A Tree Search Approach to Sparse Coding

Rui Rei[1,2], João P. Pedroso[1,2], Hideitsu Hino[3], and Noboru Murata[3]

[1] University of Porto, Faculty of Science
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
<rui.rei@dcc.fc.up.pt> <jpp@fc.up.pt>
[2] INESC Porto
Rua Dr. Roberto Frias, 378, 4200-465 Porto, Portugal
[3] Waseda University, School of Science and Engineering
3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555, Japan
<hideitsu.hino@toki.waseda.jp> <noboru.murata@eb.waseda.ac.jp>

Abstract. Sparse coding is an important optimization problem with numerous applications. In this paper, we describe the problem and the commonly used pursuit methods, and propose a best-first tree search algorithm employing multiple queues for unexplored tree nodes. We assess the effectiveness of our method in an extensive computational experiment, showing its superiority over other methods even for modest computational time.

Keywords: Sparse Coding, Tree Search

1 Introduction

Sparse Coding is an important optimization problem in signal processing, with applications in classification, image processing, etc. Computationally, it is an NP-hard problem [2], meaning that exact methods are inapplicable in medium- and large-scale instances of the problem. Thus, we take a look at some basic, yet extremely fast heuristic methods developed specifically for this problem. Afterwards, we propose a more sophisticated tree search method which will allow considerable improvements, depending on the allowed CPU time: good results are obtained quickly (even when comparing to greedy heuristics), and improved as more time is given to the search.

The paper is structured as follows. Section 2 describes the sparse coding problem and methods commonly used in practice. Section 3 discusses tree search methods and explains our approach to sparse coding. A computational experiment and its results are discussed in Section 4, and concluding remarks and ideas for future research are presented in Section 5.

2 Sparse Coding

In the sparse coding problem, given a dictionary defined as a matrix $D \in \mathbb{R}^{n \times k}$, comprised of k prototype signals or atoms (the columns of D) of dimension

n, and a target signal $Y \in \mathbb{R}^n$, we seek the best approximation of Y using a combination of at most $m < k$ atoms. The approximation can be written as $D \cdot X$, where $X \in \mathbb{R}^k$ is a vector containing the coefficients of the atoms of D in the linear combination, i.e., X_i is the coefficient of atom i (the i-th column-vector in D, denoted as D_i). Since we can use at most m atoms, the number of non-zero entries in X, $\|X\|_0$ (using the L^0 pseudo-norm) must be less or equal to m. So, Y can be written as

$$Y = D \cdot X + R \ , \tag{1}$$

where $R \in \mathbb{R}^n$ is a residual vector, that is, the portion of Y that cannot be represented by the linear combination of atoms $D \cdot X$. The objective of the problem is then to minimize the L^2 (Euclidean) norm of R, since $\|R\|_2 = 0$ means that we have a perfect representation of Y. The problem can be formally defined as follows:

$$\begin{aligned} \text{minimize} \quad & \|Y - D \cdot X\|_2 \\ \text{subject to} \quad & \|X\|_0 \leq m \\ & X \in \mathbb{R}^k \ . \end{aligned} \tag{2}$$

The given problem is NP-hard, which means that there are no known efficient (polynomial) algorithms to solve it exactly. Thus, several approximate methods have been proposed to find good values for the coefficients X within reasonable computational time. Some of these methods are briefly described below.

First, the Matching Pursuit (MP, [3]) algorithm is a greedy heuristic method which starts with an empty solution ($X^0 = \mathbf{0}$ and $R^0 = Y$), and at each iteration $j = 0, 1, \ldots$, the atom D_i with highest correlation with R^j (the current residual), $|D_i \cdot R^j|$, is added to the sparse support (the set of atoms with non-zero coefficients). Its coefficient X_i^{j+1} is then computed such that the updated residual $R^{j+1} = R^j - X_i^j.D_i$ is orthogonal to D_i, i.e., $R^{j+1} \perp D_i$.

Another greedy heuristic, Orthogonal Matching Pursuit (OMP, [4]), improves on MP by keeping the residual orthogonal to all atoms in the sparse support at each iteration. This way, unlike MP, OMP avoids undoing work done in previous iterations. An interesting characteristic of OMP is that, for any given support, the computed coefficients are optimal. That is, given a set of atoms, OMP computes the best possible approximation using those atoms. This characteristic turns sparse coding into a purely combinatorial problem, where one must find a subset of $S \subseteq D$ to represent Y, using the coefficient calculation method employed in OMP to provide optimal coefficients for any given set of atoms.

Both MP and OMP are deterministic algorithms, since the choice of the next atom to include in the support is fixed: the atom with highest correlation with the current residual is always chosen. These can be easily turned into probabilistic algorithms, for instance, by randomly selecting an atom in the case of a tie. However, due to the unlikelihood of ties in the atom selection step, these variants would most often produce the same solutions as their original deterministic counterparts. A third method, Randomized OMP (RandOMP), is a variant of OMP where variability is introduced into the atom selection step (in every iteration, not just for tie-breaking), allowing it to generate a variety of solutions if run multiple times. The variability can be easily exploited by generating a set of solutions and selecting the best of them.

In the next section, we propose a tree search algorithm to find good sparse supports, and later compare it to OMP and RandOMP.

3 Tree Search

For combinatorial problems, one may create a tree where different branches contain alternative decisions. The leaves of such tree correspond to the problem's search space, that is, the set of complete solutions. Thus, completely exploring the tree and finding the leaves with best objective function value corresponds to solving the problem to optimality.

However, this is not always possible due to the size of the search space. For example, in the case of sparse coding, the search space corresponds to all subsets of the k atoms with size less or equal to m. In other words, the number of possible solutions is equal to the number of combinations of k atoms taken m at a time.

Since it is infeasible to completely explore the tree even for medium-sized instances, tree search is commonly stopped after some given limit (e.g. CPU time) has been reached. In this case, since the tree could not be entirely explored, the best solution found may not be optimal, and the order of exploration of the nodes in the tree becomes very important. If promising nodes are explored first, then the chance of obtaining good-quality solutions when the search is stopped increases.

There exist several schemes for the traversal order of tree nodes. Uninformed traversal methods, such as depth-first or breadth-first, are often used, but other methods like best-first, or variants of beam search, may lead to better performance under time limitation. In the next section, we describe the base elements of a tree search specific to sparse coding, and follow with the description of our multi-queue tree search traversal scheme.

3.1 Application to Sparse Coding

A complete decision tree for sparse coding may be derived as follows. Let δ be a node in the tree, S_δ a set corresponding to the sparse support in δ, and R_δ a set containing the remaining atoms in δ, which may be chosen for inclusion in the support in subsequent decisions. Note that δ is a leaf node if $|S_\delta| = m$ or $R_\delta = \emptyset$.

At the root of the tree γ, we have $S_\gamma = \emptyset$ and $R_\gamma = \{1, \ldots, k\}$. Then, given a node δ, an atom a is chosen from R_δ and two child nodes (δ' and δ'') are created. On the first child node, atom a is included in the support and removed from the remaining set, i.e.,

$$\begin{aligned} S_{\delta'} &= S_\delta \cup \{a\} \\ R_{\delta'} &= R_\delta \setminus \{a\} \ . \end{aligned} \tag{3}$$

On the second child node, δ'', atom a is simply removed from the remaining set

$$\begin{aligned} S_{\delta''} &= S_\delta \\ R_{\delta''} &= R_\delta \setminus \{a\} \ . \end{aligned} \tag{4}$$

Using this branching method, we create two subtrees: on the first subtree all solutions will contain atom a, and on the second subtree no solution will contain a. This will be the basic branching scheme used in our tree search algorithm.

3.2 Multi-Queue Tree Search

The proposed tree search scheme is a variant of best-first search which uses multiple queues to hold the tree nodes which are waiting to be explored. The idea of using several queues comes from the fact that sparse supports of different size are not comparable, since the larger support will in general have a smaller residual. To avoid putting together nodes containing supports of different size, a sub-queue is created for each support size from 0 to $m-1$. No queue is created for size m because all such supports are already complete solutions.

Each time the tree search iterates, a node is picked from one of the sub-queues (in round-robin fashion), its two child nodes are generated and checked, to determine if new leaves were reached. Then, non-leaf child nodes are placed in the sub-queue corresponding to the size of their support. The specific position of a node in a sub-queue is determined by its current residual norm: nodes with smaller residuals are placed in positions closer to the front of the sub-queue, meaning that these will be explored before nodes with larger residuals. Note that this results in a per-sub-queue best-first order, using the residual norm of the partial solutions as a scoring criterion.

Using this scheme allows us to quickly find complete solutions, since nodes are taken from all sub-queues in turn, and at least one leaf node is generated from each node in the $(m-1)$-th sub-queue. The selection of the atom to include in the branching step is taken from the OMP heuristic, i.e., the atom with highest correlation with the current residual is chosen. This way, the first solution found by our Multi-Queue Tree Search (MQTS) is identical to the one produced by OMP, which guarantees a good minimum quality level even with very little time. Since nodes are taken from all sub-queues in equal number, the search will not be trapped in a part of the tree as would occur with depth-first and sometimes best-first search. One additional benefit of tree search is that no duplicate solutions are ever analyzed, as opposed to, for example, repeated RandOMP, where the same supports may be selected in different runs.

In the next section, we describe a computational experiment designed to assess the effectiveness of MQTS, comparing it to OMP and repeated RandOMP.

4 Computational Experiment

Image encoding and compression is a common application of sparse coding, traditionally using a fixed predefined dictionary. However, the use of specially designed dictionaries is known to yield better results. In our experiment, we used the K-SVD [1] dictionary learning algorithm to build specific dictionaries for color and greyscale images. K-SVD was run for 30 iterations on two images of each set, using OMP as a pursuit algorithm in its sparse coding step, producing two dictionaries with $k = 500$ atoms, for color and greyscale images, respectively.

After generating the dictionaries, the three pursuit algorithms were run on the images, broken down into manageable patches of 16×16 pixels, with a CPU limit of 1 second per patch. Note that each patch corresponds to an instance of sparse coding. For color images with three channels, $n = 16 \times 16 \times 3 = 768$, while for greyscale images each patch has $n = 256$ since there is only one channel. For both image sets the maximum support size m was set to 10 atoms.

The experiment was run on an Intel Atom 330 1.6GHz dual-core processor with 2GB of main memory. All programs were implemented in Python, version 2.6.5.

Tables 1a and 1b show the total representation error (Frobenius norm) for all color and greyscale images, respectively. The Frobenius norm of a matrix with r rows and c columns A is given by

$$\|A\|_F = \sqrt{\sum_{i=1}^{r} \sum_{j=1}^{c} A_{ij}^2} \ . \tag{5}$$

Considering the set of patches in the original image as a matrix A (each patch being a column in A), and the set of patches in the encoded image as a matrix B, the representation error of the encoded image is then obtained by $\epsilon = \|A - B\|_F$.

The individual image results indicate a consistent improvement of MQTS over both OMP and repeated RandOMP. The repeated RandOMP algorithm also proved to be better than OMP, due to its exploitation of the variability introduced in the atom selection step. As for MQTS, its superior performance even with very little CPU time indicates that the best-first search order is suitable for this problem, and the overhead of maintaining a search tree and the algorithm's additional complexity do not represent a significant burden. Additionally, this overhead should be diluted as CPU time is increased. The complete absence of symmetries in the tree (no repeated solutions) is also an advantage over RandOMP, which should manifest even more with longer run times.

A deeper analysis of the algorithm, for example by comparison with depth-first search, should allow us to conclude whether the multi-queue mechanism to avoid entrapment is effective or not.

5 Conclusion

We propose a tree search algorithm for obtaining good quality solutions to the sparse coding problem. The computational results reveal superior performance of Multi-Queue Tree Search in all test images, despite the very low CPU time budget. The algorithm quickly provides solutions of reasonable quality, improving

Table 1: Representation error on color (left) and greyscale (right) images, given by the Frobenius norm of the difference between original and encoded images.

(a) Results for color images.

Image	**OMP** ϵ	**RandOMP** ϵ	**RandOMP** Impr. %	**MQTS** ϵ	**MQTS** Impr. %
4.1.01	5103.17	5013.60	1.76	4970.13	2.61
4.1.02	4685.48	4620.57	1.39	4560.13	2.68
4.1.03	4692.30	4609.49	1.76	4531.38	3.43
4.1.04	5650.37	5550.54	1.77	5481.10	3.00
4.1.05	6013.83	5912.02	1.69	5814.47	3.31
4.1.06	8877.23	8713.37	1.85	8623.09	2.86
4.1.07	5202.37	5119.77	1.59	5037.36	3.17
4.1.08	6400.62	6293.05	1.68	6230.79	2.65
4.2.01	9491.35	9249.17	2.55	9143.37	3.67
4.2.02	9227.53	9083.93	1.56	9001.29	2.45
4.2.03	10594.51	10531.24	0.60	10501.82	0.87
4.2.04	7642.89	7556.65	1.13	7523.30	1.56
4.2.05	11554.95	11299.07	2.21	11146.34	3.54
4.2.06	14900.17	14640.49	1.74	14475.44	2.85
4.2.07	12692.56	12458.14	1.85	12272.67	3.31
Average	8181.96	8043.41	1.67	7954.18	2.80

(b) Results for greyscale images.

Image	**OMP** ϵ	**RandOMP** ϵ	**RandOMP** Impr. %	**MQTS** ϵ	**MQTS** Impr. %
5.1.09	2178.22	2146.45	1.46	2104.44	3.39
5.1.10	4685.04	4577.79	2.29	4478.13	4.42
5.1.11	1966.47	1910.98	2.82	1870.58	4.88
5.1.12	3030.55	2955.66	2.47	2886.36	4.76
5.1.13	7035.98	6860.73	2.49	6635.06	5.70
5.1.14	3509.27	3428.30	2.31	3343.48	4.72
5.2.08	6319.43	6174.19	2.30	6050.19	4.26
5.2.09	8174.90	7988.63	2.28	7818.44	4.36
5.2.10	6030.22	5976.06	0.90	5912.82	1.95
5.3.01	7743.45	7627.46	1.50	7535.48	2.69
5.3.02	12019.84	11780.98	1.99	11565.14	3.78
7.1.01	3826.14	3746.98	2.07	3671.68	4.04
7.1.02	2879.91	2820.30	2.07	2756.68	4.28
7.1.03	3801.43	3743.87	1.51	3682.09	3.14
7.1.04	3360.94	3290.87	2.08	3223.30	4.10
7.1.05	5357.74	5258.38	1.85	5167.89	3.54
7.1.06	5265.03	5167.05	1.86	5075.77	3.59
7.1.07	4717.24	4636.78	1.71	4564.54	3.24
7.1.08	3202.14	3153.27	1.53	3109.18	2.90
7.1.09	4726.41	4645.52	1.71	4569.80	3.31
7.1.10	3370.25	3286.69	2.48	3229.31	4.18
7.2.01	5602.48	5552.87	0.89	5491.84	1.97
Average	4945.59	4851.35	1.93	4761.01	3.78

as more time is allowed. Its performance gap over repeated Randomized Orthogonal Matching Pursuit should increase as more time is given, since solutions are analyzed exactly once.

Comparing the tree search to other metaheuristics and commercial solvers is an interesting direction for future research. Also, the performance of the algorithm could be radically improved with the use of a lower bound function, since it would allow us to discard branches of the search tree where we would be sure not to find improving solutions.

References

1. Aharon, M., Elad, M., Bruckstein, A.: K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. Signal Processing, IEEE Transactions on 54(11), 4311 –4322 (nov 2006)
2. Davis, G., Mallat, S., Avellaneda, M.: Greedy adaptive approximation. Journal of Constructive Approximation 13, 57–98 (1997)
3. Mallat, S., Zhang, Z.: Matching pursuits with time-frequency dictionaries. Signal Processing, IEEE Transactions on 41(12), 3397 –3415 (dec 1993)
4. Pati, Y., Rezaiifar, R., Krishnaprasad, P.: Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In: Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on. pp. 40 –44 vol.1 (nov 1993)