Bag of Pursuits and Neural Gas for Improved Sparse Coding

Kai Labusch, Erhardt Barth, and Thomas Martinetz

University of Lübeck
Institute for Neuro- and Bioinformatics
Ratzeburger Allee 160
23562 Lübeck, Germany {*labusch,barth,martinetz*}*@inb.uni-luebeck.de*

Abstract. Sparse coding employs low-dimensional subspaces in order to encode high-dimensional signals. Finding the optimal subspaces is a difficult optimization task. We show that stochastic gradient descent is superior in finding the optimal subspaces compared to MOD and K-SVD, which are both state-of-the art methods. The improvement is most significant in the difficult setting of highly overlapping subspaces. We introduce the so-called "Bag of Pursuits" that is derived from Orthogonal Matching Pursuit. It provides an improved approximation of the optimal sparse coefficients, which, in turn, significantly improves the performance of the gradient descent approach as well as MOD and K-SVD. In addition, the "Bag of Pursuits" allows to employ a generalized version of the Neural Gas algorithm for sparse coding, which finally leads to an even more powerful method.

Keywords: sparse coding, neural gas, dictionary learning, matching pursuit

1 Introduction

Many tasks in signal processing and machine learning can be simplified by choosing an appropriate representation of given data. There are a number of desirable properties one wants to achieve, e.g., coding-efficiency, resistance against noise, or invariance against certain transformations. In machine learning, finding a good representation of given data is an important first step in order to solve classification or regression tasks.

Suppose that we are given data $X = (\mathbf{x}_1, \ldots, \mathbf{x}_L)$, $\mathbf{x}_i \in \mathbb{R}^N$. We want to represent X as a linear combination of some dictionary C, i.e., $\mathbf{x}_i = C\mathbf{a}_i$, where $C = (\mathbf{c}_1, \ldots, \mathbf{c}_M)$, $\mathbf{c}_l \in \mathbb{R}^N$. In case of $M > N$, the dictionary is overcomplete.

In this work, we consider the following framework for dictionary design: We are looking for a dictionary C that minimizes the representation error

$$E_h = \frac{1}{L} \sum_{i=1}^{L} \|\mathbf{x}_i - C\mathbf{a}_i\|_2^2 \tag{1}$$

where $\mathbf{x}_i^{\mathrm{opt}} = C\mathbf{a}_i$ with $\mathbf{a}_i = \arg\min_{\mathbf{a}} \|\mathbf{x}_i - C\mathbf{a}\|$,$\|\mathbf{a}\|_0 \leq k$ denotes the best k-term representation of $\mathbf{x}_i$ in terms of C. The number of dictionary elements

M and the maximum number of non-zero entries k are user-defined model parameters.

It has been shown that finding $\mathbf{a}_i$ is in general NP-hard (Davis et al. (1997)). Methods such as Orthogonal Matching Pursuit (OMP) (Pati et al. 1993) or Optimized Orthogonal Matching Pursuit (OOMP, Rebollo-Neira and Lowe (2002)) can be used in order to find an approximation of the coefficients of the best k-term representation. The Method of Optimal Directions (MOD, Engan et al. (1999)) and the K-SVD algorithm (Aharon et al. (2006)) can employ an arbitrary approximation method for the coefficients in order to learn a dictionary from the data. First, the coefficients are determined, then they are considered fixed in order to update the dictionary.

Using data that actually was generated as a sparse linear combination of some given dictionary, it has been shown (Aharon et al. (2006)) that methods such as MOD or K-SVD can be used in order to reconstruct the dictionary only from the data even in highly overcomplete settings under the presence of strong noise. However, our experiments show that even K-SVD which performed best in (Aharon et al. (2006)) requires highly sparse linear combinations for good dictionary reconstruction performance. The SCNG algorithm does not possess this deficiency (Labusch et al. (2009)), but, unlike MOD or K-SVD, it is bound to a specific approximation method for the coefficients, i.e., OOMP.

Here, we propose a new method for designing overcomplete dictionaries that performs well even if the linear combinations are less sparse. Like MOD or K-SVD, it can employ an arbitrary approximation method for the coefficients. In order to demonstrate the performance of the method, we test it on synthetically generated overcomplete linear combinations of known dictionaries and compare the obtained performance against MOD and K-SVD.

2 From vector quantization to sparse coding

Vector quantization learns a representation of given data in terms of so-called codebook vectors. Each given sample is encoded by the closest codebook vector. Vector quantization can be understood as a special case of sparse coding where the coefficients are constrained by $\|\mathbf{a}_i\|_0 = 1$ and $\|\mathbf{a}_i\|_2 = 1$, i.e., vector quantization looks for a codebook C that minimizes (1) choosing the coefficients according to

$$(\mathbf{a}_i)_m = 1,\ (\mathbf{a}_i)_l = 0\ \forall l \neq m \quad \text{where} \quad m = \arg\min_l \|\mathbf{c}_l - \mathbf{x}_i\|_2^2 \,. \tag{2}$$

In order to learn a good codebook, many vector quantization algorithms consider only the winner for learning, i.e., the codebook vector $\mathbf{c}_m$ for which $(\mathbf{a}_i)_m = 1$ holds. As a consequence of that type of hard-competitive learning scheme, problems such as bad quantization, initialization sensitivity, or slow convergence can arise.

In order to remedy these problems soft-competitive vector quantization methods such as the NG algorithm (Martinetz et al. (1993)) have been proposed. In the NG algorithm all possible encodings are considered in each learning step, i.e., $\mathbf{a}_i^1, \ldots, \mathbf{a}_i^M$ with $(\mathbf{a}_i^j)_j = 1$. Then, the encodings are sorted according to their reconstruction error

$$\|\mathbf{x}_i - C\mathbf{a}_i^{j_0}\| \leq \|\mathbf{x}_i - C\mathbf{a}_i^{j_1}\| \leq \cdots \leq \|\mathbf{x}_i - C\mathbf{a}_i^{j_p}\| \leq \cdots \leq \|\mathbf{x}_i - C\mathbf{a}_i^{j_M}\| \,. \quad (3)$$

In contrast to the hard-competitive approaches, in each learning iteration, every codebook vector $\mathbf{c}_l$ is updated. The update is weighted according to the rank of the encoding that uses the codebook vector $\mathbf{c}_l$. It has been shown in (Martinetz et al. (1993)) that this type of update is equivalent to a gradient descent on a well-defined cost function. Due to the soft-competitive learning scheme the NG algorithm shows robust convergence to close to optimal distributions of the codebook vectors over the data manifold.

Here we want to apply this ranking approach to the learning of sparse codes. Similar to the NG algorithm, for each given sample $\mathbf{x}_i$, we consider all K possible coefficient vectors $\mathbf{a}_i^j$, i.e., encodings that have at most k nonzero entries. The elements of each $\mathbf{a}_i^j$ are chosen such that $\|\mathbf{x}_i - C\mathbf{a}_i^j\|$ is minimal. We order the coefficients according to the representation error that is obtained by using them to approximate the sample $\mathbf{x}_i$

$$\|\mathbf{x}_i - C\mathbf{a}_i^{j_0}\| < \|\mathbf{x}_i - C\mathbf{a}_i^{j_1}\| < \cdots < \|\mathbf{x}_i - C\mathbf{a}_i^{j_p}\| < \cdots < \|\mathbf{x}_i - C\mathbf{a}_i^{j_K}\| \,. \quad (4)$$

If there are coefficient vectors that lead to the same reconstruction error

$$\|\mathbf{x}_i - C\mathbf{a}_i^{m_1}\| = \|\mathbf{x}_i - C\mathbf{a}_i^{m_2}\| = \cdots = \|\mathbf{x}_i - C\mathbf{a}_i^{m_V}\| \,, \quad (5)$$

we randomly pick one of them and do not consider the others. Note that we need this due to theoretical considerations while in practice this situation almost never occurs. Let $\mathrm{rank}(\mathbf{x}_i, \mathbf{a}_i^j, C) = p$ denote the number of coefficient vectors $\mathbf{a}_i^m$ with $\|\mathbf{x}_i - C\mathbf{a}_i^m\| < \|\mathbf{x}_i - C\mathbf{a}_i^j\|$. Introducing the neighborhood $h_{\lambda_t}(v) = e^{-v/\lambda_t}$, we consider the following modified error function

$$E_s = \sum_{i=1}^{L}\sum_{j=1}^{K} h_{\lambda_t}(\mathrm{rank}(\mathbf{x}_i, \mathbf{a}_i^j, C))\|\mathbf{x}_i - C\mathbf{a}_i^j\|_2^2 \quad (6)$$

which becomes equal to (1) for $\lambda_t \to 0$. In order to minimize (6), we consider the gradient of E_s with respect to C, which is

$$\frac{\partial E_s}{\partial C} = -2\sum_{i=1}^{L}\sum_{j=1}^{K} h_{\lambda_t}(\mathrm{rank}(\mathbf{x}_i, \mathbf{a}_i^j, C))(\mathbf{x}_i - C\mathbf{a}_i^j)\mathbf{a}_i^{jT} + R \quad (7)$$

with

$$R = \sum_{i=1}^{L}\sum_{j=1}^{K} h'_{\lambda_t}(\mathrm{rank}(\mathbf{x}_i, \mathbf{a}_i^j, C))\frac{\partial \mathrm{rank}(\mathbf{x}_i, \mathbf{a}_i^j, C)}{\partial C}\|\mathbf{x}_i - C\mathbf{a}_i^j\|_2^2. \quad (8)$$

In order to show that $R = 0$, we adopt the proof given in (Martinetz et al. (1993)) to our setting. With $\mathbf{e}_i^j = \mathbf{x}_i - C\mathbf{a}_i^j$, we write $\mathrm{rank}(\mathbf{x}_i, \mathbf{a}_i^j, C)$ as

$$\mathrm{rank}(\mathbf{x}_i, \mathbf{a}_i^j, C) = \sum_{m=1}^{K} \theta((\mathbf{e}_i^j)^2 - (\mathbf{e}_i^m)^2) \tag{9}$$

where $\theta(x)$ is the heaviside step function. The derivative of the heaviside step function is the delta distribution $\delta(x)$ with $\delta(x) = 0$ for $x \neq 0$ and $\int \delta(x)dx = 1$. Therefore, we can write

$$R = \sum_{i=1}^{L}\sum_{j=1}^{K} h_{\lambda_t}'(\mathrm{rank}(\mathbf{x}_i, \mathbf{a}_i^j, C))(\mathbf{e}_i^j)^2 \sum_{m=1}^{T}((\mathbf{a}_i^j)^T - (\mathbf{a}_i^m)^T)\delta((\mathbf{e}_i^j)^2 - (\mathbf{e}_i^m)^2) \tag{10}$$

Each term of (10) is non-vanishing only for those $\mathbf{a}_i^j$ for which $(\mathbf{e}_i^j)^2 = (\mathbf{e}_i^m)^2$ is valid. Since we explicitly excluded this case, we obtain $R = 0$. Hence, we can perform a stochastic gradient descent on (6) with respect to C by applying $t = 0, \ldots, t_{\max}$ updates of C using the gradient based learning rule

$$\Delta C = \alpha_t \sum_{j=1}^{K} h_{\lambda_t}(\mathrm{rank}(\mathbf{x}_i, \mathbf{a}_i^j, C))(\mathbf{x}_i - C\mathbf{a}_i^j)\mathbf{a}_j^{i\,T} \tag{11}$$

for a randomly chosen $\mathbf{x}_i \in X$ where $\lambda_t = \lambda_0 \left(\lambda_{\mathrm{final}}/\lambda_0\right)^{\frac{t}{t_{\max}}}$ is an exponentially decreasing neighborhood-size and $\alpha_t = \alpha_0 \left(\alpha_{\mathrm{final}}/\alpha_0\right)^{\frac{t}{t_{\max}}}$ an exponentially decreasing learning rate. After each update has been applied, the column vectors of C are renormalized to one. Then the $\mathbf{a}_i^j$ are re-determined and the next update for C can be performed.

3 A bag of orthogonal matching pursuits

So far, for each training sample $\mathbf{x}_i$, all possible coefficient vectors $\mathbf{a}_i^j$, $j = 1, \ldots, K$ with $\|\mathbf{a}_i^j\|_0 \leq k$ have been considered. K grows exponentially with M and k. Therefore, this approach is not applicable in practice. However, since in (6) all those contributions in the sum for which the rank is larger than the neighborhood-size λ_t can be neglected, we actually do not need all possible coefficient vectors. We only need the first best ones with respect to the reconstruction error.

There are a number of approaches, which try to find the best coefficient vector, e.g., OMP or OOMP. It has been shown that in well-behaved cases they can find at least good approximations (Tropp (2004)). In the following, we extend OOMP such that not only the best but the first K_{user} best coefficients are determined, at least approximately.

OOMP is a greedy method that iteratively constructs a given sample $\mathbf{x}$ out of the columns of the dictionary C. The algorithm starts with $U_n^j = \emptyset$,

$R_0^j = (\mathbf{r}_1^{0,j}, \ldots, \mathbf{r}_M^{0,j}) = C$ and $\boldsymbol{\epsilon}_0^j = \mathbf{x}$. The set U_n^j contains the indices of those columns of C that have been used during the j-th pursuit with respect to $\mathbf{x}$ up to the n-th iteration. R_n^j is a temporary matrix that has been orthogonalized with respect to the columns of C that are indexed by U_n^j. $\mathbf{r}_l^{n,j}$ is the l-th column of R_n^j. $\boldsymbol{\epsilon}_n^j$ is the residual in the n-th iteration of the j-th pursuit with respect to $\mathbf{x}$.

In iteration n, the algorithm looks for that column of R_n^j whose inclusion in the linear combination leads to the smallest residual $\boldsymbol{\epsilon}_{n+1}^j$ in the next iteration of the algorithm, i.e., that has the maximum overlap with respect to the current residual. Hence, with

$$\mathbf{y}_n^j = \left(({\mathbf{r}_1^{n,j}}^T \boldsymbol{\epsilon}_n^j)/\|\mathbf{r}_1^{n,j}\|, \ldots, ({\mathbf{r}_l^{n,j}}^T \boldsymbol{\epsilon}_n^j)/\|\mathbf{r}_l^{n,j}\|, \ldots, ({\mathbf{r}_M^{n,j}}^T \boldsymbol{\epsilon}_n^j)/\|\mathbf{r}_M^{n,j}\| \right) \quad (12)$$

it looks for $l_{\text{win}}(n,j) = \arg\max_{l, l \notin U_n^j} (\mathbf{y}_n^j)_l$. Then, the orthogonal projection of R_n^j to $\mathbf{r}_{l_{\text{win}}(n,j)}^{n,j}$ is removed from R_n^j

$$R_{n+1}^j = R_n^j - (\mathbf{r}_{l_{\text{win}}(n,j)}^{n,j} ({R_n^j}^T \mathbf{r}_{l_{\text{win}}(n,j)}^{n,j})^T)/({\mathbf{r}_{l_{\text{win}}(n,j)}^{n,j}}^T \mathbf{r}_{l_{\text{win}}(n,j)}^{n,j}) \, . \quad (13)$$

Furthermore, the orthogonal projection of $\boldsymbol{\epsilon}_n^j$ to $\mathbf{r}_{l_{\text{win}}(n,j)}^{n,j}$ is removed from $\boldsymbol{\epsilon}_n^j$

$$\boldsymbol{\epsilon}_{n+1}^j = \boldsymbol{\epsilon}_n^j - \left(({\boldsymbol{\epsilon}_n^j}^T \mathbf{r}_{l_{\text{win}}(n,j)}^{n,j})/({\mathbf{r}_{l_{\text{win}}(n,j)}^{n,j}}^T \mathbf{r}_{l_{\text{win}}(n,j)}^{n,j}) \right) \mathbf{r}_{l_{\text{win}}(n,j)}^{n,j} \, . \quad (14)$$

The algorithm stops if $\|\boldsymbol{\epsilon}_n^j\| = 0$ or $n = k$. The j-th approximation of the coefficients of the best k-term approximation, i.e., $\mathbf{a}^j$, can be obtained by recursively tracking the contribution of each column of C that has been used during the iterations of pursuit j. In order to obtain a set of approximations $\mathbf{a}^1, ..., \mathbf{a}^{K_{\text{user}}}$, where K_{user} is chosen by the user, we want to conduct K_{user} matching pursuits. To obtain K_{user} different pursuits, we implement the following function:

$$Q(l,n,j) = \begin{cases} 0: \begin{array}{l} \text{If there is no pursuit among all pursuits that have} \\ \text{been performed with respect to } \mathbf{x} \text{ that is equal to} \\ \text{the } j\text{-th pursuit up to the } n\text{-th iteration where in} \\ \text{that iteration column } l \text{ has been selected} \end{array} \\ 1: \text{else} \, . \end{cases} \quad (15)$$

Then, while a pursuit is performed, we track all overlaps $\mathbf{y}_n^j$ that have been computed during that pursuit. For instance if $\mathbf{a}^1$ has been determined, we have $\mathbf{y}_0^1, \ldots, \mathbf{y}_n^1, \ldots, \mathbf{y}_{s_1-1}^1$ where s_1 is the number of iterations of the 1st pursuit with respect to $\mathbf{x}$. In order to find $\mathbf{a}^2$, we now look for the largest overlap in the previous pursuit that has not been used so far

$$n_{\text{target}} = \arg \max_{n=0,\ldots,s_1-1} \max_{l, Q(l,n,j)=0} (\mathbf{y}_n^1)_l \quad (16)$$

$$l_{\text{target}} = \arg \max_l (\mathbf{y}_{n_{\text{target}}}^1)_l \, . \quad (17)$$

We replay the 1st pursuit up to iteration n_{target}. In that iteration, we select column l_{target} instead of the previous winner and continue with the pursuit until the stopping criterion has been reached. If m pursuits have been performed, among all previous pursuits, we look for the largest overlap that has not been used so far:

$$j_{\text{target}} = \arg \max_{j=1,\ldots,m} \max_{n=0,\ldots,s_j-1} \max_{l,Q(l,n,j)=0} (\mathbf{y}_n^j)_l \tag{18}$$

$$n_{\text{target}} = \arg \max_{n=0,\ldots,s_{j_{\text{target}}}-1} \max_{l,Q(l,n,j_{\text{target}})=0} (\mathbf{y}_n^{j_{\text{target}}})_l \tag{19}$$

$$l_{\text{target}} = \arg \max_{l,Q(l,n_{\text{target}},j_{\text{target}})=0} (\mathbf{y}_{n_{\text{target}}}^{j_{\text{target}}})_l \ . \tag{20}$$

We replay pursuit j_{target} up to iteration n_{target}. In that iteration, we select column l_{target} instead of the previous winner and continue with the pursuit until the stopping criterion has been reached. We repeat this procedure until K_{user} pursuits have been performed.

4 Experiments

In the experiments we use synthetic data that actually can be represented as sparse linear combinations of some dictionary. We perform the experiments in order to asses two questions: (i) How good is the target function (1) minimized? (ii) Is it possible to obtain the generating dictionary only from the given data?

In the following $C^{\text{true}} = (\mathbf{c}_1^{\text{true}}, \ldots, \mathbf{c}_{50}^{\text{true}}) \in \mathbb{R}^{20\times 50}$ denotes a synthetic dictionary. Each entry of C^{true} is uniformly chosen in the interval $[-0.5, 0.5]$. Furthermore, we set $\|\mathbf{c}_l^{\text{true}}\| = 1$. Using such a dictionary, we create a training set $X = (\mathbf{x}_1, \ldots, \mathbf{x}_{1500})$, $\mathbf{x}_i \in \mathbb{R}^{20}$ where each training sample $\mathbf{x}_i$ is a sparse linear combination of the columns of the dictionary:

$$\mathbf{x}_i = C^{\text{true}} \mathbf{b}_i \ . \tag{21}$$

We choose the coefficient vectors $\mathbf{b}_i \in \mathbb{R}^{50}$ such that they contain k non-zero entries. The selection of the position of the non-zero entries in the coefficient vectors is performed according to three different data generation scenarios:

Random dictionary elements: In this scenario all combinations of k dictionary elements are possible. Hence, the position of the non-zero entries in each coefficient vector $\mathbf{b}_i$ is uniformly chosen in the interval $[1, \ldots, 50]$.

Independent Subspaces: In this case the training samples are located in a small number of k-dimensional subspaces. We achieve this by defining $\lfloor 50/k \rfloor$ groups of dictionary elements, each group containing k randomly selected dictionary elements. The groups do not intersect, i.e., each dictionary element is at most member of one group. In order to generate a training sample, we uniformly choose one group of dictionary elements and obtain the training sample as a linear combination of the dictionary elements that belong to the selected group.

Dependent subspaces: In this case, similar to the previous scenario, the training samples are located in a small number of k-dimensional subspaces. In contrast to the previous scenario, the subspaces do highly intersect, i.e., the subspaces share basis vectors. In order to achieve this, we uniformly select $k-1$ dictionary elements. Then, we use $50-k+1$ groups of dictionary elements where each group consists of the $k-1$ selected dictionary elements plus one further dictionary element. Again, in order to generate a training sample, we uniformly choose one group of dictionary elements and obtain the training sample as a linear combination of the dictionary elements that belong to the selected group.

The value of the non-zero entries is always chosen uniformly in the interval $[-0.5, 0.5]$.

We apply MOD, K-SVD and the stochastic gradient descent method that is proposed in this paper to the training data. Let $C^{\text{learned}} = (\mathbf{c}_1^{\text{learned}}, \ldots, \mathbf{c}_{50}^{\text{learned}})$ denote the dictionary that has been learned by one of these methods on the basis of the training samples. In order to measure the performance of the methods with respect to the minimization of the target function, we consider

$$E_h = \frac{1}{1500} \sum_{i=1}^{1500} \|\mathbf{x}_i - C^{\text{learned}} \mathbf{a}_i\|_2^2 \tag{22}$$

where $\mathbf{a}_i$ is obtained from the best pursuit out of $K_{\text{user}} = 20$ pursuits that have been performed according to the approach described in section 3. In order to asses if the true dictionary can be reconstructed from the training data, we consider the mean maximum overlap between each element of the true dictionary and the learned dictionary:

$$MMO = \frac{1}{50} \sum_{l=1}^{50} \max_{k=1,\ldots,50} |\mathbf{c}_l^{\text{true}} \mathbf{c}_k^{\text{learned}}| \,. \tag{23}$$

k, the number of non-zero entries is varied from 1 to 11. For the stochastic gradient descent method, we perform 100×1500 update steps, i.e., 100 learning epochs. For MOD and K-SVD, we perform 100 learning iterations each iteration using 1500 training samples. We repeat all experiments 50 times and report the mean result over all experiments.

In the first set of experiments, for all dictionary learning methods, i.e., MOD, K-SVD, and stochastic gradient descent, a single orthogonal matching pursuit is used in order to obtain the dictionary coefficients during learning. Furthermore, the stochastic gradient descent is performed in hard-competitive mode, which uses a neighborhood-size that is practically zero, i.e., $\lambda_0 = \lambda_{\text{final}} = 10^{-10}$. The results of this experiment are depicted in table 1 (a)-(c) and (j)-(l). In case of the random dictionary elements scenario (see (a) and (j)) the stochastic gradient approach clearly outperforms MOD and K-SVD. From the mean maximum overlap it can be seen that almost all dictionary elements are well reconstructed with up to 6 non-zero coefficients in the linear

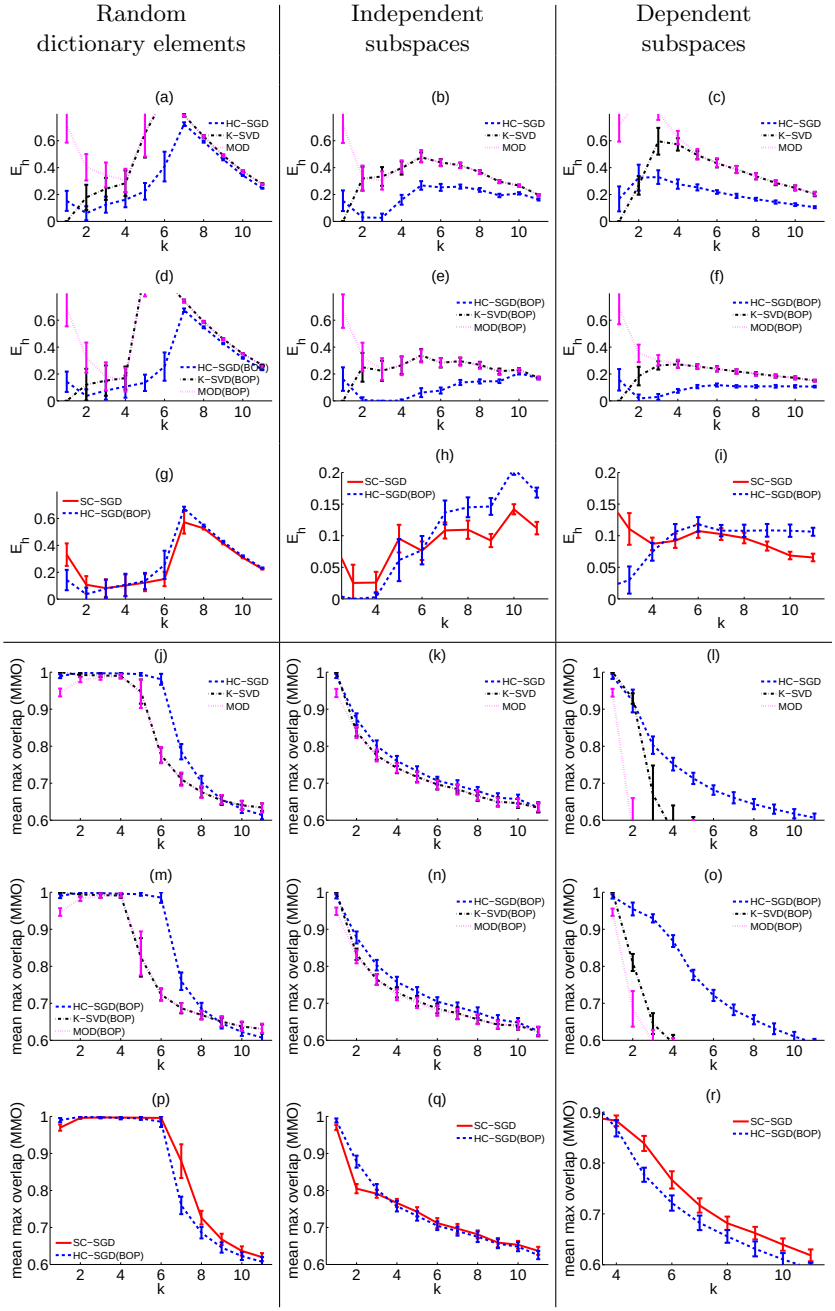

Table 1. Experimental results. See text for details. SC-SGD: soft-competitive stochastic gradient descent ($\lambda_0 = 20, \lambda_{\text{final}} = 0.65$). HC-SGD: hard-competitive stochastic gradient descent ($\lambda_0 = \lambda_{\text{final}} = 10^{-10}$).

combinations. If the dictionary elements cannot be reconstructed any more, i.e., for $k > 6$, the mean representation error E_h starts to grow. In case of the independent subspaces and dependent subspaces scenario the stochastic gradient method also outperforms MOD and K-SVD in terms of minimization of the representation error (see (b) and (c)) whereas in terms of dictionary reconstruction performance only in the intersecting subspaces scenario a clear performance gain compared to MOD and K-SVD can be seen ((k) and (l)). This might be caused by the fact that in case of the independent subspaces scenario it is sufficient to find dictionary elements that span the subspaces where the data is located in order to minimize the target function, i.e., the scenario does not force the method to find the true dictionary elements in order to minimize the target function.

In the second experiment for all methods the dictionary coefficients were obtained from the best pursuit out of $K_{\text{user}} = 20$ pursuits that were performed according to the "Bag of Pursuits" approach described in section 3. The results of this experiment are depicted in table 1 (d)-(f) and (m)-(o). Compared to the results of the first experiment (see (a)-(c)) it can be seen that the computationally more demanding method for the approximation of the best coefficients leads to a significantly improved performance of MOD, K-SVD and the stochastic gradient descent with respect to the minimization of the representation error E_h (see (d)-(f)). The most obvious improvement can be seen in case of the dependent subspaces scenario where also the dictionary reconstruction performance significantly improves (see (l) and (o)). In the random dictionary elements (see (j) and (m)) and independent subspaces scenario (see (k) and (n)) there are only small improvements with respect to the reconstruction of the true dictionary.

In the third experiment, we employed soft-competitive learning in the stochastic gradient descend, i.e., the coefficients corresponding to each of the $K_{\text{user}} = 20$ pursuits were used in the update step according to (11) with $\lambda_0 = 20$ and $\lambda_{\text{final}} = 0.65$. The results of this experiment are depicted in table 1 (g)-(i) and (p)-(r). It can be seen that for less sparse scenarios, i.e. $k > 6$, the soft-competitive learning further improves the performance. Particularly in case of the dependent subspaces scenario a significant improvement in terms dictionary reconstruction performance can be seen for $k > 4$ (see (r)). For very sparse settings, i.e. $k \leq 4$, the hard-competitive approach seems to perform better than the soft-competitive variant. Again, in case of the independent subspaces only the representation error decreases (see (h)) whereas no performance gain for the dictionary reconstruction can be seen (see (q)). Again this might be caused by the fact that in case of the subspace scenario learning of dictionary elements that span the subspaces is sufficient in order to minimize the target function.

5 Conclusion

We proposed a stochastic gradient descent method that can be used either for hard-competitive or soft-competitive learning of sparse codes. We introduced the so-called "bag of pursuits" in order to compute a better estimation of the best k-term approximation of given data. This method can be used together with a generalization of the neural gas approach to perform soft-competitive stochastic gradient learning of (overcomplete) dictionaries for sparse coding.

Our experiments on synthetic data show that compared to other state-of-the-art methods such as MOD or K-SVD a significant performance improvement in terms of minimization of the representation error as well as in terms of reconstruction of the true dictionary elements that were used to generate the data can be observed. While a significant performance gain is already obtained by hard-competitive stochastic gradient descend an even better performance is obtained by using the "bag of pursuits" and soft-competitive learning. In particular, as a function of decreasing sparseness, the performance of the method described in this paper degrades much slower than that of MOD and K-SVD. This should improve the design of overcomplete dictionaries also in more complex settings.

References

AHARON, M., ELAD, M. and BRUCKSTEIN. A. (2006): K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *IEEE Transactions on Signal Processing,[see also IEEE Transactions on Acoustics, Speech, and Signal Processing]*, 54(11):4311–4322.

DAVIS, G., MALLAT, S. and AVELLANEDA. M. (1997): Greedy adaptive approximation. *J. Constr. Approx.*, 13:57–89.

K. ENGAN, K., AASE, S. O. and HAKON HUSOY., J. (1999): Method of optimal directions for frame design. In *ICASSP '99: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 2443–2446, Washington, DC, USA, 1999. IEEE Computer Society.

LABUSCH, K., BARTH, E. and MARTINETZ. T. (2009): Sparse Coding Neural Gas: Learning of Overcomplete Data Representations. *Neurocomputing*, 72(7-9):1547–1555.

MARTINETZ, T., BERKOVICH, S. and SCHULTEN. K. (1993): "Neural-gas" Network for Vector Quantization and its Application to Time-Series Prediction. *IEEE-Transactions on Neural Networks*, 4(4):558–569.

PATI, Y., REZAIIFAR, R. and KRISHNAPRASAD. P. (1993): Orthogonal Matching Pursuit: Recursive Function Approximation with Applications to Wavelet Decomposition. *Proceedings of the 27 th Annual Asilomar Conference on Signals, Systems, and Computers.*

REBOLLO-NEIRA, L. and LOWE. D. (2002): Optimized orthogonal matching pursuit approach. *IEEE Signal Processing Letters*, 9(4):137–140.

TROPP. J. A. (2004): Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242.