2	3.6	36.2	3.8	34.7	37.6
C17	96	65	35.7	3.3	35.9	4.3	35.7	38.0
C13	96	67	40.6	3.0	40.6	3.1	39.7	42.3
C14	96	68	50.0	5.4	50.0	6.1	44.9	54.4
C18	96	70	34.7	4.2	34.7	3.0	31.3	35.3
C15	96	70	35.9	5.9	35.9	5.4	30.0	39.7
C16	96	74	36.4	3.7	36.5	2.4	34.9	36.9
C20	96	74	36.4	3.7	36.4	2.4	34.9	36.9
C19	96	79	31.7	5.5	31.7	4.5	27.1	33.6

Table 3: Equivalent description to Table 1 but fixing the compression ration of MP3 equal to 64 kbps.

Clip	cr_1	cr_2	$\overline{\text{snr}}_1$	std	$\overline{\text{snr}}_2$	std	SNR_1	SNR_2
C9	64	12	29.5	4.0	29.5	3.6	28.1	30.3
C2	64	23	30.7	4.7	30.8	6.9	30.3	36.8
C8	64	27	34.3	5.3	34.4	4.5	32.1	36.2
C1	64	28	39.2	6.5	39.2	5.6	32.5	43.3
C3	64	37	37.0	5.2	37.0	3.1	34.5	38.1
C4	64	38	34.4	4.6	34.4	5.7	31.3	38.1
C10	64	38	25.3	2.4	25.4	2.3	23.1	26.1
C5	64	40	30.7	3.6	30.7	5.6	28.1	33.8
C7	64	46	29.0	3.0	29.0	1.3	27.6	29.3
C6	64	50	34.1	4.7	34.1	4.0	33.0	36.2
C12	64	25	33.5	4.6	33.5	4.6	32.0	35.3
C13	64	25	28.2	3.1	28.2	2.8	26.8	30.1
C17	64	26	25.4	3.0	25.4	4.3	24.7	27.6
C11	64	28	24.1	2.4	24.1	3.7	23.4	25.6
C14	64	29	36.1	6.4	36.1	5.9	31.3	40.5
C20	64	32	26.0	2.7	26.0	2.7	25.7	26.7
C18	64	32	23.6	3.1	23.6	2.8	22.2	24.1
C19	64	39	23.0	4.6	23.0	4.4	19.7	24.5
C15	64	43	27.4	4.3	27.4	5.3	24.3	31.1
C16	64	46	25.9	3.0	25.9	2.2	25.2	26.3

Acknowledgements

The authors are grateful to Karl Skretting for making available the Arith06 function [6]. The test clips for the numerical examples are from `free-loops.com` and sample WAV files on `onclassical.com`

References

1 Y. Vaizman, B. McFee, and G. Lanckriet, "Codebook-based audio feature representation for music information retrieval", *IEEE/ACM Transactions on Audio, Speech and Language Processing*, **22**, 1483–1493 (2014).

2 Y. Han, S. Lee, J. Nam, and K. Lee, "Sparse feature learning for instrument identification: Effects of sampling and pooling methods", *The Journal of the Acoustical Society of America* **139**, 2290–2298 (2016).

3 H. Henaff , K. Jarrett , K. Kavukcuoglu , and Y. LeCun , "Unsupervised learning of sparse features for scalable audio classification", in Proceedings of the 15th International Society for Music Information Retrieval Conference (2014), 77–82.

4 L. Rebollo-Neira, "Cooperative Greedy Pursuit Strategies for Sparse Signal Representation by Partitioning", *Signal Processing,* **125**, 365–375 (2016).

5 L. Rebollo-Neira and G. Aggarwal, "A dedicated greedy pursuit algorithm for sparse spectral modelling of music sound", *Journal of The Acoustic Society of America*, **140,** 2933–2943 (2016).

6 `http://www.ux.uis.no/~karlsk/proj99`

7 `http://www.nonlinear-approx.info/examples/node07.html`