Overcomplete Dictionary Design: the Impact of the Sparse Representation Algorithm

Paul Irofti, Bogdan Dumitrescu
University Politehnica of Bucharest,
Department of Automatic Control and Computers,
RO-060042 Bucharest, Romania
emails: paul@irofti.net, bogdan.dumitrescu@acse.pub.ro

***Abstract*—The design of dictionaries for sparse representations is typically done by iterating two stages: compute sparse representations for the fixed dictionary and update the dictionary using the fixed representations. Most of the innovation in recent work was proposed for the update stage, while the representation stage was routinely done with Orthogonal Matching Pursuit (OMP), due to its low complexity. We investigate here the use of other greedy sparse representation algorithms, more computationally demanding than OMP but still with convenient complexity. These algorithms include a new proposal, the projection-based Orthogonal Least Squares. It turns out that the effect of using better representation algorithms may be more significant than improving the update stage, sometimes even leveling the performance of different update algorithms. The numerous experimental results presented here suggest which are the best combinations of methods and open new ways of designing and using dictionaries for sparse representations.**

Keywords*—*sparse representation, greedy algorithms, dictionary learning

I. Introduction

Sparse representations [1], [2] are intensively used in signal processing applications, like image coding, denoising, echo channels modeling and many others. The overcomplete bases used for representation are either fixed, e.g. by taking rows of popular transforms like discrete cosine or wavelet, or trained using a representative sample of the signals that appear in the application at hand. Our topic is the latter case, known also as dictionary learning (DL).

The DL problem is posed as follows. Given a data set $\boldsymbol{Y} \in \mathbb{R}^{p\times m}$, made of m vectors (signals or data items) of size p, and a sparsity level s, the aim is to solve the optimization problem

$$\begin{aligned} \underset{\boldsymbol{D},\boldsymbol{X}}{\text{minimize}} \quad & \|\boldsymbol{Y} - \boldsymbol{D}\boldsymbol{X}\|_F^2 \\ \text{subject to} \quad & \|\boldsymbol{x}_i\|_0 \leq s,\ 1 \leq i \leq m \end{aligned} \tag{1}$$

where the variables are the dictionary $\boldsymbol{D} \in \mathbb{R}^{p\times n}$, whose columns are usually named atoms, and the sparse representations matrix $\boldsymbol{X} \in \mathbb{R}^{n\times m}$, whose columns have at most s nonzero elements. By $\boldsymbol{x}_i$ we denote the i-th column of the matrix $\boldsymbol{X}$, by $\|\cdot\|_0$ the number of nonzero elements of a vector (the so-called 0-norm, although not actually a norm) and by $\|\cdot\|_F$ the Frobenius norm of a matrix.

Since problem (1) is bilinear in its variables and thus not convex, almost all successful design algorithms attempt to solve it through an iterative process that alternates two basic stages until convergence or a sufficiently large number of iterations. First, given a dictionary $\boldsymbol{D}$, a sparse representation matrix $\boldsymbol{X}$ is sought by minimizing the objective of (1), named also representation error; note that this is also not a convex problem, due to the sparsity constraint. Then, keeping $\boldsymbol{X}$ fixed, a new dictionary is designed by minimizing or just decreasing the representation error; this is usually called the atom update stage. The iterative process usually converges to a local minimum, although even this is not guaranteed; the sparse representation stage does not necessarily decrease the representation error.

The known design algorithms are different in the treatment of the second stage. MOD [3] minimizes directly $\|\boldsymbol{Y} - \boldsymbol{D}\boldsymbol{X}\|$, which is a convex problem for a given $\boldsymbol{X}$. K-SVD [4] optimizes the atoms one by one, using a singular value decomposition (SVD), and also changes the representations together with the atoms; an approximate version called AK-SVD [5] replaces the costly SVD with one or few iterations of the power method. Several improvements of K-SVD were proposed recently; for earlier work we refer to [6], [7]. The SGK algorithm [8] also optimizes the atoms sequentially, by solving a least-squares problem, but without changing the representations. NSGK [9] expresses the optimization problem in terms of differences to the previous values of $\boldsymbol{D}$ and $\boldsymbol{X}$ and applies SGK to a linearized version of the objective function. Other methods like [10] and [11] play with the number of atom updates that are made for a given representation. Parallel instead of sequential updates were proposed in [12], with clear benefit in sparse image representations.

All these algorithms use Orthogonal Matching Pursuit (OMP) [13] in the first stage to compute the sparse representations. The main reason is that OMP is fast and also that it is used in applications together with the optimized dictionary; it makes sense to appeal to the same representation algorithm in training the dictionary as well as in using it. However, there are other greedy algorithms, like Orthogonal Least Squares (OLS) [14], Subspace Pursuit [15], Projection-Based OMP (POMP) or Look-Ahead OLS (LAOLS) [16], that are still fast enough for wide practical use, but achieving typically better representations than OMP. OLS can be implemented efficiently as an orthogonal triangularization with pivoting,

This work was supported by the Romanian National Authority for Scientific Research, CNCS - UEFISCDI, project number PN-II-ID-PCE-2011-3-0400. P. Irofti was also supported by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Ministry of European Funds through the Financial Agreement POSDRU/159/1.5/S/132395.

while POMP and LAOLS are able to trade off complexity and representation error. Besides the greedy category, the class of Basis Pursuit algorithms [1] offers a number of algorithms with very good representation error, but much slower than OMP and its improved versions.

Our contribution here is an empirical investigation of the impact that the sparse representation algorithm has in DL. The spark to start this study was the somewhat disconcerting fact that, in a test problem often used in the DL community, several algorithms gave quite similar result. The immediate but apparently ignored question is if OMP is the bottleneck of DL algorithms and thus progress in the atom update stage might be masked by it. Would better representation algorithms allow to discern which atom update method is in fact superior? Would better representation algorithms cause a significant decrease of the overall error in the DL problem (1)? Although we cannot provide definitive answers, we try at least to gain more insight into the DL process and assess DL algorithms on a steadier ground.

The paper is structured as follows. In section II we review several greedy representation algorithms that are good candidates to replace OMP. Among them, there is a new proposal, Projection-Based OLS, naturally derived from POMP. Section III is dedicated to a short description of the main atom update techniques used in DL. In section IV we present extensive numerical results for two standard test problems and draw practical conclusions based on these results.

II. Sparse representation algorithms

For the sake of completeness, we review here the considered sparse representation algorithms, although without giving all the implementation details. The basic problem is to find a sparse least-squares solution to the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, i.e. one that has at most s nonzero elements and minimizes $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2$. Given a set of indices $\mathcal{I}$, we denote $\boldsymbol{x}_\mathcal{I}$ the vector of solution elements corresponding to these indices. We implicitly assume that $\mathcal{I}$ is the current support of the solution and thus the other elements of $\boldsymbol{x}$ are zero. Similarly, $\boldsymbol{A}_\mathcal{I}$ is the restriction of $\boldsymbol{A}$ to the columns belonging to $\mathcal{I}$. We will assume that a generic function is available for finding the least-squares solution to the system $\boldsymbol{A}_\mathcal{I}\boldsymbol{x}_\mathcal{I} = \boldsymbol{b}$ with known support $\mathcal{I}$ and denote its use by

$$\boldsymbol{x} = \mathrm{LS}(\boldsymbol{A}, \boldsymbol{b}, \mathcal{I}). \qquad (2)$$

Since in greedy algorithms the support usually grows on the current one, this function could be implemented efficiently by using a partial orthogonal triangularization of $\boldsymbol{A}$ or a partial Cholesky factorization of the associated normal matrix $\boldsymbol{A}^T\boldsymbol{A}$. However, we will leave these details out of the presentation and give only the main ideas of the methods.

Orthogonal Matching Pursuit (OMP) [13], presented in Algorithm 1, grows the support by looking at the correlations of the matrix columns (atoms) with the current residual and adding the index of the largest correlation to the support; this is traditionally called matching pursuit criterion. Then, it computes the LS solution for the current support and updates the corresponding residual, thus preparing the next iteration. The residual is orthogonal on the selected columns, hence a column cannot be twice selected. (This is also the reason for the word "orthogonal" in OMP.) The interpretation of the selection criterion is simple: the new column is the one that decreases the most the residual norm, keeping fixed the values of the current solution $\boldsymbol{x}_\mathcal{I}$.

Algorithm 1: Orthogonal Matching Pursuit (OMP)

```
1  Arguments: $\boldsymbol{A}$, $\boldsymbol{b}$, $s$
2  Initialize: $\boldsymbol{r} = \boldsymbol{b}$, $\mathcal{I} = \emptyset$
3  for $k = 1 : s$ do
4      Compute correlations with residual: $\boldsymbol{z} = \boldsymbol{A}^T\boldsymbol{r}$
5      Select new column: $i = \arg\max_j |z_j|$
6      Increase support: $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$
7      Compute new solution: $\boldsymbol{x} = \mathrm{LS}(\boldsymbol{A}, \boldsymbol{b}, \mathcal{I})$
8      Update residual: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}_\mathcal{I}\boldsymbol{x}_\mathcal{I}$
```

Algorithm 2: Orthogonal Least Squares (OLS)

```
1  Arguments: $\boldsymbol{A}$, $\boldsymbol{b}$, $s$
2  Initialize: $\mathcal{I} = \emptyset$
3  for $k = 1 : s$ do
4      for $j \notin \mathcal{I}$ do
5          Build new support: $\mathcal{J} = \mathcal{I} \cup \{j\}$
6          Try solution: $\boldsymbol{x} = \mathrm{LS}(\boldsymbol{A}, \boldsymbol{b}, \mathcal{J})$
7          Residual norm: $\rho_j = \|\boldsymbol{b} - \boldsymbol{A}_\mathcal{J}\boldsymbol{x}_\mathcal{J}\|^2$
8      Select new column: $i = \arg\min_j \rho_j$
9      Increase support: $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$
10     Compute new solution: $\boldsymbol{x} = \mathrm{LS}(\boldsymbol{A}, \boldsymbol{b}, \mathcal{I})$
```

Orthogonal Least Squares (OLS) [14] takes a slightly different approach. The next column is the one that, together with the previous ones, gives a solution that minimizes the residual norm; hence, the elements of $\boldsymbol{x}_\mathcal{I}$ can change during the selection process. Algorithm 2 describes OLS. It can be efficiently implemented as an orthogonal triangularization with pivoting, which permanently orthogonalizes the not selected columns on the selected ones. This allows the computation of the selection criterion as a simple matrix-vector multiplication. In average, OLS is slightly better than OMP.

A refinement of OMP was proposed in [16], in the form of Projection-Based OMP (POMP), presented in Algorithm 3. Unlike OMP, a number of L candidate columns are selected via the matching pursuit criterion, where L is an argument of the algorithm. Then, a LS solution is computed for the support extended with all these columns. The winner is the column with the largest element of the solution. This approach is partly inspired from Subspace Pursuit [15], where selection is made by attempting to find LS solutions with larger support and looking at the magnitude of the solution elements: a large magnitude means a higher likelihood that the position belongs to the true support. It is clear that POMP with $L = 1$ is identical to OMP. Also, it is not necessarily true that POMP with $L > 1$ gives a better result than OMP, although this is usually the case.

An immediate extension, not investigated until now, is the Projection-Based OLS (POLS), presented in Algorithm 4, which is the application of the POMP selection idea in the context of OLS. Of course, for $L = 1$ POLS is identical to OLS. The performance of POLS with respect to POMP should

Algorithm 3: Projection-Based Orthogonal Matching Pursuit (POMP)

```
1  Arguments: $\boldsymbol{A}$, $\boldsymbol{b}$, $s$, $L$
2  Initialize: $\boldsymbol{r} = \boldsymbol{b}$, $\mathcal{I} = \emptyset$
3  for $k = 1 : s$ do
4      Compute correlations with residual: $\boldsymbol{z} = \boldsymbol{A}^T \boldsymbol{r}$
5      Select indices $\mathcal{J}$ of $L$ largest $|z_j|$
6      Compute potential solution: $\boldsymbol{x} = \text{LS}(\boldsymbol{A}, \boldsymbol{b}, \mathcal{I} \cup \mathcal{J})$
7      Select largest element index: $i = \arg\max_{j \in \mathcal{J}} |x_j|$
8      Increase support: $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$
9      Compute new solution: $\boldsymbol{x} = \text{LS}(\boldsymbol{A}, \boldsymbol{b}, \mathcal{I})$
10     Update residual: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}_{\mathcal{I}} \boldsymbol{x}_{\mathcal{I}}$
```

Algorithm 4: Projection-Based Orthogonal Least Squares (POLS)

```
1  Arguments: $\boldsymbol{A}$, $\boldsymbol{b}$, $s$, $L$
2  Initialize: $\mathcal{I} = \emptyset$
3  for $k = 1 : s$ do
4      Compute values $\rho_j$ like in steps 4–7 of OLS
5      Select indices $\mathcal{J}$ of $L$ largest $\rho_j$
6      Apply steps 6–9 of POMP
```

be similar to that of OLS with respect to OMP, but we will be able to say more in the numerical experiments section.

The last sparse representation algorithm that we use is Look-Ahead OLS (LAOLS) [16], given in Algorithm 5. Like in POMP, L indices are selected at each iteration via the matching pursuit criterion. After appending each of these indices to the current support, OMP is run starting from this support to the completion of an s-sparse solution. The newly selected index is that giving the lowest residual for the final OMP solution. So, a look-ahead search is performed for each index selection. We note that in fact the proper name of this algorithm would be LAOMP, since OMP is run in step 7 in the selection process. An OLS version is easy to derive by replacing step 4 with steps 4–7 of OLS and using OLS instead of OMP in step 7. However, we did not pursue the investigation of such an algorithm, due to the higher complexity of LAOLS with respect to the other presented algorithms.

Although we have mentioned Subspace Pursuit [15], we skip its presentation due to the poor results obtained in dictionary learning and we will not report any experiments with it.

Algorithm complexity. The significant instructions from an OMP iteration and their complexities are: the correlations in step 4, $\mathcal{O}(pn)$, the least squares computation from step 7 (which we assume to be incrementally computed for each new column), $\mathcal{O}(sp)$, and the residual update, $\mathcal{O}(sp)$. For s iterations, this amounts to a total complexity of $\mathcal{O}(spn + s^2p)$. POMP performs an extra LS operation in step 6 on the support and its L-sized extension $\mathcal{J}$. This amounts to L incremental LS calculations at each iteration, which results in a total $\mathcal{O}(sL(s + L)p)$ extra cost compared to plain OMP. If $L = \mathcal{O}(s)$, then the extra cost is $\mathcal{O}(s^3p)$.

An efficient implementation of OLS consists of two in-

Algorithm 5: Look-Ahead Orthogonal Least Squares (LAOLS)

```
1  Arguments: $\boldsymbol{A}$, $\boldsymbol{b}$, $s$, $L$
2  Initialize: $\boldsymbol{r} = \boldsymbol{b}$, $\mathcal{I} = \emptyset$
3  for $k = 1 : s$ do
4      Compute correlations with residual: $\boldsymbol{z} = \boldsymbol{A}^T \boldsymbol{r}$
5      Select indices $\mathcal{J}$ of $L$ largest $|z_j|$
6      for $j \in \mathcal{J}$ do
7          Run OMP starting from $\mathcal{I} \cup \{j\}$, obtaining $\boldsymbol{x}$
8          Compute residual norm $\rho_j = \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|^2$
9      Select new column: $i = \arg\min_j \rho_j$
10     Increase support: $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$
11     Compute new solution: $\boldsymbol{x} = \text{LS}(\boldsymbol{A}, \boldsymbol{b}, \mathcal{I})$
```

tensive tasks: selection via correlations (similar to OMP), $\mathcal{O}(spn)$ and performing the transformations needed by the partial orthogonal triangularization, $\mathcal{O}(spn)$. Even though OLS has a similar theoretical complexity as OMP, it is slower by a constant factor, due to a larger constant multiplying the complexity term spn. POLS adds a single computationally significant instruction to OLS which is, again, the LS on the extended support, that was shown earlier to total to an extra cost of $\mathcal{O}(sL(s + L)p)$.

The increase in complexity for POMP and POLS with respect to OMP and OLS is negligible for small s, but becomes significant when $s^2 > n$. (We assume that $L \leq s$, which is a good practical choice.)

While the fastest option for representation remains OMP, POLS (and the other alternatives) are still a good candidate for DL, where execution time is not critical and representation error is important. For a more in-depth comparison and analysis of the algorithmic complexity we refer the reader to table I and section V from [16].

III. Atom update methods

Before presenting several recent methods for atom update, let us first remind in Algorithm 6 the general structure of most DL algorithms. The dictionary D is initialized either randomly or by a random selection of signal vectors. The norm of the atoms is forced to 1 at all stages, in order to eliminate the multiplicative indeterminacy in the product $\boldsymbol{DX}$. The iterative process is composed of the two main stages mentioned in the introduction. First, a sparse representation algorithm like those presented in the previous section is used to compute the sparse representation matrix X. Note that the problem can be decoupled, since each column of $\boldsymbol{X}$ can be computed separately by attempting to find a sparse solution to the LS problem $\boldsymbol{D}\boldsymbol{x}_i = \boldsymbol{y}_i$, for $i = 1 : m$. Then, the atoms are updated using different methods, all aiming to reduce the objective of (1). We will present next several techniques for updating atom $\boldsymbol{d}_j$. We denote $\mathcal{I}_j$ the set of the indices of nonzero elements on the j-th row of $\boldsymbol{X}$. Otherwise said, these are the indices of the signals whose representation involves the atom $\boldsymbol{d}_j$.

Algorithm 6: Dictionary learning – general structure

```
1 Arguments: signal matrix $\boldsymbol{Y}$, target sparsity $s$
2 Initialize: dictionary $\boldsymbol{D}$ (with normalized atoms)
3 for $k = 1, 2, \ldots$ do
4     With fixed $\boldsymbol{D}$, compute sparse representations $\boldsymbol{X}$
5     With fixed $\boldsymbol{X}$, update atoms $\boldsymbol{d}_j$, $j = 1 : n$
```

K-SVD [4] solves the optimization problem

$$\min_{\boldsymbol{d}_j, \boldsymbol{X}_{j,\mathcal{I}_j}} \left\| \left(\boldsymbol{Y}_{\mathcal{J}} - \sum_{\ell \neq j} \boldsymbol{d}_\ell \boldsymbol{X}_{\ell,\mathcal{I}_\ell} \right) - \boldsymbol{d}_j \boldsymbol{X}_{j,\mathcal{I}_j} \right\|_F^2 \tag{3}$$

where all atoms excepting $\boldsymbol{d}_j$ are fixed. Note that the matrix within parenthesis is the representation error of the dictionary without the atom $\boldsymbol{d}_j$. To minimize the error, the problem (3) is seen as a rank-1 approximation of this modified error matrix. The solution is given by the singular vectors corresponding to the largest singular value. The less complex AK-SVD [5] runs a single iteration of the power method to compute the two vectors. Note that (3) allows the representations to be changed also in this stage.

SGK [8] considers the same optimization problem, but only with $\boldsymbol{d}_j$ as variable:

$$\min_{\boldsymbol{d}_j} \left\| \left(\boldsymbol{Y}_{\mathcal{J}} - \sum_{\ell \neq j} \boldsymbol{d}_\ell \boldsymbol{X}_{\ell,\mathcal{I}_\ell} \right) - \boldsymbol{d}_j \boldsymbol{X}_{j,\mathcal{I}_j} \right\|_F^2 \tag{4}$$

This is a simple least-squares problem, for which an explicit solution can be easily computed. NSGK [9] operates similarly, but using differences with the previous values of the dictionary and representations.

All these algorithms update the atoms sequentially, using thus their most recent values when updating another atom. This is the typical Gauss-Seidel approach. We have looked also at the Jacobi version, in which the atoms are updated independently, meaning that problems like (3) or (4) are solved simultaneously for $j = 1 : n$. This technique can be applied to all presented atom update algorithms. We append the initial P (from parallel) to their name to denote the Jacobi versions.

IV. Numerical results

We give here numerical results for two DL problems used very often as benchmark: dictionary recovery and DL for sparse image representation. In the general structure of Algorithm 6, we use all the combinations of 5 representation methods (OMP, OLS, POMP, POLS, LAOLS) and 6 atom update methods (SGK, P-SGK, NSGK, P-NSGK, AK-SVD, PAK-SVD). All DL algorithms are run with the same data, in particular with the same initial dictionary. For POMP, POLS and LAOLS we took $L = s$.

A. Dictionary recovery

We generated a signal set of $m = 1500$ vectors as a linear combination of $s \in \{3, 4, 5\}$ atoms from a random original dictionary of $n = 50$ atoms with a size of $p = 20$ each and then proceeded to add white gaussian noise with SNR 10, 20, 30 and ∞ dB to the set in order to obtain the signals matrix $\boldsymbol{Y}$. We then ran the DL algorithms, with the noisy set $\boldsymbol{Y}$ and the known target sparsity s as inputs, for $9s^2$ iterations, initializing with a random dictionary. The resulting dictionary was checked against the original one: if the scalar product between two atoms, one from the resulting dictionary and one from the original dictionary, was larger than 0.99 (in absolute value), then the atom was considered successfully recovered. Tables I, II, and III show the percentages of recovered atoms for all combinations of methods, averaging over 50 tests. Here are some conclusions that can be drawn from the results.

TABLE I. Percentage of recovered atoms for SGK and P-SGK

s	Method	SNR			
		10	20	30	∞
3	SGK(OMP)	86.84	90.32	90.20	88.72
	SGK(OLS)	89.08	89.04	89.48	89.00
	SGK(POMP)	91.48	92.72	93.48	93.48
	SGK(POLS)	91.80	92.88	93.48	92.44
	SGK(LAOLS)	89.24	92.24	92.52	**92.88**
	P-SGK(OMP)	87.76	90.44	90.12	89.16
	P-SGK(OLS)	86.52	90.60	90.76	89.40
	P-SGK(POMP)	**92.04**	**94.56**	93.48	92.52
	P-SGK(POLS)	91.48	92.64	**93.64**	92.48
	P-SGK(LAOLS)	88.92	92.92	92.24	92.40
4	SGK(OMP)	71.52	92.16	92.32	91.72
	SGK(OLS)	73.80	93.20	93.36	93.20
	SGK(POMP)	88.04	96.24	95.48	96.40
	SGK(POLS)	87.60	96.28	95.76	**96.72**
	SGK(LAOLS)	85.84	95.52	93.80	94.96
	P-SGK(OMP)	65.64	92.68	92.04	92.48
	P-SGK(OLS)	73.36	92.64	91.68	93.40
	P-SGK(POMP)	87.16	**96.84**	95.76	96.44
	P-SGK(POLS)	**89.96**	96.36	**96.04**	96.04
	P-SGK(LAOLS)	84.64	95.32	95.24	94.88
5	SGK(OMP)	10.56	93.32	93.64	93.64
	SGK(OLS)	12.28	94.96	94.84	94.56
	SGK(POMP)	53.72	98.12	97.52	**98.84**
	SGK(POLS)	64.72	**98.52**	97.48	97.84
	SGK(LAOLS)	58.44	96.76	97.32	97.24
	P-SGK(OMP)	10.44	93.24	93.80	93.96
	P-SGK(OLS)	12.12	94.16	94.12	94.72
	P-SGK(POMP)	56.32	98.28	**98.12**	98.20
	P-SGK(POLS)	**69.28**	98.36	**98.12**	97.60
	P-SGK(LAOLS)	57.76	96.56	97.48	97.36

1. As expected, the more complex representation methods bring indeed an increase in performance: the recovery percentage obtained with POMP, POLS and LAOLS (especially the first two) is clearly better than with OMP, for all atom update methods. OLS and OMP are almost at the same level, with a marginal advantage for OLS.

2. For the same representation method, there is little difference between the atom update methods. The conclusion is that the recovery test (with these commonly used data) is not relevant for comparing atom update methods and that the sparse representation is actually the bottleneck here. A reason may be the small size of the problem or the constant used to decide similarity between the recovered and original atoms. In any case, it appears that the progress in atom update methods can no longer be assessed with this test.

TABLE II. PERCENTAGE OF RECOVERED ATOMS FOR NSGK AND P-NSGK

s	Method	SNR			
		10	20	30	∞
3	NSGK(OMP)	87.12	90.52	90.32	89.92
	NSGK(OLS)	87.44	91.52	90.28	90.52
	NSGK(POMP)	91.40	93.16	92.48	**93.96**
	NSGK(POLS)	90.96	93.32	**93.28**	93.40
	NSGK(LAOLS)	88.76	92.52	92.20	92.44
	P-NSGK(OMP)	86.12	90.52	91.36	89.84
	P-NSGK(OLS)	87.60	92.08	90.40	90.04
	P-NSGK(POMP)	90.36	93.48	**93.28**	91.88
	P-NSGK(POLS)	**91.48**	**93.76**	92.96	93.32
	P-NSGK(LAOLS)	88.44	92.52	92.60	93.36
4	NSGK(OMP)	68.16	92.88	92.76	92.84
	NSGK(OLS)	70.76	94.04	93.28	93.04
	NSGK(POMP)	85.40	**96.60**	95.96	**96.68**
	NSGK(POLS)	86.32	96.36	95.80	95.80
	NSGK(LAOLS)	83.44	96.08	95.60	95.48
	P-NSGK(OMP)	69.28	93.64	92.92	93.88
	P-NSGK(OLS)	70.24	93.48	93.44	93.96
	P-NSGK(POMP)	87.44	96.44	**96.20**	96.04
	P-NSGK(POLS)	**88.76**	96.52	96.12	96.44
	P-NSGK(LAOLS)	83.64	95.00	95.20	95.48
5	NSGK(OMP)	9.60	94.12	93.88	94.16
	NSGK(OLS)	8.08	94.96	94.72	95.04
	NSGK(POMP)	53.48	**98.52**	97.80	97.72
	NSGK(POLS)	**67.96**	97.48	97.48	97.76
	NSGK(LAOLS)	56.96	97.08	97.08	97.72
	P-NSGK(OMP)	11.68	94.32	93.60	93.52
	P-NSGK(OLS)	8.96	94.84	95.12	94.40
	P-NSGK(POMP)	51.72	98.32	**98.56**	**98.32**
	P-NSGK(POLS)	65.52	98.20	98.20	97.72
	P-NSGK(LAOLS)	51.96	97.64	97.56	97.36

TABLE III. PERCENTAGE OF RECOVERED ATOMS FOR AK-SVD AND PAK-SVD

s	Method	SNR			
		10	20	30	∞
3	AK-SVD(OMP)	88.00	89.16	89.96	88.20
	AK-SVD(OLS)	88.72	90.52	89.68	89.84
	AK-SVD(POMP)	91.24	94.00	92.92	92.32
	AK-SVD(POLS)	91.48	93.40	92.88	**92.56**
	AK-SVD(LAOLS)	88.40	92.08	92.76	92.28
	PAK-SVD(OMP)	87.76	90.44	90.12	89.16
	PAK-SVD(OLS)	86.52	90.60	90.76	89.40
	PAK-SVD(POMP)	**92.04**	**94.56**	93.48	92.52
	PAK-SVD(POLS)	91.52	92.64	**93.64**	92.48
	PAK-SVD(LAOLS)	88.92	92.64	92.28	92.44
4	AK-SVD(OMP)	71.80	92.44	92.76	92.68
	AK-SVD(OLS)	73.12	93.44	92.96	92.96
	AK-SVD(POMP)	88.04	**96.84**	95.84	**96.76**
	AK-SVD(POLS)	88.24	96.04	**96.36**	95.60
	AK-SVD(LAOLS)	82.92	94.92	96.24	95.68
	PAK-SVD(OMP)	65.64	92.68	92.04	92.48
	PAK-SVD(OLS)	73.36	92.64	91.68	93.40
	PAK-SVD(POMP)	87.16	**96.84**	95.76	96.44
	PAK-SVD(POLS)	**90.08**	96.36	96.04	96.04
	PAK-SVD(LAOLS)	84.60	95.32	95.24	94.92
5	AK-SVD(OMP)	10.48	93.44	93.20	93.00
	AK-SVD(OLS)	12.24	93.64	94.60	95.20
	AK-SVD(POMP)	54.04	**98.44**	97.64	98.12
	AK-SVD(POLS)	62.56	98.12	**98.28**	**98.96**
	AK-SVD(LAOLS)	48.16	96.96	96.72	96.44
	PAK-SVD(OMP)	10.44	93.24	93.80	93.96
	PAK-SVD(OLS)	12.12	94.16	94.12	94.72
	PAK-SVD(POMP)	56.32	98.28	98.12	98.20
	PAK-SVD(POLS)	**69.16**	98.36	98.12	97.52
	PAK-SVD(LAOLS)	59.60	97.56	97.00	97.36

B. Dictionary learning

We built the training signals $\boldsymbol{Y}$ from $m = 8192$ random 8×8 blocks taken from the USC-SIPI image collection [17] (e.g. barb, lena, boat etc.) that we vectorized as columns of $\boldsymbol{Y}$ (so, $p = 64$). With these signals, we trained dictionaries of size $n = 256$ over 200 iterations, with target sparsity $s = 8$, using again all combinations of methods. The final errors $\|\boldsymbol{Y} - \boldsymbol{D}\boldsymbol{X}\|_F / \sqrt{pm}$ are shown in table IV while the evolution of the representation error over the number of iterations is depicted in figures 1–11.

Regarding the final error, the conclusions are mixed. With a single exception, all the other representation methods are better than OMP, often much better; in this sense, the results are meeting normal expectations. The best results for an atom update method (in bold) are similar and OLS, LAOLS and POLS share the winners, while POMP is rather disappointing. OLS is clearly better for the parallel methods, a feature shared by OMP and POMP. On the contrary, LAOLS and POLS are systematically better for the sequential methods. We do not have an explanation for this feature of the results.

Figures 1–6 present the error evolution for each atom update method combined with the diverse representation methods. The error does not decrease uniformly, especially in the first iterations. The parallel methods suffer more from sudden increases in error, however having a decreasing trend.

Figures 7–11 present the error evolution from the viewpoint of the sparse representation methods. We note that POMP, POLS and LAOLS have a smoothing effect on the curves for all atom update methods, the decrease being more or less uniform after the first iterations.

Finally, one may wonder if the designed dictionaries could be used with OMP as representation method, although another method has been employed in learning. The reason is that one may need the dictionaries in practical applications where speed is of the essence, hence one may want to use the fastest reliable representation method, i.e. OMP, which is at least a few times faster than the other methods discussed in this study. On the contrary, in learning, which is a one-time operation, we often have the luxury to use higher complexity methods for obtaining a very good dictionary. So, we computed with OMP the representations $\boldsymbol{X}$ associated the dictionaries $\boldsymbol{D}$ designed by the various methods. The errors are given in table V, whose structure is similar to that of table IV. The first column is identical, since OMP is used both for learning and representation. The other columns are different, since the method used for learning is not OMP. As expected, the results in table V are generally worse than those in table IV. However, we are interested in the results that are better than in the first column; these are shown with bold digits. Remarkably, POLS gives consistently better results, which means that we could use any atom update method coupled with POLS in the DL process, then use the designed dictionary with OMP and thus

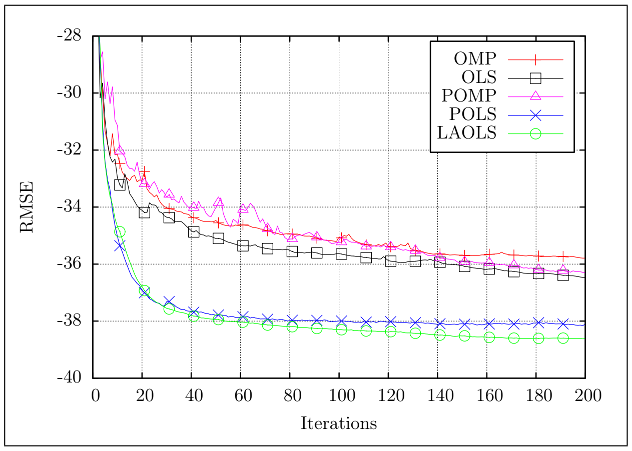

Fig. 1: Error evolution for SGK with different representation methods.

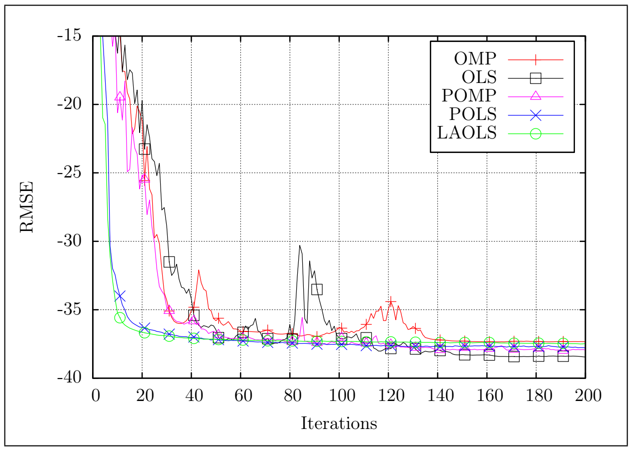

Fig. 4: Error evolution for P-SGK with different representation methods.

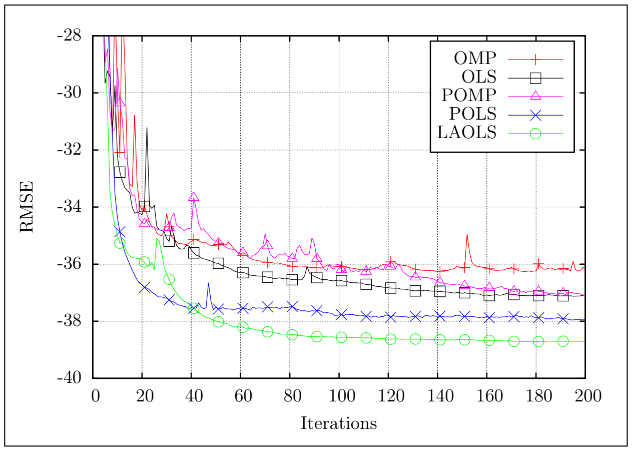

Fig. 2: Error evolution for NSGK with different representation methods.

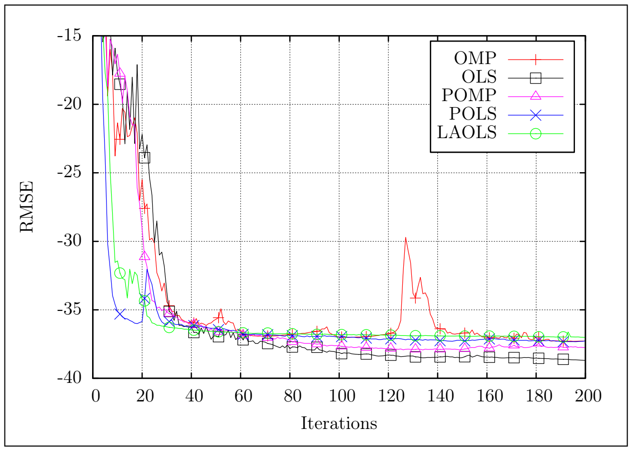

Fig. 5: Error evolution for P-NSGK with different representation methods.

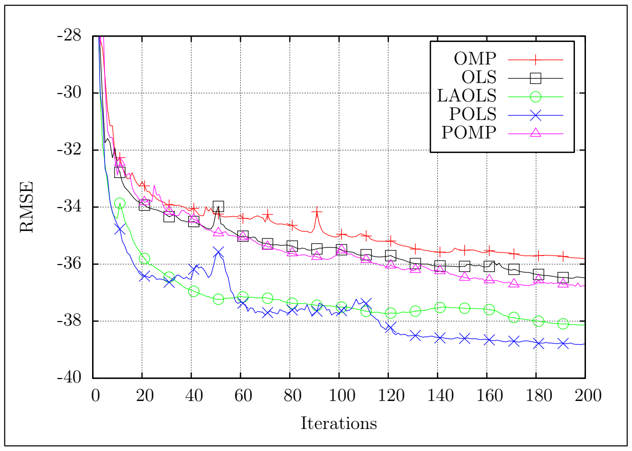

Fig. 3: Error evolution for AK-SVD with different representation methods.

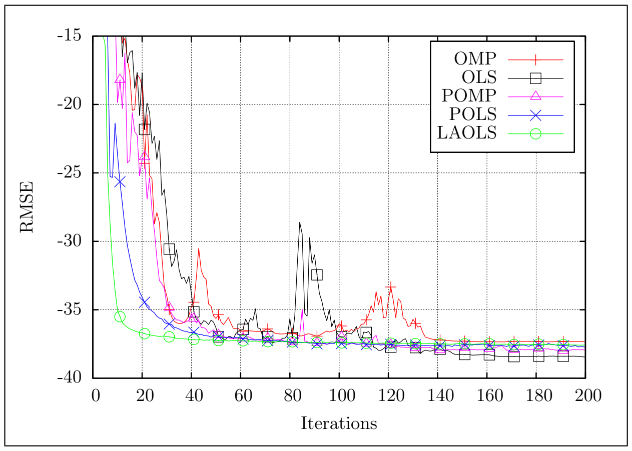

Fig. 6: Error evolution for PAK-SVD with different representation methods.

TABLE IV. DL FINAL ERRORS

	OMP	OLS	POMP	POLS	LAOLS
SGK	0.0162	0.0150	0.0153	0.0124	**0.0117**
P-SGK	0.0136	**0.0119**	0.0126	0.0129	0.0133
NSGK	0.0154	0.0139	0.0139	0.0126	**0.0116**
P-NSGK	0.0136	**0.0116**	0.0127	0.0136	0.0141
AK-SVD	0.0162	0.0150	0.0148	**0.0114**	0.0124
PAK-SVD	0.0136	**0.0119**	0.0126	0.0130	0.0132

TABLE V. REPRESENTATION ERRORS USING DESIGNED DICTIONARIES AND OMP

	OMP	OLS	POMP	POLS	LAOLS
SGK	0.0162	0.0332	0.0170	**0.0119**	**0.0127**
P-SGK	0.0136	0.0156	0.0157	**0.0124**	0.0136
NSGK	0.0154	0.0252	**0.0149**	**0.0123**	**0.0124**
P-NSGK	0.0136	**0.0133**	0.0159	**0.0128**	0.0144
AK-SVD	0.0162	0.0251	0.0165	**0.0119**	**0.0136**
PAK-SVD	0.0136	0.0156	0.0157	**0.0124**	0.0136

get better representations than using OMP in training. We can say that this is a new design method that is better than the current methods in exactly the same conditions.

V. CONCLUSIONS

We have studied the effects of combining several sparse representation and atom update methods for solving the problem of overcomplete dictionary design. As expected, replacing OMP with more sophisticated methods like OLS, POMP, POLS and LAOLS leads to better results and smoother convergence. In image representation applications, we also conclude that combining OLS with parallel atom update methods gives systematically good results. Finally, combining POLS and any considered atom update method produces dictionaries that give small representation errors also with OMP, thus allowing the fastest implementation for the use of the designed dictionary and obtaining better representations than all existing dictionary design methods that employ OMP in the learning process.

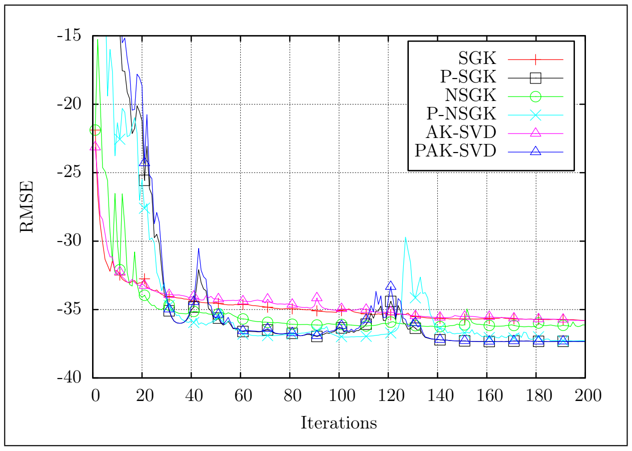

Fig. 7: Error evolution for OMP with different dictionary update methods.

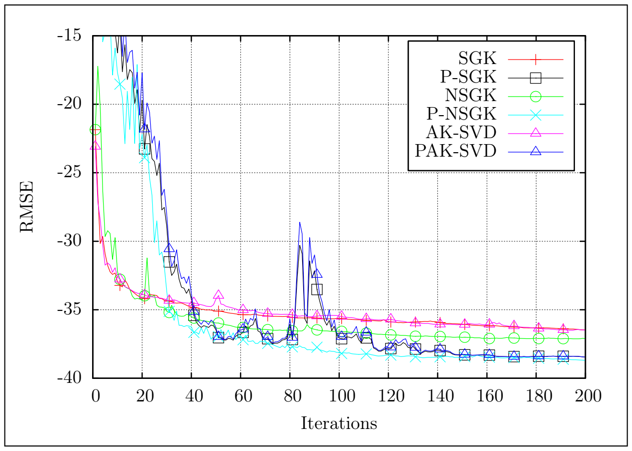

Fig. 8: Error evolution for OLS with different dictionary update methods.

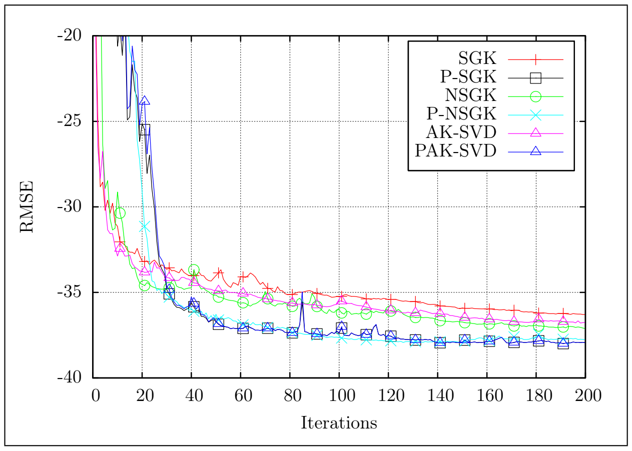

Fig. 9: Error evolution for POMP with different dictionary update methods.

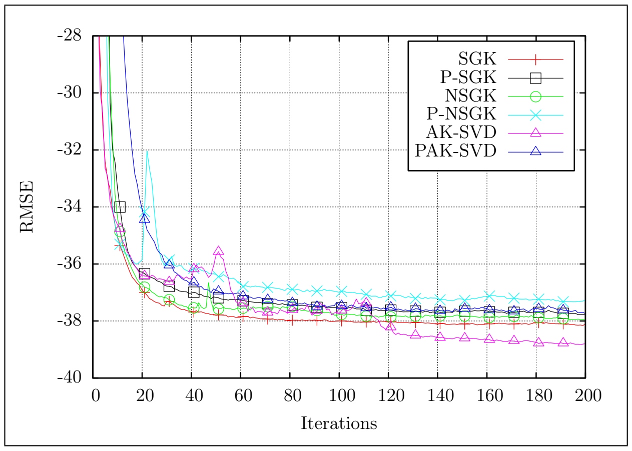

Fig. 10: Error evolution for POLS with different dictionary update methods.

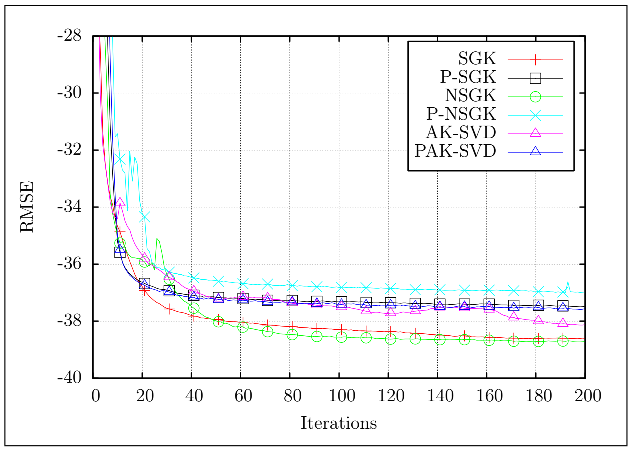

Fig. 11: Error evolution for LAOLS with different dictionary update methods.

References

[1] A. Bruckstein, D. Donoho, and M. Elad, "From Sparse Solutions of Systems of Equations to Sparse Modeling of Signals and Images," *SIAM Rev.*, vol. 51, no. 1, pp. 34–81, 2009.

[2] M. Elad, *Sparse and Redundant Representations: from Theory to Applications in Signal Processing*. Springer, 2010.

[3] K. Engan, S. Aase, and J. Husoy, "Method of optimal directions for frame design," in *IEEE Int. Conf. Acoustics Speech Signal Proc.*, vol. 5, 1999, pp. 2443–2446.

[4] 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.

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

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

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

[8] 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.

[9] 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.

[10] L. 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.

[11] 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.

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

[13] Y. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," in *27th Asilomar Conf. Signals Systems Computers*, vol. 1, Nov. 1993, pp. 40–44.

[14] S. Chen, S. Billings, and W. Luo, "Orthogonal Least Squares Methods and Their Application to Non-Linear System Identification," *Int. J. Control*, vol. 50, no. 5, pp. 1873–1896, 1989.

[15] W. Dai and O. Milenkovic, "Subspace Pursuit for Compressive Sensing Signal Reconstruction," *IEEE Trans. Info. Theory*, vol. 55, no. 5, pp. 2230–2249, May 2009.

[16] S. Chatterjee, M. Vehkapera, and M. Skoglund, "Projection-Based and Look-Ahead Strategies for Atom Selection," *IEEE Trans. Signal Proc.*, vol. 60, no. 2, pp. 634–647, Feb. 2012.

[17] A. Weber, "The USC-SIPI Image Database," 1997. [Online]. Available: http://sipi.usc.edu/database