Efficient Implementation of the K-SVD Algorithm using Batch Orthogonal Matching Pursuit

Ron Rubinstein[*], Michael Zibulevsky[*] and Michael Elad[*]

Abstract

The K-SVD algorithm is a highly effective method of training overcomplete dictionaries for sparse signal representation. In this report we discuss an efficient implementation of this algorithm, which both accelerates it and reduces its memory consumption. The two basic components of our implementation are the replacement of the exact SVD computation with a much quicker approximation, and the use of the *Batch-OMP* method for performing the sparse-coding operations.

Batch-OMP, which we also present in this report, is an implementation of the Orthogonal Matching Pursuit (OMP) algorithm which is specifically optimized for sparse-coding large sets of signals over the same dictionary. The Batch-OMP implementation is useful for a variety of sparsity-based techniques which involve coding large numbers of signals. In the report, we discuss the Batch-OMP and K-SVD implementations and analyze their complexities. The report is accompanied by Matlab® toolboxes which implement these techniques, and can be downloaded at `http://www.cs.technion.ac.il/~ronrubin/software.html`.

1 Introduction

Sparsity in overcomplete dictionaries is the basis for a wide variety of highly effective signal and image processing techniques. The basic model suggests that natural signals can be efficiently explained as linear combinations of prespecified *atom signals*, where the linear coefficients are sparse (most of them zero). Formally, if $\underline{x}$ is a column signal and $\mathbf{D}$ is the *dictionary* (whose columns are the atom signals), the sparsity assumption can be described by the following *sparse approximation* problem,

$$\hat{\underline{\gamma}} = \underset{\underline{\gamma}}{\text{Argmin}} \; \|\underline{\gamma}\|_0 \quad \text{Subject To} \quad \|\underline{x} - \mathbf{D}\underline{\gamma}\|_2^2 \le \epsilon \; . \tag{1.1}$$

In this formulation, $\underline{\gamma}$ is the *sparse representation* of $\underline{x}$, ϵ the error tolerance, and $\|\cdot\|_0$ is the ℓ^0 pseudo-norm which counts the non-zero entries. The sparse approximation problem, which is known to be NP-hard, can be efficiently solved using several available

[*]Computer Science Department, Technion – Israel Institute of Technology, Haifa 32000 Israel.

approximation techniques, including Orthogonal Matching Pursuit (OMP) [1, 2], Basis Pursuit (BP) [3, 4], FOCUSS [5], and others.

A fundamental question in the above formulation is the choice of the dictionary. The dictionary will commonly emerge from one of two sources — either a predesigned transform, such as the Wavelet [6], Curvelet [7], or Contourlet [8] transforms, or from example data, using some adaptive training process. The *K-SVD algorithm* [9] is such a technique for training a dictionary from given example data. It is a highly effective method, and is successfully applied in several image processing tasks [10, 11, 12, 13, 14].

The K-SVD algorithm is quite computationally demanding, however, especially when the dimensions of the dictionary increase or the number of training signals becomes large. In this report we discuss an efficient implementation of the algorithm which reduces its complexity as well as its memory requirements. The improvements are achieved by using a modified dictionary update step which replaces the explicit SVD computation with a simpler approximation, and employing an optimized OMP implementation which accelerates the sparse-coding step.

We discuss our optimized OMP implementation in Section 2. The implementation is motivated by the observation that sparsity-based techniques often involve the coding of a *large number* of signals over *the same* dictionary. In a common image processing task, for instance, the image could be decomposed into thousands of (possibly overlapping) image patches, with each patch undergoing one or more sparse-coding processes ([10, 11, 12, 13, 14] and others). In our context, algorithms such as the K-SVD method commonly use tens or even hundreds of thousands of signals for the learning process, in order to reduce overfitting. Our implementation, which we term *Batch-OMP*, is specifically suited for sparse-coding large sets of signals over a single dictionary, and it combines existing techniques for OMP acceleration with a new contribution to allow its use with error-driven signal processing tasks.

In Section 3 we review the K-SVD algorithm and discuss its efficient implementation. Our *Approximate K-SVD* method uses a simple approximation of the SVD computation to obtain a faster and more memory-efficient method. We show that the complexity of the resulting implementation is dominated by its sparse-coding step, and thus employ Batch-OMP to obtain further acceleration.

1.1 Notation

This report uses the following notation in the algorithm descriptions and mathematical propositions:

- Bold uppercase letters designate matrices ($\mathbf{M}$, $\mathbf{\Gamma}$), while underlined lowercase letters designate column vectors ($\underline{v}$, $\underline{\gamma}$). The columns of a matrix are referenced using the corresponding lowercase letter, e.g. $\mathbf{M} = [\,\underline{m}_1 \,|\, \dots \,|\, \underline{m}_n\,]$ or $\mathbf{\Gamma} = [\,\underline{\gamma}_1 \,|\, \dots \,|\, \underline{\gamma}_m\,]$; the coefficients of a vector are similarly referenced using non-underlined letters, e.g. $\underline{v} = (v_1, \dots, v_n)^T$. The notation $\underline{0}$ is used to denote the zero vector, with its length deduced from the context.

- Given a single index $I = i_1$ or an ordered sequence of indices $I = (i_1, \ldots, i_k)$, we denote by $\mathbf{M}_I = [\, \underline{m}_{i_1} \,|\, \ldots \,|\, \underline{m}_{i_k} \,]$ the sub-matrix of $\mathbf{M}$ containing the columns indexed by I, *in the order in which they appear in* I. For vectors we similarly denote the sub-vector $\underline{v}_I = (v_{i_1}, \ldots, v_{i_k})^T$. We use the notation $\mathbf{M}_{I,J}$, with J a second index or sequence of indices, to refer to the sub-matrix of $\mathbf{M}$ containing the rows indexed by I and the columns indexed by J, in their respective orders. The indexed notations are used in algorithm descriptions for both *access* and *assignment*, so if $I = (2, 4, 6, \ldots, n)$, for instance, the statement $\mathbf{M}_{I,j} := \underline{0}$ means nullifying the even-indexed entries in the j-th row of $\mathbf{M}$.

2 OMP and Batch-OMP

2.1 The OMP Algorithm

The OMP algorithm aims to approximate the solution of one of the two following problems, the *sparsity-constrained* sparse coding problem, given by

$$\hat{\underline{\gamma}} \; = \; \underset{\underline{\gamma}}{\text{Argmin}} \; \|\underline{x} - \mathbf{D}\underline{\gamma}\|_2^2 \quad \text{Subject To} \quad \|\underline{\gamma}\|_0 \leq K \; , \tag{2.1}$$

and the *error-constrained* sparse coding problem, given by

$$\hat{\underline{\gamma}} \; = \; \underset{\underline{\gamma}}{\text{Argmin}} \; \|\underline{\gamma}\|_0 \quad \text{Subject To} \quad \|\underline{x} - \mathbf{D}\underline{\gamma}\|_2^2 \leq \epsilon \; . \tag{2.2}$$

For simplicity, we assume hereon that the columns of $\mathbf{D}$ are normalized to unit ℓ^2-length (though this restriction may easily be removed).

The greedy OMP algorithm selects at each step the atom with the highest correlation to the current residual. Once the atom is selected, the signal is orthogonally projected to the span of the selected atoms, the residual is recomputed, and the process repeats (see *Algorithm 1*). Note that line 5 is the greedy selection step, and line 7 is the orthogonalization step.

The computation in line 7 will usually not be carried out explicitly due to its high cost. A practical implementation employs a progressive Cholesky or QR update process [15, 16] to reduce the work involved in the matrix inversion. In a nutshell, the computation

$$\begin{aligned} \underline{\gamma}_I &= (\mathbf{D}_I)^+ \underline{x} \\ &= (\mathbf{D}_I^T \mathbf{D}_I)^{-1} \mathbf{D}_I^T \underline{x} \end{aligned}$$

requires the inversion of the matrix $\mathbf{D}_I^T \mathbf{D}_I$, which remains non-singular due to the orthogonalization process which ensures the selection of linearly independent atoms. The matrix $\mathbf{D}_I^T \mathbf{D}_I$ is an SPD (symmetric positive-definite) matrix which is updated every iteration by simply appending a single row & column to it, and therefore its Cholesky factorization requires only the computation of its last row. It is readily verified that

Algorithm 1 ORTHOGONAL MATCHING PURSUIT

```
1: Input: Dictionary $\mathbf{D}$, signal $\underline{x}$, target sparsity $K$ *or* target error $\epsilon$
2: Output: Sparse representation $\underline{\gamma}$ such that $\underline{x} \approx \mathbf{D}\underline{\gamma}$
3: Init: Set $I := (\,)$, $\underline{r} := \underline{x}$, $\underline{\gamma} := \underline{0}$
4: while (stopping criterion not met) do
5:     $\hat{k} := \operatorname{Argmax}_k |\underline{d}_k^T \underline{r}|$
6:     $I := (I, \hat{k})$
7:     $\underline{\gamma}_I := (\mathbf{D}_I)^+ \underline{x}$
8:     $\underline{r} := \underline{x} - \mathbf{D}_I \underline{\gamma}_I$
9: end while
```

given the Cholesky factorization of $\tilde{\mathbf{A}} = \tilde{\mathbf{L}}\tilde{\mathbf{L}}^T \in \mathbb{R}^{(n-1)\times(n-1)}$, the Cholesky factorization of

$$\mathbf{A} = \begin{pmatrix} \tilde{\mathbf{A}} & \underline{v} \\ \underline{v}^T & c \end{pmatrix} \in \mathbb{R}^{n\times n} \tag{2.3}$$

is given by $\mathbf{A} = \mathbf{L}\mathbf{L}^T$, with

$$\mathbf{L} = \begin{pmatrix} \tilde{\mathbf{L}} & \underline{0} \\ \underline{w}^T & \sqrt{c - \underline{w}^T\underline{w}} \end{pmatrix} \quad , \quad \underline{w} = \tilde{\mathbf{L}}^{-1}\underline{v} \; . \tag{2.4}$$

See *Algorithm 2* for the full OMP-Cholesky implementation.

2.2 Handling Large Sets of Signals

When large numbers of signals must be coded over the same dictionary, it is worthwhile to consider pre-computation to reduce the total amount of work involved in coding the entire set. This approach has been previously discussed in [1], and our Batch-OMP implementation adopts this technique, extending it to the error-constrained case.

The key observation is that the atom selection step at each iteration does not require knowing $\underline{r}$ or $\underline{\gamma}$ explicitly, but only $\mathbf{D}^T\underline{r}$. The idea is therefore to replace the explicit computation of $\underline{r}$ and its multiplication by $\mathbf{D}^T$ with a lower-cost computation of $\mathbf{D}^T\underline{r}$. Denoting $\underline{\alpha} = \mathbf{D}^T\underline{r}$, $\underline{\alpha}^0 = \mathbf{D}^T\underline{x}$, and $\mathbf{G} = \mathbf{D}^T\mathbf{D}$, we can write:

$$\begin{aligned} \underline{\alpha} &= \mathbf{D}^T\left(\underline{x} - \mathbf{D}_I(\mathbf{D}_I)^+\underline{x}\right) \\ &= \underline{\alpha}^0 - \mathbf{G}_I(\mathbf{D}_I)^+\underline{x} \\ &= \underline{\alpha}^0 - \mathbf{G}_I(\mathbf{D}_I^T\mathbf{D}_I)^{-1}\mathbf{D}_I^T\underline{x} \\ &= \underline{\alpha}^0 - \mathbf{G}_I(\mathbf{G}_{I,I})^{-1}\underline{\alpha}_I^0 \; . \end{aligned} \tag{2.5}$$

This means that given the pre-computed $\underline{\alpha}^0$ and $\mathbf{G}$, we can compute $\underline{\alpha}$ each iteration without explicitly computing $\underline{r}$. The modified update step requires only multiplication

Algorithm 2 OMP-Cholesky

```
1: Input: Dictionary $\mathbf{D}$, signal $\underline{x}$, target sparsity $K$ *or* target error $\epsilon$
2: Output: Sparse representation $\underline{\gamma}$ such that $\underline{x} \approx \mathbf{D}\underline{\gamma}$
3: Init: Set $I := (\,)$, $\mathbf{L} := [1]$, $\underline{r} := \underline{x}$, $\underline{\gamma} := \underline{0}$, $\underline{\alpha} := \mathbf{D}^T\underline{x}$, $n := 1$
4: while (stopping criterion not met) do
5:     $\hat{k} := \underset{k}{\text{Argmax}} \, |\underline{d}_k^T \underline{r}|$
6:     if $n > 1$ then
7:         $\underline{w} :=$ Solve for $\underline{w}$ $\{\mathbf{L}\underline{w} = \mathbf{D}_I^T \underline{d}_{\hat{k}}\}$
8:         $\mathbf{L} := \begin{bmatrix} \mathbf{L} & \underline{0} \\ \underline{w}^T & \sqrt{1-\underline{w}^T\underline{w}} \end{bmatrix}$
9:     end if
10:    $I := (I, \hat{k})$
11:    $\underline{\gamma}_I :=$ Solve for $\underline{c}$ $\{\mathbf{L}\mathbf{L}^T\underline{c} = \underline{\alpha}_I\}$
12:    $\underline{r} := \underline{x} - \mathbf{D}_I\underline{\gamma}_I$
13:    $n := n + 1$
14: end while
```

by the matrix $\mathbf{G}_I$ instead of applying the complete dictionary $\mathbf{D}^T$. Note that the matrix $\mathbf{G}_{I,I}$ is inverted using the progressive Cholesky factorization discussed above.

The limitation of this approach is that since the residual is never explicitly computed, an error-based stopping criterion (which is required for most practical cases) becomes challenging to employ. In the following, we extend this method to the error-driven case by deriving an efficient incremental formula for the ℓ^2 error. This makes the accelerated OMP implementation useful for the many applications where error-constrained coding is required.

We denote by $\underline{r}^n$ and $\underline{\gamma}^n$ the residual and the sparse approximation, respectively, at the end of the n-th iteration. We can now write

$$\begin{aligned} \underline{r}^n &= \underline{x} - \mathbf{D}\underline{\gamma}^n \\ &= \underline{x} - \mathbf{D}\underline{\gamma}^{n-1} + \mathbf{D}\underline{\gamma}^{n-1} - \mathbf{D}\underline{\gamma}^n \\ &= \underline{r}^{n-1} + \mathbf{D}(\underline{\gamma}^{n-1} - \underline{\gamma}^n) \ . \end{aligned} \tag{2.6}$$

The orthogonalization process in OMP ensures that at each iteration, the residual is orthogonal to the current signal approximation. We thus have for all n,

$$(\underline{r}^n)^T \mathbf{D}\underline{\gamma}^n = 0 \ . \tag{2.7}$$

Using expression (2.6), and plugging-in property (2.7), we obtain the following expansion for the squared approximation error:

$$
\begin{aligned}
\|\underline{r}^n\|_2^2 &= (\underline{r}^n)^T\underline{r}^n \;=\; (\underline{r}^n)^T\left(\underline{r}^{n-1} + \mathbf{D}(\underline{\gamma}^{n-1} - \underline{\gamma}^n)\right) \\
&= (\underline{r}^n)^T\underline{r}^{n-1} + (\underline{r}^n)^T\mathbf{D}\underline{\gamma}^{n-1} \\
&= \left(\underline{r}^{n-1} + \mathbf{D}(\underline{\gamma}^{n-1} - \underline{\gamma}^n)\right)^T\underline{r}^{n-1} + (\underline{r}^n)^T\mathbf{D}\underline{\gamma}^{n-1} \\
&= \|\underline{r}^{n-1}\|_2^2 - (\underline{r}^{n-1})^T\mathbf{D}\underline{\gamma}^n + (\underline{r}^n)^T\mathbf{D}\underline{\gamma}^{n-1} \\
&= \|\underline{r}^{n-1}\|_2^2 - (\underline{x} - \mathbf{D}\underline{\gamma}^{n-1})^T\mathbf{D}\underline{\gamma}^n + (\underline{x} - \mathbf{D}\underline{\gamma}^n)^T\mathbf{D}\underline{\gamma}^{n-1} \\
&= \|\underline{r}^{n-1}\|_2^2 - \underline{x}^T\mathbf{D}\underline{\gamma}^n + \underline{x}^T\mathbf{D}\underline{\gamma}^{n-1} \\
&= \|\underline{r}^{n-1}\|_2^2 - (\underline{r}^n + \mathbf{D}\underline{\gamma}^n)^T\mathbf{D}\underline{\gamma}^n + (\underline{r}^{n-1} + \mathbf{D}\underline{\gamma}^{n-1})^T\mathbf{D}\underline{\gamma}^{n-1} \\
&= \|\underline{r}^{n-1}\|_2^2 - (\underline{\gamma}^n)^T\mathbf{D}^T\mathbf{D}\underline{\gamma}^n + (\underline{\gamma}^{n-1})^T\mathbf{D}^T\mathbf{D}\underline{\gamma}^{n-1} \\
&= \|\underline{r}^{n-1}\|_2^2 - (\underline{\gamma}^n)^T\mathbf{G}\underline{\gamma}^n + (\underline{\gamma}^{n-1})^T\mathbf{G}\underline{\gamma}^{n-1}\;. \qquad (2.8)
\end{aligned}
$$

For simplicity, we designate the squared approximation error by $\epsilon^n = \|\underline{r}^n\|_2^2$. We also introduce the notation $\delta^n = (\underline{\gamma}^n)^T\mathbf{G}\underline{\gamma}^n$. The error update step is thus given by the simple formula

$$\epsilon^n = \epsilon^{n-1} - \delta^n + \delta^{n-1}\;. \qquad (2.9)$$

Note that the computation of δ^n each iteration is extremely cheap, as the product $\mathbf{G}\underline{\gamma}^n = \mathbf{G}_I\underline{\gamma}^n_I$ is just the product $\mathbf{G}_I(\mathbf{G}_{I,I})^{-1}\underline{\alpha}^0_I$, which is computed in any event for the update of $\underline{\alpha}$. Therefore, the only added work in computing δ^n is the dot product between this vector and the sparse vector $\underline{\gamma}^n$, which requires a negligible amount of work.

The complete error-constrained algorithm is presented as *Algorithm 3*. For convenience, Table 1 summarizes the symbols used in the algorithm. We name this implementation *Batch-OMP* as it is specifically designed for sparse coding large sets of signals — indeed, the precomputation of $\mathbf{G}$ limits its usability for coding small numbers of signals.

I	Indices of the selected atoms (ordered sequence)
$\underline{\gamma}$	The sparse representation of $\underline{x}$
$\underline{r}$	The residual $(\underline{x} - \mathbf{D}\underline{\gamma})$
$\underline{\alpha}^0$	The product $\mathbf{D}^T\underline{x}$
$\underline{\alpha}$	The product $\mathbf{D}^T\underline{r}$
$\underline{\beta}$	The product $\mathbf{G}\underline{\gamma}$
$\mathbf{L}$	The Cholesky factorization of $\mathbf{G}_{I,I}$
δ^n	The weighted norm $\underline{\gamma}^T\mathbf{G}\underline{\gamma}$
ϵ^n	The squared error $\|\underline{r}\|_2^2$

Table 1: Values computed in the Batch-OMP algorithm. The superscript $n = 0, 1, 2, \ldots$ (where used) indicates the iteration number: we define a symbol's value at the n-th iteration as its value at the *end* of the n-th iteration, and use the superscript $n = 0$ to refer to its *initial* value.

Algorithm 3 BATCH-OMP

```
1: Input: $\underline{\alpha}^0 = \mathbf{D}^T\underline{x}$, $\epsilon^0 = \underline{x}^T\underline{x}$, $\mathbf{G} = \mathbf{D}^T\mathbf{D}$, target error $\epsilon$
2: Output: Sparse representation $\underline{\gamma}$ such that $\underline{x} \approx \mathbf{D}\underline{\gamma}$
3: Init: Set $I := ()$, $\mathbf{L} := [1]$, $\underline{\gamma} := \underline{0}$, $\underline{\alpha} := \underline{\alpha}^0$, $\delta^0 := 0$, $n := 1$
4: while $\epsilon^{n-1} > \epsilon$ do
5:   $\hat{k} := \underset{k}{\text{Argmax}} \{|\underline{\alpha}_k|\}$
6:   if $n > 1$ then
7:     $\underline{w}$ := Solve for $\underline{w}$ $\left\{ \mathbf{L}\underline{w} = \mathbf{G}_{I,\hat{k}} \right\}$
8:     $\mathbf{L} := \begin{bmatrix} \mathbf{L} & \underline{0} \\ \underline{w}^T & \sqrt{1 - \underline{w}^T\underline{w}} \end{bmatrix}$
9:   end if
10:  $I := (I, \hat{k})$
11:  $\underline{\gamma}_I$ := Solve for $\underline{c}$ $\left\{ \mathbf{L}\mathbf{L}^T\underline{c} = \underline{\alpha}^0_I \right\}$
12:  $\underline{\beta} = \mathbf{G}_I\underline{\gamma}_I$
13:  $\underline{\alpha} := \underline{\alpha}^0 - \underline{\beta}$
14:  $\delta^n = \underline{\gamma}_I^T\underline{\beta}_I$
15:  $\epsilon^n = \epsilon^{n-1} - \delta^n + \delta^{n-1}$
16:  $n := n + 1$
17: end while
```

Note that for the sparsity-constrained version, we simply remove the updates of δ^n and ϵ^n and replace the stopping criterion with the appropriate one.

2.3 Complexity Analysis

We now consider the complexity of these OMP implementations, focusing on the gain offered by precomputing $\mathbf{G}$ when coding large sets of signals. For the analysis we assume a signal $\underline{x} \in \mathbb{R}^N$ and a dictionary $\mathbf{D} \in \mathbb{R}^{N \times L}$. We distinguish between two types of dictionaries — explicit dictionaries which are stored in memory in their entirety (such as in the K-SVD case), and implicit dictionaries, which are specified through their forward and adjoint operators (and may not provide access to individual atoms). We denote the complexity of applying $\mathbf{D}$ and $\mathbf{D}^T$ as T_D ; for explicit dictionaries, we have $T_D = 2NL$.

The following analysis assumes that a triangular $n \times n$ back-substitution process requires n^2 operations. Also, it does not take into account vector or matrix reallocation operations, which can be avoided or optimized when sufficient memory is pre-allocated, or a clever memory management scheme is employed.

We begin with the OMP-Cholesky implementation (Algorithm 2). The n-th iteration includes computing $\mathbf{D}^T\underline{r}$ (T_D operations), taking the absolute-value maximum ($2L$ oper-

Dictionary Type	OMP (Per Signal)	Batch-OMP Per Signal	Batch-OMP T_G
Implicit	$4KT_D + 2K(L+N) + K^3$	$T_D + K^2L + 3KL + K^3$	?
Explicit	$KT_D + 2K^2N + 2K(L+N) + K^3$	$T_D + K^2L + 3KL + K^3$	NL^2
Dictionary size: $N \times L$		Target sparsity: K	

Table 2: Complexity of standard OMP versus Batch-OMP. The Batch-OMP complexity is divided into the precomputation complexity (T_G) and the complexity per signal.

ations), computing $\underline{w}$ (n^2 operations for the back-substitution, plus the computation of $\mathbf{D}_I^T \underline{d}_{\hat{k}}$), computing the new representation coefficients (two additional back-substitution processes, or $2n^2$ operations), and updating the residual, which involves computing $\mathbf{D}_I \underline{\gamma}_I$ and N subtraction operations.

The operations $\mathbf{D}_I^T \underline{d}_{\hat{k}}$ and $\mathbf{D}_I \underline{\gamma}_I$ differ between the two dictionary types. For explicit dictionaries, each can be done in $2nN$ operations using a direct computation. For implicit dictionaries, each matrix multiplication involves a full applications of the dictionary, and computing $\underline{d}_{\hat{k}}$ requires yet another application of the dictionary. Summing over all K iterations, we therefore find that the final OMP-Cholesky runtime is given by

$$\begin{array}{rcl} T_{omp}\,\{\textit{implicit-dict}\} & = & 4KT_D + 2K(L+N) + K^3 \\ T_{omp}\,\{\textit{explicit-dict}\} & = & KT_D + 2K^2N + 2K(L+N) + K^3 \end{array} . \tag{2.10}$$

The analysis of the Batch-OMP algorithm is similar. Assuming $\mathbf{G}$ is precomputed, the algorithm requires a single application of $\mathbf{D}^T$ during initialization (T_D operations), and other than that, the dominant operations at the n-th iteration are the absolute-value maximum ($2L$ operations), computing $\underline{w}$ (n^2 operations), computing the updated coefficients ($2n^2$ operations), computing $\underline{\beta}$ ($2nL$ operations), and updating $\underline{\alpha}$ (L operations). The total number of operations, summing over all K iterations, is therefore

$$T_{b-omp} \;=\; T_D + K^2L + 3KL + K^3 \; . \tag{2.11}$$

Comparing this with (2.10), we see that the number of applications of $\mathbf{D}$ is significantly reduced. This is a substantial improvement when the dictionary is costly to apply, such as when it is stored explicitly.

Table 2 summarizes the complexities of OMP and Batch-OMP for a general and explicit dictionary. For Batch-OMP, the time required to precompute $\mathbf{G}$ (denoted as T_G) is also specified. Note that for an explicit dictionary, this is

$$T_G\{\textit{explicit-dict}\} = NL^2 \; , \tag{2.12}$$

where we have taken into account the symmetry of the matrix.

# of Signals	OMP	Batch-OMP	Ratio
1	2.14	67.4	x0.03
10	21.43	70.2	x0.31
10^2	214.3	97.9	x2.19
10^3	2,142.7	374.8	x5.72
10^5	214,272.0	30,838.3	x6.95

Table 3: Operation count (in millions of operations) of standard OMP versus Batch-OMP for an explicit dictionary. Signal size is 256, dictionary size is 256×512, target sparsity is 8. Batch-OMP time includes precomputation time.

As a concrete example, assume an explicit dictionary with $K = \sqrt{N}/2$ and $L = 2N$. The complexities in (2.10) and (2.11) become

$$\begin{array}{lcl} T_{omp} & \approx & 2N^{2.5} \\ T_{b-omp} & \approx & 4.5N^2 \end{array} . \tag{2.13}$$

We see that the advantage of Batch-OMP over standard OMP is indeed asymptotic. This per-signal advantage quickly overcomes the added cost of precomputing $\mathbf{G}$ ($4N^3$ operations in this case) when the number of signals passes $\sim 2\sqrt{N}$. Table 3 lists the total number of operations required to sparse-code a variable number of signals using both OMP and Batch-OMP for $N = 256$ (which could stand for 16×16 image patches). We note that as the number of signals increases, the computation of $\mathbf{G}$ becomes insignificant and the ratio between the two methods approaches $T_{omp}/T_{b-omp} = (2/4.5)\sqrt{N} = 7.11$.

2.4 Other Acceleration Techniques

With its high popularity, many acceleration techniques have been proposed for the OMP algorithm over the years. In [17], the authors improved the straightforward OMP implementation by exploiting mathematical properties of the orthogonalization process. Their improvements were based on the Modified Matching Pursuit implementation [16], which implements OMP by orthogonalizing and storing the selected atoms as the algorithm advances. The complexity they achieve is similar to that of the progressive Cholesky-based OMP implementation, however it requires more memory and is therefore less useful.

More recently, a clever acceleration scheme has been proposed for the Matching Pursuit algorithm, reducing its complexity to an impressive $O(N \log N)$ per iteration [18]. The acceleration is achieved by assuming a specific spatial locality property on the dictionary atoms, which significantly reduces the number of atom-signal inner-products that must be recomputed between iterations. The authors further employ a tree-structure to accelerate the maximization process in the atom selection step. This approach, named MPTK (MP-ToolKit), is specifically designed for coding very large signals over highly regular dictionaries. Batch-OMP, in contrast, is useful for sparse-coding large *sets* of smaller signals over generic dictionaries. The dictionary in these cases lacks the locality features required by MPTK (due to the size of the signals), and thus it cannot be employed.

A different acceleration approach recently proposed is the Gradient Pursuit algorithm [15], which employs an iterative process in order to more efficiently solve the ℓ^2 orthogonalization problem in OMP. This algorithm is not an exact implementation of OMP, and rather an approximation of it. Like MPTK, it is best suited for sparse-coding very large signals, though it does not impose any specific structure on the dictionary.

3 Efficient K-SVD Implementation

3.1 The K-SVD Algorithm

The K-SVD algorithm accepts an initial overcomplete dictionary $\mathbf{D}_0$, a number of iterations k, and a set of training signals arranged as the columns of the matrix $\mathbf{X}$. The algorithm aims to iteratively improve the dictionary to achieve sparser representations of the signals in $\mathbf{X}$, by solving the optimization problem

$$\underset{\mathbf{D},\mathbf{\Gamma}}{\text{Min}} \; \|\mathbf{X} - \mathbf{D}\mathbf{\Gamma}\|_F^2 \qquad \text{Subject To} \quad \forall i \;\; \|\underline{\gamma}_i\|_0 \le K. \tag{3.1}$$

The K-SVD algorithm involves two basic steps, which together constitute the algorithm iteration: (i) the signals in $\mathbf{X}$ are sparse-coded given the current dictionary estimate, producing the sparse representations matrix $\mathbf{\Gamma}$, and (ii) the dictionary atoms are updated given the current sparse representations; see *Algorithm 4*. The sparse-coding part (line 5) is commonly implemented using OMP. The dictionary update (lines 6-13) is performed one atom at a time, optimizing the target function for each atom individually while keeping the rest fixed.

The main innovation in the algorithm is the atom update step, which is performed while preserving the constraint in (3.1). To achieve this, the update step uses only the signals in $\mathbf{X}$ whose sparse representations use the current atom. Letting I denote the indices of the signals in $\mathbf{X}$ which use the j-th atom, the update is obtained by optimizing the target function

$$\|\mathbf{X}_I - \mathbf{D}\mathbf{\Gamma}_I\|_F^2 \tag{3.2}$$

over both the atom and its associated coefficient row in $\mathbf{\Gamma}_I$. The resulting problem is a simple rank-1 approximation task given by

$$\{\, \underline{d}\,, \underline{g}\,\} := \underset{\underline{d},\underline{g}}{\text{Argmin}} \; \|\mathbf{E} - \underline{d}\,\underline{g}^T\|_F^2 \quad \text{Subject To} \quad \|\underline{d}\|_2 = 1\ , \tag{3.3}$$

where $\mathbf{E} = \mathbf{X}_I - \sum_{i\ne j} \underline{d}_i \mathbf{\Gamma}_{i,I}$ is the error matrix without the j-th atom, $\underline{d}$ is the updated atom, and $\underline{g}^T$ is the new coefficients row in $\mathbf{\Gamma}_I$. The problem can be solved directly via an SVD decomposition, or more efficiently using some numeric power method.

Algorithm 4 K-SVD

```
1: Input: Signal set $\mathbf{X}$, initial dictionary $\mathbf{D}_0$, target sparsity $K$, number of iterations $k$.
2: Output: Dictionary $\mathbf{D}$ and sparse matrix $\mathbf{\Gamma}$ such that $\mathbf{X} \approx \mathbf{D\Gamma}$
3: Init: Set $\mathbf{D} := \mathbf{D}_0$
4: for $n = 1 \dots k$ do
5:   $\forall i: \quad \mathbf{\Gamma}_i := \underset{\underline{\gamma}}{\text{Argmin}} \, \|\underline{x}_i - \mathbf{D}\underline{\gamma}\|_2^2 \quad \text{Subject To} \quad \|\underline{\gamma}\|_0 \le K$
6:   for $j = 1 \dots L$ do
7:     $\mathbf{D}_j := \underline{0}$
8:     $I := \{$indices of the signals in $\mathbf{X}$ whose representations use $\underline{d}_j\}$
9:     $\mathbf{E} := \mathbf{X}_I - \mathbf{D\Gamma}_I$
10:    $\{\underline{d}, \underline{g}\} := \underset{\underline{d},\underline{g}}{\text{Argmin}} \, \|\mathbf{E} - \underline{d}\,\underline{g}^T\|_F^2 \quad \text{Subject To} \quad \|\underline{d}\|_2 = 1$
11:    $\mathbf{D}_j := \underline{d}$
12:    $\mathbf{\Gamma}_{j,I} := \underline{g}^T$
13:  end for
14: end for
```

3.2 Approximate K-SVD

In practice, the exact solution of (3.3) can be computationally difficult, as the size of $\mathbf{E}$ is proportional to the number of training signals. Fortunately, an exact solver is not usually required here. Indeed, the entire K-SVD algorithm only converges to a *local* minimum and not a global one, and respectively, its analysis as provided in [9] only assumes a *reduction* of the target function value in (3.3), *not* an optimal solution. Put differently, the goal of K-SVD is really to *improve* a given initial dictionary, not find an optimal one. Therefore, a much quicker approach is to use an approximate solution of (3.3) rather than the exact one — just as long as this approximation ensures a reduction of the final target function.

Our implementation uses a single iteration of alternate-optimization over the atom $\underline{d}$ and the coefficients row $\underline{g}^T$, which is given by

$$\begin{array}{rcl} \underline{d} & := & \mathbf{E}\underline{g}/\|\mathbf{E}\underline{g}\|_2 \\ \underline{g} & := & \mathbf{E}^T\underline{d} \end{array} . \tag{3.4}$$

This process is known to ultimately converge to the optimum, and when truncated, supplies an approximation which still reduces the penalty term. Our experience shows that a single iteration of this process is generally sufficient to provide very close results to the full computation.

A significant advantage of this method is that it eliminates the need to explicitly compute the matrix $\mathbf{E}$. This computation is both time and memory consuming, and in the approximate formulation is avoided by computing only the products of this matrix with vectors. The complete Approximate K-SVD implementation is given as *Algorithm 5*.

Algorithm 5 APPROXIMATE K-SVD

```
1: Input: Signal set $\mathbf{X}$, initial dictionary $\mathbf{D}_0$, target sparsity $K$, number of iterations $k$.
2: Output: Dictionary $\mathbf{D}$ and sparse matrix $\mathbf{\Gamma}$ such that $\mathbf{X} \approx \mathbf{D\Gamma}$
3: Init: Set $\mathbf{D} := \mathbf{D}_0$
4: for $n = 1 \dots k$ do
5:   $\forall i: \ \mathbf{\Gamma}_i := \underset{\underline{\gamma}}{\mathrm{Argmin}} \|\underline{x}_i - \mathbf{D}\underline{\gamma}\|_2^2$  Subject To  $\|\underline{\gamma}\|_0 \leq K$
6:   for $j = 1 \dots L$ do
7:     $\mathbf{D}_j := \underline{0}$
8:     $I := \{$indices of the signals in $\boldsymbol{X}$ whose representations use $\underline{d}_j\}$
9:     $\underline{g} := \mathbf{\Gamma}_{j,I}^T$
10:    $\underline{d} := \mathbf{X}_I\underline{g} - \mathbf{D\Gamma}_I\underline{g}$
11:    $\underline{d} := \underline{d}/\|\underline{d}\|_2$
12:    $\underline{g} := \mathbf{X}_I^T\underline{d} - (\mathbf{D\Gamma}_I)^T\underline{d}$
13:    $\mathbf{D}_j := \underline{d}$
14:    $\mathbf{\Gamma}_{j,I} := \underline{g}^T$
15:  end for
16: end for
```

3.3 Complexity Analysis

The complexity analysis of the Approximate K-SVD algorithm is similar to that of the OMP algorithm in the previous section. We assume that the sparse-coding in line 5 is implemented using Batch-OMP, and hence requires $2NL + K^2L + 3KL + K^3$ operations per training signal, plus NL^2 operations for the precomputation of $\mathbf{G}$.

As to the dictionary update step, the only difficulty with its analysis is that the atom update complexity depends on the number of signals using it (in other words, we do not know the number of columns in $\mathbf{X}_I$ and $\mathbf{\Gamma}_I$). This, however, can be worked-around by performing a *cumulative* analysis over the entire dictionary update loop. Indeed, the *total* number of columns in each of these matrices, summing over all L iterations, is exactly equal to the number of non-zeros in the matrix $\mathbf{\Gamma}$ (as each such non-zero represents one training signal using one atom). The number of non-zeros in $\mathbf{\Gamma}$ is known to be $R \cdot K$, with R the number of training signals, and thus an exact analysis can be carried out.

Considering line 10, for instance, the number of columns in $\mathbf{X}_I$ for a single iteration is unknown; however, the total number of columns in all $\mathbf{X}_I$'s over all L iterations is exactly RK. Therefore, the cumulative number of operations required to perform the multiplications $\mathbf{X}_I\underline{g}$ in all iterations sums up to $2RKN$. Similarly, the multiplications $\mathbf{\Gamma}_I\underline{g}$ sum up to $2RKL$. The multiplication by $\mathbf{D}$ adds $2NL$ operations per atom, which sums up to $2NL^2$ operations over the entire dictionary update.

Concluding the analysis, the dominant operations in a single K-SVD iteration include the sparse-coding in line 5, the atom updates in line 10, and the coefficient updates in line 12 (which are computationally equivalent to the atom updates). We thus arrive at a total of

$$\begin{aligned} T_{\text{K-SVD}} &= R\cdot\big(2NL+K^2L+3KL+K^3\big)+NL^2+4RKN+4RKL+4NL^2 \\ &= R\cdot\big(2NL+K^2L+7KL+K^3+4KN\big)+5NL^2 \qquad (3.1) \end{aligned}$$

operations per iteration. Assuming an asymptotic behavior of $K \ll L\sim N \ll R$, the above expression simplifies to the following final K-SVD operation count *per training iteration*,

$$T_{\text{K-SVD}} \approx R\cdot\big(K^2L+2NL\big)\ . \qquad (3.2)$$

As can be seen, this is really just the operation count for sparse-coding R signals using Batch-OMP (excluding the precomputation which is negligible). The reason is that with the elimination of the explicit computation of $\mathbf{E}$ each iteration, the remaining computations become insignificant compared to the sparse-coding task in line 5. The Approximate K-SVD algorithm is therefore dominated by its sparse-coding step, emphasizing once again the significance of using Batch-OMP for its implementation.

4 Conclusion

We have discussed efficient OMP and K-SVD implementations which vastly improve over the straightforward ones. The Batch-OMP method is useful for a wide range of sparsity-based techniques which require large sets of signals to be coded. The Approximate K-SVD implementation, which employs Batch-OMP as its sparse-coding method, also improves on the dictionary update step by using a significantly faster and more memory efficient atom update computation.

Both the Batch-OMP and K-SVD implementations are made available as Matlab® toolboxes at `http://www.cs.technion.ac.il/~ronrubin/software.html`. The toolboxes can be freely used for personal and educational purposes. Questions and comments can be sent to `ronrubin@cs.technion.ac.il`.

References

[1] G. Davis, S. Mallat, and M. Avellaneda, "Adaptive Greedy Approximations," *Constructive Approximation*, vol. 13, no. 1, pp. 57–98, 1997.

[2] Y. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal Matching Pursuit: Recursive Function Approximation with Applications to Wavelet Decomposition," *1993 Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, pp. 40–44, 1993.

[3] S. Chen, D. Donoho, and M. Saunders, "Atomic Decomposition by Basis Pursuit," *SIAM Review*, vol. 43, no. 1, pp. 129–159, 2001.

[4] D. Donoho and M. Elad, "Optimal Sparse Representation in General (Nonorthogonal) Dictionaries via L_1 Minimization," *Proc. of the National Academy of Sciences*, vol. 100, pp. 2197–2202, 2003.

[5] I. Gorodnitsky and B. Rao, "Sparse signal reconstruction from limited data using FOCUSS: A re-weighted minimum norm algorithm," *IEEE Trans. on Signal Processing*, vol. 45, no. 3, pp. 600–616, 1997.

[6] S. Mallat, *A Wavelet Tour of Signal Processing.* Academic Press, 1999.

[7] E. Candes and D. Donoho, "Curvelets – a surprisingly effective nonadaptive representation for objects with edges," *Curves and Surfaces*, 1999.

[8] M. Do and M. Vetterli, "The contourlet transform: an efficient directional multiresolution image representation," *IEEE Trans. on Image Processing*, vol. 14, no. 12, pp. 2091–2106, 2005.

[9] M. Aharon, M. Elad, and A. Bruckstein, "The K-SVD: An algorithm for designing of overcomplete dictionaries for sparse representation," *IEEE Trans. on Signal Processing*, vol. 54, no. 11, pp. 4311–4322, 2006.

[10] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Trans. on Image Processing*, vol. 15, no. 12, pp. 3736–3745, 2006.

[11] M. Protter and M. Elad, "Image Sequence Denoising via Sparse and Redundant Representations," *IEEE Trans. on Image Processing*, 2008.

[12] O. Bryt and M. Elad, "Compression of Facial Images Using the K-SVD Algorithm," *J. of Visual Communication and Image Representation*, 2008. To appear.

[13] J. Mairal, G. Sapiro, and M. Elad, "Learning Multiscale Sparse Representations for Image and Video Restoration," *SIAM Multiscale Modeling and Simulation*, 2008. To appear.

[14] J. Mairal, M. Elad, and G. Sapiro, "Sparse Representation for Color Image Restoration," *IEEE Trans. on Image Processing*, vol. 17, no. 1, pp. 53–69, 2008.

[15] T. Blumensath and M. Davies, "Gradient pursuits," 2007. Submitted.

[16] S. Cotter, R. Adler, R. Rao, and K. Kreutz-Delgado, "Forward Sequential Algorithms for Best Basis Selection," *IEEE Proc. Vision, Image and Signal Processing*, vol. 146, no. 5, pp. 235–244, 1999.

[17] M. Gharavi-Alkhansari and T. Huang, "A Fast Orthogonal Matching Pursuit Algorithm," *Proc. of the 1998 IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, 1998.*, vol. 3, 1998.

[18] S. Krstulovic and R. Gribonval, "MPTK: Matching Pursuit made Tractable," *Proc. ICASSP'06, Toulouse, France*, 2006.