A New Algorithm for Dictionary Learning Based on Convex Approximation

Javad Parsa, Mostafa Sadeghi, Massoud Babaie-Zadeh
Electrical Engineering Department
Sharif University of Technology, Tehran, Iran
javadparsa86@gmail.com, m.saadeghii@gmail.com, mbzadeh@yahoo.com

Christian Jutten
GIPSA-Lab, Grenoble, and Institut. Universitaire de France, France.
christian.jutten@gipsa-lab.grenoble-inp.fr

***Abstract*—The purpose of dictionary learning problem is to learn a dictionary $\mathbf{D}$ from a training data matrix $\mathbf{Y}$ such that $\mathbf{Y} \approx \mathbf{DX}$ and the coefficient matrix $\mathbf{X}$ is sparse. Many algorithms have been introduced to this aim, which minimize the representation error subject to a sparseness constraint on $\mathbf{X}$. However, the dictionary learning problem is non-convex with respect to the pair $(\mathbf{D},\mathbf{X})$. In a previous work [Sadeghi *et al.*, 2013], a convex approximation to the non-convex term $\mathbf{DX}$ has been introduced which makes the whole DL problem convex. This approach can be almost applied to any existing DL algorithm and obtain better algorithms. In the current paper, it is shown that a simple modification on that approach significantly improves its performance, in terms of both accuracy and speed. Simulation results on synthetic dictionary recovery are provided to confirm this claim.**

***Index Terms*—Compressed sensing, sparse coding, convex approximation, convergence rate, dictionary learning**

I. Introduction

Dictionary learning (DL) for sparse approximation has been utilized in many areas such as compressed sensing and image processing [1]–[3]. In the dictionary learning problem, given a training dataset $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_L] \in \mathbb{R}^{m\times L}$, the goal is to learn a dictionary $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_n] \in \mathbb{R}^{m\times n}$ such that $\mathbf{Y} \approx \mathbf{DX}$ where $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L] \in \mathbb{R}^{n\times L}$ is sparse. Each column of $\mathbf{D}$ is called an atom [1]. A general dictionary learning algorithm solves the following problem [2], [4]–[6]:

$$\min_{\mathbf{D}\in\mathcal{D},\mathbf{X}\in\mathcal{X}} \ \|\mathbf{Y}-\mathbf{DX}\|_F^2, \tag{1}$$

where $\|.\|_F$ is the Frobenius norm, $\mathcal{D} \triangleq \left\{\mathbf{D} :\ \forall i, \|\mathbf{d}_i\|_2^2 \le 1\right\}$ and $\mathcal{X} \triangleq \{\mathbf{X} :\ \forall i, \|\mathbf{x}_i\|_0 \le \tau\}$. During the last decade, many algorithms have been introduced for solving (1) [2], [7], most of which are based on iterating between the two following stages:

1) Sparse representation:

$$\mathbf{X}^{(k+1)} = \underset{\mathbf{X}\in\mathcal{X}}{\operatorname{argmin}} \ \|\mathbf{Y}-\mathbf{D}^{(k)}\mathbf{X}\|_F^2, \tag{2}$$

2) Dictionary update:

$$\mathbf{D}^{(k+1)} = \underset{\mathbf{D}\in\mathcal{D}}{\operatorname{argmin}} \ \|\mathbf{Y}-\mathbf{D}\mathbf{X}^{(k+1)}\|_F^2. \tag{3}$$

Stage 1 is an ordinary sparse coding problem [8]–[11], which can be done, for example, by Orthogonal Matching Pursuit (OMP) [12]. In stage 2, by using $\mathbf{X}^{k+1}$ obtained in stage 1, a new estimation of dictionary is obtained.

Method of Optimal Directions (MOD) [13], is one of the simplest DL algorithms. It updates the dictionary according to the following least squares solution followed by normalizing all columns of the dictionary:

$$\mathbf{D}^{k+1} = \mathbf{Y}\mathbf{X}^{k+1}(\mathbf{X}^{(k+1)}{\mathbf{X}^{(k+1)}}^T)^{-1}. \tag{4}$$

K-Singular Value Decomposition (K-SVD) [4] is another well-known algorithm, which unlike MOD updates the atoms one at a time. K-SVD is actually a generalization of the K-means algorithm for clustering [4]. As mentioned in [14], although K-SVD is sequential like K-means, it fails to simplify to K-means by destroying the structure in the sparse coefficients. However, in contrast to K-means, K-SVD forces the atoms of the dictionary to be normalized, and the matrix $\mathbf{X}$ does not necessarily contain 0 or 1. In Sequential Generalization of K-means (SGK) algorithm [14] these issues have been resolved.

Multiple Dictionary Update (MDU) [15] is another DL algorithm which is similar to MOD in its structure. This algorithm uses (4) for dictionary update step along with updating non-zero entries of $\mathbf{X}$ as $\forall i :\ \mathbf{x}_i(\omega_i) \leftarrow (\mathbf{D}_i^T\mathbf{D}_i)^{-1}\mathbf{D}_i^T\mathbf{y}_i$, where $\omega_i \triangleq \{j | 1 \le j \le n, \mathbf{x}_i(j) \ne 0\}$.

As can be seen from (1), dictionary learning is a non-convex problem due to the multiplicative term $\mathbf{DX}$. In a previous work [5], Sadeghi *et al.* proposed to approximate this term with a convex one. To do this, it was suggested to write $\mathbf{D} = \mathbf{D}_0 + \mathbf{D} - \mathbf{D}_0$ and $\mathbf{X} = \mathbf{X}_0 + \mathbf{X} - \mathbf{X}_0$. Then, we would have

$$\begin{aligned}\mathbf{DX} &= (\mathbf{D}_0 + \mathbf{D} - \mathbf{D}_0)(\mathbf{X}_0 + \mathbf{X} - \mathbf{X}_0)\\ &\approx \mathbf{D}_0\mathbf{X} + \mathbf{D}\mathbf{X}_0 - \mathbf{D}_0\mathbf{X}_0,\end{aligned} \tag{5}$$

in which, it is assumed that $\|(\mathbf{D}-\mathbf{D}_0)(\mathbf{X}-\mathbf{X}_0)\|_F$ is small[1]. Then, instead of solving the original non-convex DL problem (1), the following convex problem has been proposed in [5]:

$$\min_{\mathbf{D}\in\mathcal{D},\mathbf{X}\in\mathcal{X}} \ \|\mathbf{Y} + \mathbf{D}_0\mathbf{X}_0 - \mathbf{D}_0\mathbf{X} - \mathbf{D}\mathbf{X}_0\|_F^2. \tag{6}$$

To solve (6), an alternative minimization method has been utilized as follows:

1) Sparse representation:

$$\mathbf{X}^{(k+1)} = \underset{\mathbf{X}\in\mathcal{X}}{\operatorname{argmin}} \ \|\mathbf{Y}-\mathbf{D}^{(k)}\mathbf{X}\|_F^2. \tag{7}$$

[1]It is easy to see that the approximation in (5) is actually the first order approximation of Taylor series of $\mathbf{DX}$ around $(\mathbf{D}_0, \mathbf{X}_0)$. Refer to Appendix.

978-9-0827-9703-9/19/$31.00 ©2019 IEEE

2) Dictionary update:

$$\mathbf{D}^{(k+1)} = \underset{\mathbf{D}\in\mathcal{D}}{\operatorname{argmin}} \ \|\mathbf{Y} - \mathbf{D}^{(k)}(\mathbf{X}^{(k+1)} - \mathbf{X}^{(k)}) - \mathbf{D}\mathbf{X}^{(k)}\|_F^2. \quad (8)$$

For the above stages, the following substitutions have been made in (6):

- In stage 1, $\mathbf{D} = \mathbf{D}_0 = \mathbf{D}^{(k)}$,
- In stage 2, $\mathbf{X} = \mathbf{X}^{(k+1)}$, $\mathbf{X}_0 = \mathbf{X}^{(k)}$, and $\mathbf{D}_0 = \mathbf{D}^{(k)}$.

Note that, this approach can be applied on almost any DL algorithm by using (8) instead of (3) as its dictionary update step. In [5], this idea has been successfully tested on MOD, MDU, and SGK, and the reported simulation results have confirmed its effectiveness.

In this paper, it is seen that the method composed of (7) and (8) is not the only way of using the main idea (6), and there is actually a significantly better choice, in terms of both performance and speed.

The rest of the paper is organized as follows. Section II presents the main idea and the resulting algorithm. Then, Section III evaluates the new algorithm numerically and compare it with previous ones.

II. The main idea

When solving (6) using alternating minimization, there are several scenarios to set the parameters that are fixed during minimizations over $\mathbf{D}$ and $\mathbf{X}$. In [5] only one case has been introduced and used. However, it will be seen that the way these fixed parameters are set has an important impact on the performance of the algorithms. There are in general four different cases to set these parameters as summarized below:

1) **X-update:** $\mathbf{D} = \mathbf{D}_0 = \mathbf{D}^{(k)}$. $\mathbf{X}_0$ has no effect.
   D-update: $\mathbf{X} = \mathbf{X}_0 = \mathbf{X}^{(k+1)}$. $\mathbf{D}_0$ has no effect.
2) **X-update:** $\mathbf{D} = \mathbf{D}^{(k)}$, $\mathbf{D}_0 = \mathbf{D}^{(k-1)}$. $\mathbf{X}_0 = \mathbf{X}^{(k)}$.
   D-update: $\mathbf{X} = \mathbf{X}^{(k+1)}, \mathbf{X}_0 = \mathbf{X}^{(k)}$. $\mathbf{D}_0 = \mathbf{D}^{(k)}$.
3) **X-update:** $\mathbf{D} = \mathbf{D}_0 = \mathbf{D}^{(k)}$. $\mathbf{X}_0$ has no effect.
   D-update: $\mathbf{X} = \mathbf{X}^{(k+1)}$, $\mathbf{X}_0 = \mathbf{X}^{(k)}$. $\mathbf{D}_0 = \mathbf{D}^{(k)}$.
4) **X-update:** $\mathbf{D} = \mathbf{D}^{(k)}$, $\mathbf{D}_0 = \mathbf{D}^{(k-1)}$. $\mathbf{X}_0 = \mathbf{X}^{(k)}$.
   D-update: $\mathbf{X} = \mathbf{X}_0 = \mathbf{X}^{(k+1)}$. $\mathbf{D}_0$ has no effect.

The first case reduces to the traditional DL problem described by (2) and (3). The third case is the one used in [5], which is described by (7) and (8). The second and fourth cases are being considered in this paper.

Note that the above approaches can be applied on almost any DL algorithm and obtain new ones. To call the resulted algorithms, we add prefixes 'UD1' to 'UD4' to the name of the original algorithm, corresponding to the cases 1 to 4, respectively ('UD' stands for 'UpDated'). For example, applying these cases on MOD, there will be four algorithms: UD1-MOD (same as the original MOD), UD3-MOD (called 'NewMOD' in [5]), and UD2-MOD and UD4-MOD which are new. As it will be seen in the simulations of Section III, 'UD4' algorithms significantly outperform the others, hence, it will be referred also as 'the proposed algorithm', and is summarized in Algorithm 1. In this pseudo-code, s denotes the number of nonzero entries in each column of X, and we call it 'sparsity parameter'. Note that in this algorithm, the dictionary update step is the same as (3).

Algorithm 1 Proposed DL algorithm

Inputs: $\mathbf{Y}$, $\mathbf{D}_0$, s (sparsity parameter)
Initialization: Set initial dictionary $\mathbf{D}^1 = \mathbf{D}^0$ and $\mathbf{X} = \mathbf{0}$
for $k = 1, 2, \ldots$ **do**
  Set $\mathbf{R} = \mathbf{Y} - (\mathbf{D}^k - \mathbf{D}^{k-1})\mathbf{X}^k$
  Sparse representation: $\mathbf{X}^{k+1} = \text{OMP}(\mathbf{R}, \mathbf{D}^{k-1}, s)$
  Dictionary update:

$$\mathbf{D}^{(k+1)} = \underset{\mathbf{D}\in\mathcal{D}}{\operatorname{argmin}} \ \|\mathbf{Y} - \mathbf{D}\mathbf{X}^{(k+1)}\|_F^2.$$

end for

III. Simulations

In this section, we apply all cases that explained in Section II on MOD, SGK and MDU, to evaluate their performances experimentally. Our simulations were performed in MATLAB R2017b environment on a system with 4.00 GHz i7-6700k CPU and 16 GB RAM, under Microsoft Windows 10 operating system.

As a common practice [1], [4], we generated a random dictionary of size 40×100 by drawing its entries from a Gaussian distribution with zero mean and unit variance, and then its columns were normalized. The training data were then generated by sparse combinations of the atoms, where the position and values of sparse coefficients were chosen randomly. In our simulations, we used three sparsity parameters $s = 5, 10, 15$, with 3000 training data. We then applied all the algorithms on the training data to recover the underlying dictionary. The simulations were repeated 300 times and the averaged results were reported.

The performance measures are root mean square error defined as $\varepsilon_k = \frac{\|\mathbf{Y}-\mathbf{D}^k\mathbf{X}^k\|_F}{\sqrt{mL}}$, and percentage of atom recovery. Assuming that $\mathbf{D}_t$ is the true dictionary and $\mathbf{D}$ is the recovered dictionary, we say that the i-th atom of dictionary $\mathbf{D}$ is successfully recovered if:

$$\min_j(1 - |\mathbf{d}_i{}^T\mathbf{d}_{t,j}|) < 0.01. \quad (9)$$

Average run time of each algorithm in seconds is another criterion for comparison. Table I compares these values for various algorithms. Values for the proposed algorithm ('UD4') are in braces.

With these results in mind, we conclude that our proposed approach results in much better convergence rate in dictionary recovery and decreases the running time to converge. Table II reports the number of iterations that each algorithm needs to achieve a percentage of recovery equal to 85. Those of our algorithms are mentioned in parentheses. According to all the figures and tables, our proposed method has higher convergence rates and lower RMSEs than the other algorithms in all the sparsity parameters. According to Table I and II, if s increases, the difference in the convergence rate and running time between our approach and the other algorithms also

increases. For example, when $s = 15$, UD4-MDU is 232.47s and 598.06s faster than MDU and UD3-MDU, respectively. In accordance with Figs. 1 and 2, UD3-MOD, UD3-MDU, UD3-SGK, and UD2-MDU all have low convergence rates at higher s compared to MOD, MDU, SGK, and our algorithms. Furthermore, UD2-MOD and UD2-SGK diverge in high s. Moreover, from Table II it is concluded that our algorithms need lower iteration numbers than the other algorithms to converge. This difference is more noticeable when the s increases.

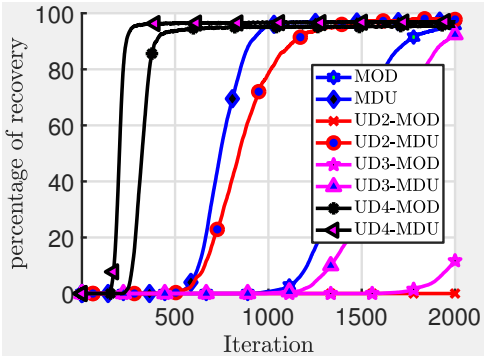

Fig. 1: Percentage of recovery with $s = 15$ and SNR=30.

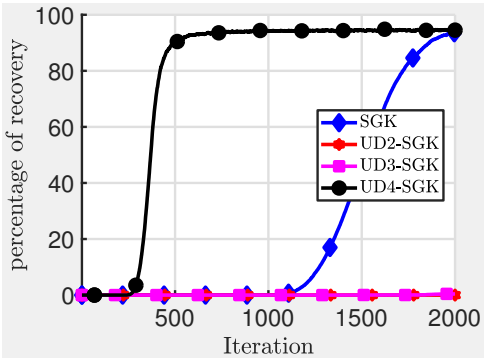

Fig. 2: Percentage of recovery with $s = 15$ and SNR=30.

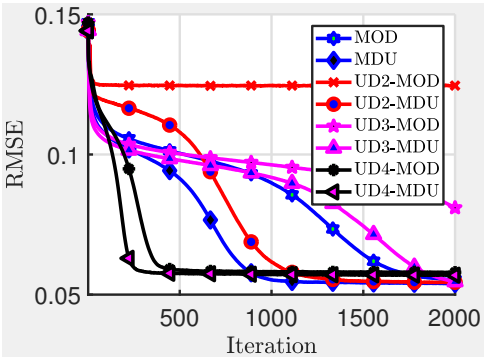

Fig. 3: Root Mean Square Error (RMSE) with $s = 15$ and SNR=30.

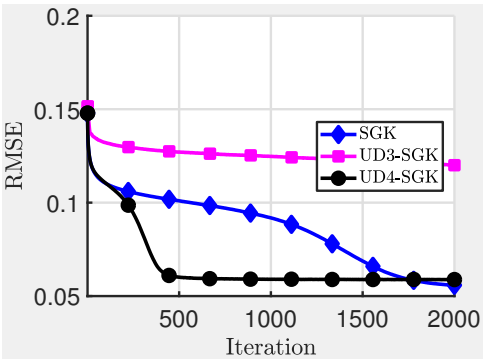

Fig. 4: Root Mean Square Error (RMSE) with $s = 15$ and SNR=30. UD2-SGK method is diverged.

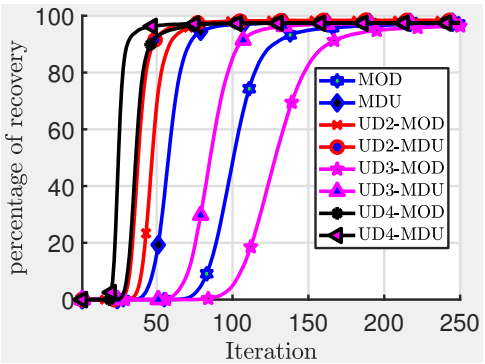

Fig. 5: Percentage of recovery with $s = 10$ and SNR=30.

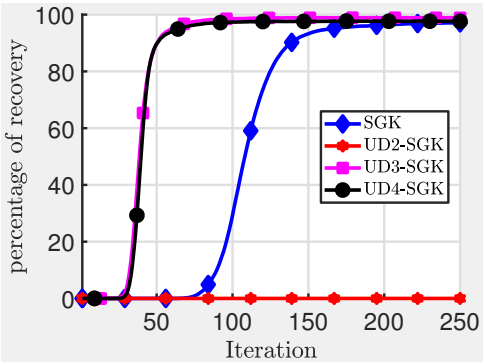

Fig. 6: Percentage of recovery with $s = 10$ and SNR=30.

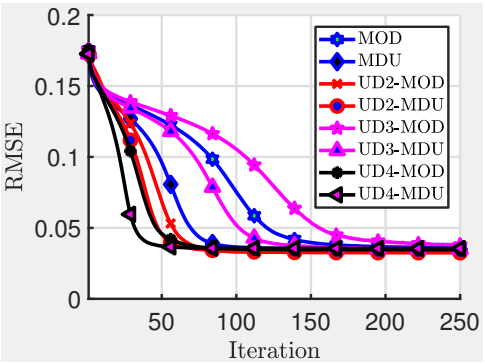

Fig. 7: Root Mean Square Error (RMSE) with $s = 10$ and SNR=30.

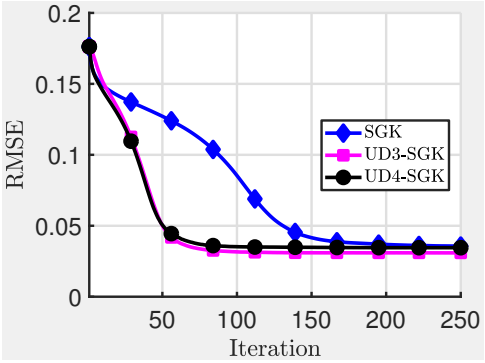

Fig. 8: Root Mean Square Error (RMSE) with $s = 10$ and SNR=30. UD2-SGK method is diverged.

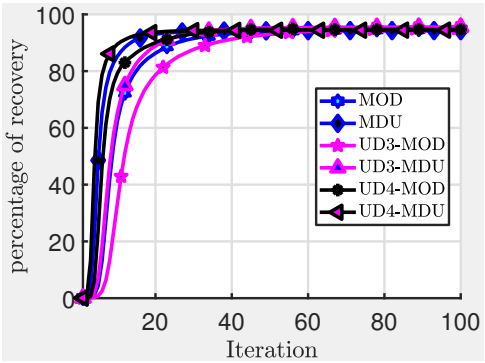

Fig. 9: Percentage of recovery with $s = 5$ and SNR=30.

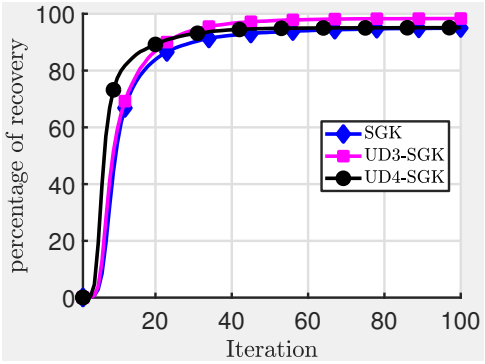

Fig. 10: Percentage of recovery with $s = 5$ and SNR=30.

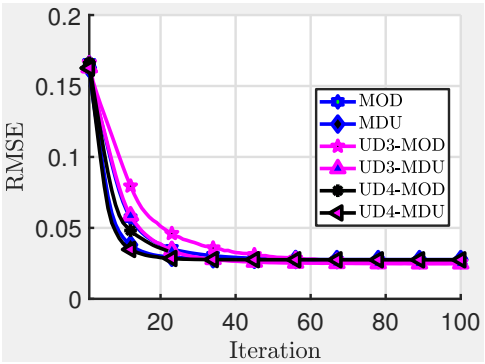

Fig. 11: Root Mean Square Error (RMSE) with $s = 5$ and SNR=30.

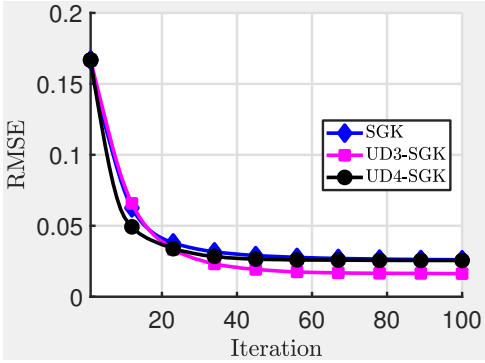

Fig. 12: Root Mean Square Error (RMSE) with $s = 5$ and SNR=30.

TABLE I: Average running times (in seconds) for achieving percentage of recovery=85. Those of our proposed algorithm are reported in parentheses. A dash sign indicates divergence.

Algorithm	$s = 5$	$s = 10$	$s = 15$
MOD	**0.47** (0.36)	**5.7** (2.14)	**109.72** (25.78)
MDU	**2** (1.6)	**21.36** (9.12)	**315.06** (82.59)
SGK	**1** (0.68)	**10.05** (3.68)	**160.93** (39.90)
UD3-MOD	**0.65** (0.36)	**7.41** (2.14)	**172.72** (25.78)
UD3-MDU	**3.41** (1.6)	**21.36** (9.12)	**680.65** (82.59)
UD3-SGK	**0.86** (0.68)	**3.68** (3.68)	– (39.90)

TABLE II: Average number of iterations for achieving percentage of recovery=85. The values shown in parentheses are those of our algorithms. A dash sign indicates divergence.

Algorithm	$s = 5$	$s = 10$	$s = 15$
MOD	**19** (14)	**120** (43)	**1680** (380)
MDU	**10** (8)	**68** (29)	**880** (230)
SGK	**22** (15)	**130** (47)	**1800** (440)
UD3-MOD	**26** (14)	**153** (43)	**2600** (380)
UD3-MDU	**17** (8)	**68** (29)	**1880** (230)
UD3-SGK	**19** (15)	**47** (47)	– (440)
UD2-MOD	**18** (14)	**56** (43)	– (380)
UD2-MDU	**13** (8)	**45** (29)	**1060** (230)
UD2-SGK	– (15)	– (47)	– (440)

IV. Conclusion

In this paper, we showed that the main idea of [5] could actually been used in different ways, and the way it had been used in [5] was not the best one. We then proposed to use another choice that results in a highly better performance, in terms of both accuracy and speed, as confirmed by our simulations. Note that his approach can be applied on almost any existing DL algorithm to obtain modified versions. A theoretical justification for the good performance of the proposed method as well as a convergence proof are subjects for future works.

Appendix

Here, it is shown that the approximation in (5) is actually the first order approximation of the Taylor series of $\mathbf{DX}$. Defining

$\mathbf{H} \triangleq \mathbf{DX}$, the (j,i)-th entry of $\mathbf{H}$ is given by $h_{ji} = \mathbf{d}_{[j]}^T \mathbf{x}_i$, where $\mathbf{d}_{[j]}^T$ denotes the jth row of $\mathbf{D}$ and $\mathbf{x}_i$ denotes the ith column of $\mathbf{X}$. Let us define

$$\mathbf{z} \triangleq \begin{bmatrix} \mathbf{d}_{[j]} \\ \mathbf{x}_i \end{bmatrix}, \mathbf{Q} \triangleq \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix}, \tag{10}$$

in which $\mathbf{I}$ and $\mathbf{0}$ are the identity and zero matrices of appropriate sizes, respectively. The first order Taylor series expansion of $f_{ij}(\mathbf{z}) \triangleq \frac{1}{2}\mathbf{z}^T\mathbf{Q}\mathbf{z} = \mathbf{d}_{[j]}^T\mathbf{x}_i$ around a point $\mathbf{z}_0 = [\mathbf{d}_{[j0]}^T, \mathbf{x}_{i0}^T]^T$ is calculated as $f_{ij}(\mathbf{z}) \approx f_{ij}(\mathbf{z}_0) + \nabla f_{ij}(\mathbf{z}_0)^T(\mathbf{z} - \mathbf{z}_0)$, or equivalently:

$$\begin{aligned} \frac{1}{2}\mathbf{z}^T\mathbf{Q}\mathbf{z} &\approx \frac{1}{2}\mathbf{z}_0^T\mathbf{Q}\mathbf{z}_0 + (\mathbf{z} - \mathbf{z}_0)^T\mathbf{Q}\mathbf{z}_0 \\ &= \mathbf{d}_{[j]}^T\mathbf{x}_{i0} + \mathbf{d}_{[j0]}^T\mathbf{x}_i - \mathbf{d}_{[j0]}^T\mathbf{x}_{i0}. \end{aligned} \tag{11}$$

Writing the above Taylor series expansion for all the elements of $\mathbf{H}$ results in the approximation in (5).

REFERENCES

[1] M. Elad, *Sparse and Redundant Representations*, Springer, 2010.

[2] R. Rubinstein, A. M. Bruckstein, and M. Elad, "Dictionaries for sparse representation modeling," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 1045–1057, 2010.

[3] J. Mairal, F. Bach, and J. Ponce, "Task-driven dictionary learning," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 791–804, 2012.

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

[5] M. Sadeghi, M. Babaie-Zadeh, and C. Jutten, "Dictionary learning for sparse representation: A novel approach," *IEEE Signal Proc. Letters*, vol. 20, no. 12, pp. 1195–1198, 2013.

[6] M. Sadeghi, M. Babaie-Zadeh, and C. Jutten, "Learning over-complete dictionaries based on atom-by-atom updating," *IEEE Trans. on Signal Proc.*, vol. 62, no. 4, pp. 883–891, 2014.

[7] I. Tosic and P. Frossard, "Dictionary learning: What is the right representation for my signal?," *IEEE Signal Processing Magazine*, vol. 28, no. 2, pp. 27–38, 2011.

[8] J. A. Tropp and S. J. Wright, "Computational methods for sparse solution of linear inverse problems," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 948–958, 2010.

[9] H. Mohimani, M. Babaie-Zadeh, and Ch. Jutten, "A fast approach for overcomplete sparse decomposition based on smoothed ℓ^0 norm," *IEEE Trans. on Signal Processing*, vol. 57, pp. 289–301, 2009.

[10] T. Blumensath and M. E. Davies, "Iterative hard thresholding for compressed sensing," *Applied and Computational Harmonic Analysis*, vol. 27, no. 3, pp. 265–274, 2009.

[11] I. Daubechies, R. DeVore, M. Fornasier, and C. S. Güntürk, "Iteratively re-weighted least squares minimization for sparse recovery," *Communications on Pure and Applied Mathematics*, vol. 63, no. 1, pp. 1–38, Jan. 2010.

[12] J. A. Tropp and A. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Trans. Info. Theory*, vol. 53, no. 12, pp. 4655–4666, 2007.

[13] K. Engan, S. O. Aase, and J. Hakon Husoy, "Method of optimal directions for frame design," in *Proceedings of IEEE ICASSP*, 1999, vol. 5.

[14] S. K. Sahoo and A. Makur, "Dictionary training for sparse representation as generalization of K-Means clustering," *IEEE Signal Proc. Letters*, vol. 20, no. 6, pp. 587–590, 2013.

[15] 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, 2013.