Overcomplete Dictionary Learning With Jacobi Atom Updates

Paul Irofti, Bogdan Dumitrescu
Department of Automatic Control and Computers
University Politehnica of Bucharest
313 Spl. Independenţei, 060042 Bucharest, Romania
Email: paul@irofti.net, bogdan.dumitrescu@acse.pub.ro

***Abstract*—Dictionary learning for sparse representations is traditionally approached with sequential atom updates, in which an optimized atom is used immediately for the optimization of the next atoms. We propose instead a Jacobi version, in which groups of atoms are updated independently, in parallel. Extensive numerical evidence for sparse image representation shows that the parallel algorithms, especially when all atoms are updated simultaneously, give better dictionaries than their sequential counterparts.**

***Keywords*—dictionary learning, parallel algorithm, sparse representation**

I. Introduction

Sparse representations are the basis for a wide range of very effective signal processing techniques with numerous applications for, but not limited to, audio and image processing.

In this paper, we approach the problem of training dictionaries for sparse representations by learning from a representative data set. Given a set of signals $Y \in \mathbb{R}^{p\times m}$ and a sparsity level s, the goal is to find a dictionary $D \in \mathbb{R}^{p\times n}$ that minimizes the Frobenius norm of the approximation error

$$E = Y - DX, \tag{1}$$

where $X \in \mathbb{R}^{n\times m}$ is the associated s-sparse representations matrix, with at most s nonzero elements on each column. Otherwise said, each column (signal or data vector) from Y is represented as a linear combination of at most s columns (atoms) from D. To eliminate the magnitude ambiguity in this bilinear problem, where both D and X are unknown, the columns of the dictionary are constrained to unit norm.

Since dictionary learning (DL) for sparse representations is a hard problem, the most successful algorithms, like K-SVD [1] (and its approximate version AK-SVD [2]) and MOD [3], adopt an alternating optimization procedure with two basic stages. First, fixing the current dictionary D (initialized randomly or with a subset of Y), the sparse representations X are computed with Orthogonal Matching Pursuit (OMP) [4] or another algorithm. Then, keeping X fixed, a new dictionary is obtained through various techniques. The second stage, where the atoms of the dictionary are updated, makes the main difference between DL algorithms. Recent methods or improvements can be found in [5]–[8]. Overviews of earlier work and applications are presented in [9], [10].

Excepting MOD, all these DL algorithms update the atoms one by one, in Gauss-Seidel style. The motivation is the classical one: an updated atom, assumed to be better than its previous value, can be used immediately for other updates. We investigate here the Jacobi version of several algorithms, where groups of atoms are updated simultaneously. We started this work in [11], where our study was confined to AK-SVD, aiming at reducing the dictionary design time on a GPU architecture. However, extensive numerical evidence shows that not only this strategy is not worse than the standard sequential approach, but in many circumstances gives a smaller representation error (1). This manuscript presents the Jacobi atom updates (JAU) strategy in section II, its particular form for a few of the best sequential methods in section III and the above mentioned numerical evidence in section IV.

As a side remark, the name ”parallel atom updates” (PAU) is at least as good as JAU to label our approach. Unfortunately, this name was already used in [8] although the atoms are updated sequentially there, using several AK-SVD update sweeps. An idea similar with PAU is called more appropriately ”dictionary update cycles” in [5], in the context of K-SVD. An algorithm that updates the atoms simultaneously is SimCO [12]. However, optimal gradient descent is used there, which makes the algorithm very different from ours and in particular much more complex.

II. Jacobi Atom Updates Stategy

The general form of the proposed dictionary learning method with Jacobi atom updates is presented in Algorithm 1. At iteration k of the DL method, the two usual stages are performed. In step 1, the current dictionary $D^{(k)}$ and the signals Y are used to find the sparse representation matrix $X^{(k)}$ with s nonzero elements on each column; we used OMP, as widely done in the literature.

The atom update stage takes place in groups of $\tilde{n}$ atoms. We assume that $\tilde{n}$ divides n only for the simplicity of description. Steps 2 and 3 of Algorithm 1 perform a full sweep of the atoms. All the $\tilde{n}$ atoms from the same group are updated independently (step 4), using one of the various available rules, some of them discussed in the next section. Once a group is

This work was supported by the Romanian National Authority for Scientific Research, CNCS - UEFISCDI, project number PN-II-ID-PCE-2011-3-0400.

```
Input: current dictionary $D^{(k)} \in \mathbb{R}^{p\times n}$
       signals set $Y \in \mathbb{R}^{p\times m}$
       number of parallel atoms $\tilde{n}$
Output: next dictionary $D^{(k+1)}$
Compute $s$-sparse representations $X^{(k)} \in \mathbb{R}^{n\times m}$
such that $Y \approx D^{(k)}X^{(k)}$
for $\ell = 1$ to $n/\tilde{n}$ do
  for $j = (\ell-1)\tilde{n}+1$ to $\ell\tilde{n}$, in parallel do
    Update $d_j^{(k+1)}$
    Normalize: $d_j^{(k+1)} \leftarrow d_j^{(k+1)}/\|d_j^{(k+1)}\|$
  end for
end for
```

Algorithm 1: General structure of a DL-JAU iteration

```
Input: current dictionary $D \in \mathbb{R}^{p\times n}$
       signals set $Y \in \mathbb{R}^{p\times m}$
       sparse representations $X \in \mathbb{R}^{n\times m}$
       number of parallel atoms $\tilde{n}$
Output: next dictionary $D$
for $\ell = 1$ to $n/\tilde{n}$ do
  $E = Y - DX$
  for $j = (\ell-1)\tilde{n}+1$ to $\ell\tilde{n}$, in parallel do
    $F = E_{\mathcal{I}_j} + d_j x_{j,\mathcal{I}_j}$
    $d_j = F x_{j,\mathcal{I}_j}^T/(x_{j,\mathcal{I}_j} x_{j,\mathcal{I}_j}^T)$
    $d_j \leftarrow d_j/\|d_j\|_2$
  end for
end for
```

Algorithm 2: P-SGK Atom Updates

processed, its updated atoms are used for updating the other atoms; so, atom $d_j^{(k+1)}$ (column j of $D^{(k+1)}$) is computed in step 4 using $d_i^{(k+1)}$ if

$$\lfloor (i-1)/\tilde{n} \rfloor < \lfloor (j-1)/\tilde{n} \rfloor, \tag{2}$$

i.e. $i < j$ and d_i not in the same group as d_j, and $d_i^{(k)}$ otherwise. Putting $\tilde{n} = 1$ gives the usual sequential Gauss-Seidel form. Taking $\tilde{n} = n$ leads to a fully parallel update, i.e. the form that is typically labeled with Jacobi's name.

The specific atom update strategy of each algorithm is contained in step 4 while step 5 is the usual normalization constraint on the dictionary.

The proposed form has obvious potential for a smaller execution time on a parallel architecture. We lightly touch this issue here and provide comparative execution times from a few experiments in section IV; the reader can consult [11] for an analysis of the GPU implementation of AK-SVD. Our main focus here is on the quality of the designed dictionary.

III. Particular forms of the algorithm

Typically, the atom update problem is posed as follows. We have the dictionary, denoted generically D, and the associated representations matrix X and we want to optimize atom d_j. In the DL context, at iteration k of the learning process, the dictionary is made of atoms from $D^{(k)}$ and $D^{(k+1)}$, as explained by the phrase around equation (2). We denote $\mathcal{I}_j$ the (column) indices of the signals that use d_j in their representation, i.e. the indices of the nonzero elements on the j-th row of X. Excluding atom d_j, the representation error matrix (1), reduced to the relevant columns, becomes

$$F = E_{\mathcal{I}_j} + d_j x_{j,\mathcal{I}_j}. \tag{3}$$

The updated atom d_j is the solution of the optimization problem

$$\min_{d_j \in \mathbb{R}^p} \quad \|F - d_j\, x_{j,\mathcal{I}_j}\|_F^2 \tag{4}$$

The norm constraint $\|d_j\|_2 = 1$ is usually imposed after solving the optimization problem.

AK-SVD. The K-SVD algorithm and its approximate version AK-SVD [2] consider that $x_{j,\mathcal{I}_j}$ is also a variable. Problem (4) becomes a rank-1 approximation problem, solved by AK-SVD with a single iteration of the power method:

$$\begin{aligned} d_j^{(k+1)} &= F(x_{j,\mathcal{I}_j}^{(k)})^T/\|F(x_{j,\mathcal{I}_j}^{(k)})^T\|_2 \\ x_{j,\mathcal{I}_j}^{(k+1)} &= F^T d_j^{(k+1)} \end{aligned} \tag{5}$$

Note that the representations are also changed.

SGK. DL for sparse representations as a generalization of K-Means clustering (SGK) [6] solves directly problem (4). This is a least squares problem whose solution is

$$d_j^{(k+1)} = F x_{j,\mathcal{I}_j}^T/(x_{j,\mathcal{I}_j} x_{j,\mathcal{I}_j}^T). \tag{6}$$

The atom updates part of the general JAU scheme from Algorithm 1 has the form described by Algorithm 2, named P-SGK (with P from Parallel). The error E is recomputed in step 2 before each group of atom updates, thus taking into account the updated values of the previous groups. Steps 4 and 5 implement relations (3) and (6), respectively. Step 6, the normalization, is identical with that from the general scheme.

To obtain the JAU version of AK-SVD (named PAK-SVD), we replace step 4 by the operations from (5). Note that, for full parallelism ($\tilde{n} = n$), P-SGK and PAK-SVD are identical, since the atoms produced by (5) and (6) have the same direction. For full parallelism the representations updated by PAK-SVD are not used, while if $\tilde{n} < n$, some updated representations affect the error matrix from step 2.

NSGK. The update problem (4) is treated in [7] in terms of differences with respect to the current dictionary and representations, instead of working directly with D and X. Applying this idea to SGK, the optimization problem is similar, but with the signal matrix Y replaced by

$$Z = Y + D^{(k)}X^{(k-1)} - D^{(k)}X^{(k)} \tag{7}$$

where $X^{(k-1)}$ is the sparse representation matrix at the beginning of the k-iteration of the DL algorithm, while $X^{(k)}$ is the matrix computed in the k-th iteration (e.g. in step 1 of Algorithm 1). The P-NSGK algorithm (NSGK stands for New SGK, the name used in [7]) is thus identical with P-SGK, with step 2 modified according to (7).

TABLE I. BEST RMSE VALUES AFTER 200 ITERATIONS

	$n = 128$	$n = 256$	$n = 512$
NSGK	0.0185	0.0168	0.0154
P-NSGK	0.0167	0.0154	0.0139
SGK	0.0201	0.0185	0.0166
P-SGK	0.0165	0.0153	0.0138
AK-SVD	0.0201	0.0184	0.0163

IV. NUMERICAL RESULTS

We give here numerical evidence supporting the advantages of the JAU scheme. We compare the algorithms PAK-SVD, P-SGK and P-NSGK with their sequential counterparts. We report results obtained with the same input data for all the algorithms; in particular, the initial dictionary is the same. The sparse representations were computed via OMP.

Dictionary learning. The training signals were images taken from the USC-SIPI [13] database (e.g. barb, lena, boat, etc.). The images were normalized and split into random 8×8 blocks. The initial dictionary was built with random atoms.

In a first experiment, we used $m = 16384$ signals of dimension $p = 64$ to train dictionaries with a target sparsity of $s = 8$. Table I shows the lowest RMSE after $k = 200$ iterations, averaged over 10 runs, for three values of the dictionary size n. Among the sequential algorithms, NSGK is the best, confirming the findings from [7]. However, all parallel algorithms are better than NSGK.

JAU versus MOD. We now compare the performance in representation error of the JAU algorithms with the intrinsically parallel algorithm named method of optimal directions (MOD) [3], which updates the dictionary D with the least-squares solution of the linear system $DX = Y$. For completeness we also include the sequential versions on which JAU algorithms are built. All algorithms performed DL for $k = 200$ iterations. Each data point from figures 1–4 represents an average of 10 runs of the same algorithm with the same data dimensions but with training sets composed of different image patches.

To see how sparsity influences the end result, Figure 1 presents the final errors for several sparsity levels when performing DL for dictionaries of $n = 128$ atoms on training sets of size $m = 8192$. We notice that for all three algorithms (NSGK, SGK and AK-SVD) the JAU methods perform similar to MOD at lower sparsity constraints, but as we pass $s = 8$ our proposed parallel strategy is clearly better. The sequential versions always come in last.

Figure 2 presents the final errors for DL keeping fixed $m = 12288$, $s = 12$ and varying the number of atoms. Again, the JAU versions are the winners for all three algorithms. Out of the sequential algorithms, NSGK is the only one that manages to out-perform MOD, while the others lag behind coming in last.

In Figure 3 we kept fixed $n = 256$ and $s = 10$ and performed DL with training sets going from $m = 4096$ to $m = 16384$ signals. JAU stays ahead of MOD almost always,

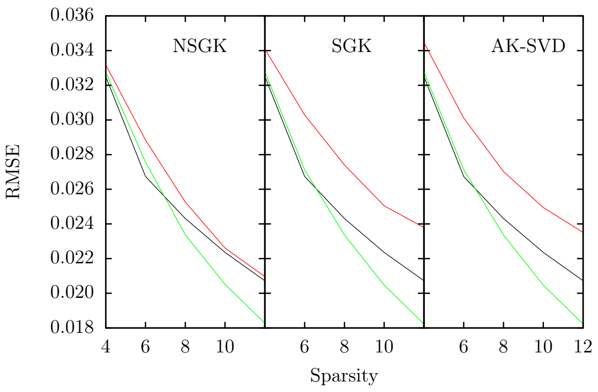

Fig. 1. Final errors for different sparsity constraints. Red: sequential versions; green: JAU algorithms; black: MOD.

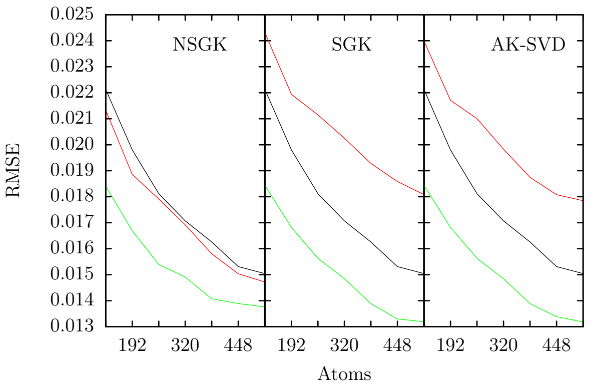

Fig. 2. Final errors for varied dictionary sizes. Red: sequential versions; green: JAU algorithms; black: MOD.

except for small signal sets where the results are similar. The sequential versions are once again the poorest performers.

Finally, we present in Figure 4 the error improvement at each iteration for all algorithms, for several sparsity levels, with $n = 128$, $m = 8192$. We can see that the JAU versions can jump back and forwards, especially during the first iterations. This is likely due to the fact that, initially, parallel updates do not necessarily progress towards the nearest local minimum. However, this may be globally beneficial and ultimately produce a lower representation error. Even though the JAU convergence is not as smooth as MOD or the sequential versions, it has a consistent descendent trend.

Execution times. We used OpenCL for our GPU implementation of the JAU algorithms on an ATI FirePro V8800 card, running at a maximum clock frequency of 825MHz, having 1600 streaming processors, 2GB global memory and 32KB local memory. For the sequential versions we used a C implementation that kept identical instructions wherever possible in order to show the improvements in execution time

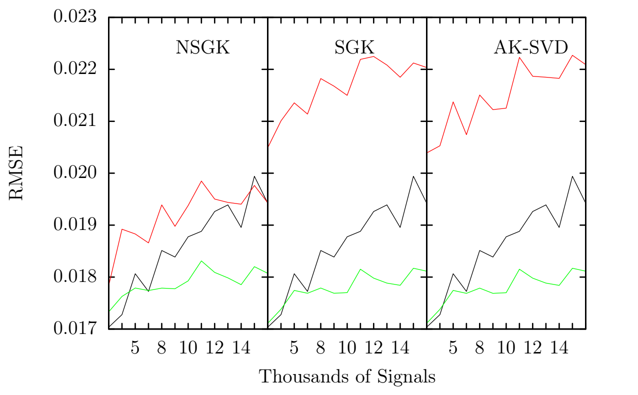

Fig. 3. Final errors for varied training set sizes. Red: sequential versions; green: JAU algorithms; black: MOD.

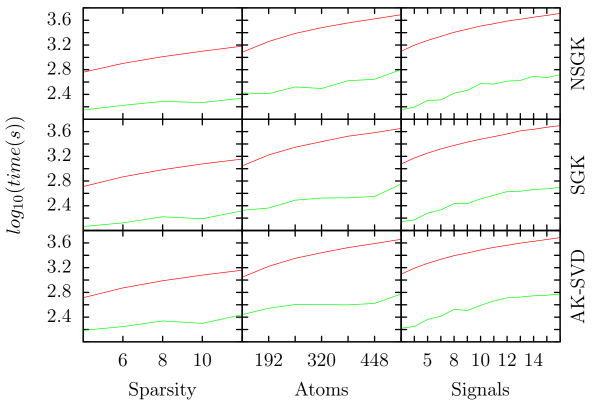

Fig. 5. Execution times. Red: sequential versions; green: JAU algorithms. Number of signals is in thousands.

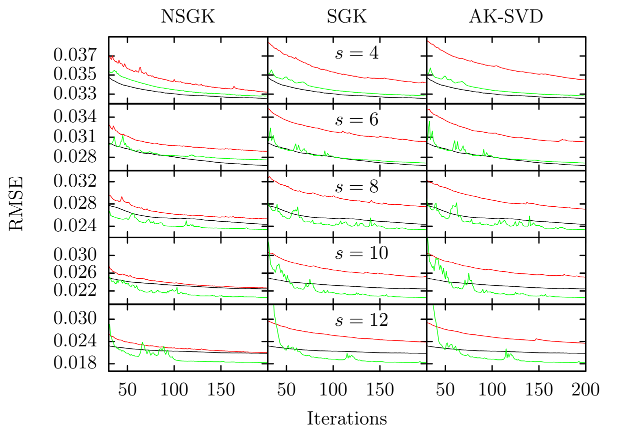

Fig. 4. Error evolution at different sparsity constraints. Red: sequential versions; green: JAU algorithms; black: MOD.

brought by the JAU strategy. The sequential tests were run on an Intel i7-3930K CPU working at 3.2GHz.

We present three experiments in Figure 5 where we vary the sparsity constraint, the number of atoms in the dictionary and the number of signals in the training set. Again, we used $k = 200$ iterations for all methods. For the sparsity experiment we used $n = 128$ and $m = 8192$. When studying the dictionary impact on the execution performance we kept fixed $m = 12288$ and $s = 6$ and varied the number of atoms. Finally, we increased the signal set with fixed $n = 256$, $s = 10$.

In all our experiments the JAU versions showed important improvements in execution time, with speed-up as high as 10.6 times for NSGK, 10.8 times for SGK and 12 times for AK-SVD. This was to be expected, since JAU algorithms are naturally parallel in the atom update stage.

V. Conclusions

We have shown that several dictionary learning algorithms, like AK-SVD [2], SGK [6] and NSGK [7], benefit from adopting Jacobi (parallel) atom updates instead of the usual Gauss-Seidel (sequential) ones. We have also shown that the new Jacobi algorithms outperform their sequential standard versions and also other types of algorithms like MOD [3], having a clearly better behavior with lower execution times.

References

[1] M. Aharon, M. Elad, and A. Bruckstein, “K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation,” *IEEE Trans. Signal Proc.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.

[2] R. Rubinstein, M. Zibulevsky, and M. Elad, “Efficient Implementation of the K-SVD Algorithm using Batch Orthogonal Matching Pursuit,” Tech. Rep. CS-2008-08, Technion Univ., Haifa, Israel, 2008.

[3] K. Engan, S.O. Aase, and J.H. Husoy, “Method of Optimal Directions for Frame Design,” in *IEEE Int. Conf. Acoustics Speech Signal Proc.*, 1999, vol. 5, pp. 2443–2446.

[4] Y.C. Pati, R. Rezaiifar, and P.S. Krishnaprasad, “Orthogonal Matching Pursuit: Recursive Function Approximation With Applications to Wavelet Decomposition,” in *27th Asilomar Conf. Signals Systems Computers*, Nov. 1993, vol. 1, pp. 40–44.

[5] L.N. Smith and M. Elad, “Improving Dictionary Learning: Multiple Dictionary Updates and Coefficient Reuse,” *IEEE Signal Proc. Letters*, vol. 20, no. 1, pp. 79–82, Jan. 2013.

[6] S. K. Sahoo and A. Makur, “Dictionary Training for Sparse Representation as Generalization of K-Means Clustering,” *Signal Processing Letters, IEEE*, vol. 20, no. 6, pp. 587–590, June 2013.

[7] M. Sadeghi, M. Babaie-Zadeh, and C. Jutten, “Dictionary Learning for Sparse Representation: a Novel Approach,” *IEEE Signal Proc. Letter*, vol. 20, no. 12, pp. 1195–1198, Dec. 2013.

[8] M. Sadeghi, M. Babaie-Zadeh, and C. Jutten, “Learning Overcomplete Dictionaries Based on Atom-by-Atom Updating,” *IEEE Trans. Signal Proc.*, vol. 62, no. 4, pp. 883–891, Feb. 2014.

[9] R. Rubinstein, A.M. Bruckstein, and M. Elad, “Dictionaries for Sparse Representations Modeling,” *Proc. IEEE*, vol. 98, no. 6, pp. 1045–1057, June 2010.

[10] I. Tosic and P. Frossard, “Dictionary Learning,” *IEEE Signal Proc. Mag.*, vol. 28, no. 2, pp. 27–38, Mar. 2011.

[11] P. Irofti and B. Dumitrescu, “GPU Parallel Implementation of the Approximate K-SVD Algorithm Using OpenCL,” in *EUSIPCO*, Lisbon, Portugal, 2014.

[12] W. Dai, T. Xu, and W. Wang, “Simultaneous Codeword Optimization (SimCO) for Dictionary Update and Learning,” *Signal Processing, IEEE Transactions on*, vol. 60, no. 12, pp. 6340–6353, 2012.

[13] A.G. Weber, “The USC-SIPI Image Database,” 1997.