Research Article

Feasibility Pump Algorithm for Sparse Representation under Laplacian Noise

Florin Ilarion Miertoiu and Bogdan Dumitrescu

Department of Automatic Control and Computers, University Politehnica of Bucharest, 313 Spl. Independen¸tei, 060042 Bucharest, Romania
Correspondence should be addressed to Bogdan Dumitrescu; bogdan.dumitrescu@acse.pub.ro

Received 4 November 2018; Accepted 6 February 2019; Published 24 February 2019

Academic Editor: Benjamin Ivorra

Copyright © 2019 Florin Ilarion Miertoiu and Bogdan Dumitrescu. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The Feasibility Pump is an effective heuristic method for solving mixed integer optimization programs. In this paper the algorithm is adapted for finding the sparse representation of signals affected by Laplacian noise. Two adaptations of the algorithm, regularized and nonregularized, are proposed, tested, and compared against the regularized least absolute deviation (RLAD) model. The obtained results show that the addition of the regularization factor always improves the algorithm. The regularized version of the algorithm also offers better results than the RLAD model in all cases. The Feasibility Pump recovers the sparse representation with good accuracy while using a very small computation time when compared with other mixed integer methods.

1. Problem Formulation

Sparse representations are among the most important linear representation methods. They are directly related to the compressed sensing problem, but can also be used for machine learning, signal or image processing. As presented in [1] different types of algorithms have been proposed. Sparse representations allow the identification of only a few relevant features or atoms, which can be used to represent the signal of interest with high accuracy. They can also help to well separate different types of signals, which makes them appropriate for denoising and classification problems.

The sparse representation of a signal \( y \in \mathbb{R}^m \), using a dictionary \( D \in \mathbb{R}^{m \times n} \), consists in finding a solution \( x \in \mathbb{R}^n \) having the smallest possible number of nonzero coefficients to the system \( y = Dx \). In practice, the equality constraint is relaxed and replaced by the minimization of the data misfit measure \( \| y - Dx \| \). We adopt here the \( l_1 \) norm, although usually the norm used for the error minimization is \( l_2 \). The \( l_1 \) norm is more robust against outliers and is optimal for Laplacian noise [2]. It is also a norm that promotes sparsity (the error between the original signal and the obtained representation has only few significant elements). Bounding the number of nonzero coefficients (denoted \( \| x \|_0 \)) by an imposed sparsity level \( K \), the sparse representation is found by solving

\[
\begin{align*}
\text{minimize} & \quad \| y - Dx \|_1 \\
\text{subject to} & \quad \| x \|_0 \leq K
\end{align*}
\]  

(1)

An alternative formulation of the problem is obtained by minimizing the number of coefficients and requiring the data misfit measure to be under a selected threshold \( u \):

\[
\begin{align*}
\text{minimize} & \quad \| x \|_0 \\
\text{subject to} & \quad \| y - Dx \|_1 \leq u
\end{align*}
\]  

(2)

Because the norm used to bound the number of nonzero coefficients in (1) is \( l_0 \), Mixed Integer Programming (MIP) algorithms such as Branch and Bound and Feasibility Pump can be used [3].

Mixed Integer Programs are optimization problems that contain both integer and continuous variables. The most used MIP algorithms are Branch and Bound [4, 5] and Cutting Planes [6]. Another popular variant is the Branch and Cut [7] algorithm, which combines both algorithms. These algorithms can offer good solutions for problems with
integer constraints, but often they are very time consuming. The Feasibility Pump, initially proposed in [8, 9], is a MIP heuristic method that generates two sequences of solutions, one satisfying the linear constraints and one satisfying the integer constraint. These sequences are generated until a solution that satisfies both conditions is found. The Feasibility Pump starts from an initial solution and then proceeds through several iterations to minimize the distance between the two solutions, first by solving a linear optimization problem with the integer constraints removed in order to obtain the solution that satisfies the linear constraint, then by a rounding step in order to obtain the solution that satisfies the integer constraint. The algorithm is prone to cycling and loops and several approaches have been proposed to avoid this [10–14]. The Feasibility Pump offers much better execution time (several orders of magnitude compared to Branch and Bound or Branch and Cut), but the quality of the solution is not as good; there is no guarantee that the optimum is attained. The Feasibility Pump has been proposed for both linear and nonlinear problems [15, 16].

In order for the Feasibility Pump algorithm to be used, we introduce a binary variable \( b \in \{0, 1\}^n \), which shows whether an atom of the dictionary is used to represent the signal \( y \).

Problem (1) becomes

\[
\begin{align*}
\text{minimize} & \quad \|y - Dx\|_1 \\
\text{subject to} & \quad 1^T_n b \leq K \\
& \quad -Mb \leq x \leq Mb
\end{align*}
\]

As usual, the starting point for the Feasibility Pump algorithm is the solution of problem (3) when \( b \) is a vector with values in the \([0, 1]^n\) box, instead of a binary one. We also introduce an auxiliary variable \( w \in \mathbb{R}^m \) to bound the data misfit measure. The resulting problem is

\[
\begin{align*}
\text{minimize} & \quad 1^T_m w \\
\text{subject to} & \quad 1^T_n b \leq K \\
& \quad -w \leq y - Dx \leq w \\
& \quad -Mb \leq x \leq Mb
\end{align*}
\]

This is a linear programming problem that can be easily solved with one of the many existing algorithms. We have used the MATLAB function linprog, based on the dual simplex algorithm.

This real \( b \) is then rounded to obtain a binary vector \( \tilde{b} \), by setting to 1 the \( K \) values of \( b \) corresponding to the \( K \) largest absolute values of \( x \) and the others to 0. This is in contrast with the usual rounding to the nearest integer used by Feasibility Pump, but is justified by the fact that a smaller representation error is obtained in the long run by taking the maximum allowed number of nonzero coefficients.

In the ideal case where the vector \( b \) is already binary, the algorithm stops. Note that the first constraint of (4) is satisfied; hence the number of coefficients with a value of 1 cannot exceed \( K \). The other stopping condition is when the maximum number of iterations is reached. In this case, \( b \) is rounded as described above and the coefficients of \( x \) that correspond to zero coefficients in \( b \) are also set to zero.

In each iteration of Feasibility Pump, the vector \( b \) and the tentative solution \( x \) are improved by solving

\[
\begin{align*}
\text{minimize} & \quad (1 - \alpha) \Delta(b, \tilde{b}) + \alpha 1^T_m w \\
\text{subject to} & \quad 1^T_n b \leq K \\
& \quad -w \leq y - Dx \leq w \\
& \quad -Mb \leq x \leq Mb
\end{align*}
\]

where \( \Delta(b, \tilde{b}) = \|b - \tilde{b}\|_1 \) and \( \alpha \in (0, 1) \) is a weight. This linear program is also solved with linprog. The iteration step has an objective that combines the representation error with a term that enforces the new solution \( b \) to be near from the current integer vector \( \tilde{b} \), with the aim of making the solution \( b \) nearer from a binary vector. This kind of modification is proposed in [18, 19]. After each iteration, the parameter \( \alpha \) is reduced and the integer condition will weigh more than the error objective. The reduction of \( \alpha \) is done by multiplication with a value \( y \in (0, 1) \):

\[
\alpha \leftarrow y \alpha.
\]

Typically, a large \( y \) gives the smallest error, but at the expense of a larger execution time, while a smaller \( y \) offers faster results, but with a larger error.

2. Algorithm

Feasibility Pump [8] is a MIP algorithm that tries to minimize the difference between the solution of the relaxed linear programming problem (where the integer condition is relaxed) and a solution that satisfies the integer conditions.

Algorithm 1 presents our approach for solving problem (1) via its reformulation (3). In what follows, we explain its steps and the refinements that we have made to standard Feasibility Pump.
Data: Dictionary $D \in \mathbb{R}^{m \times n}$, signal to represent $y \in \mathbb{R}^m$, sparsity level $K \in \mathbb{N}$, maximum number of iterations $\text{Iter}$, weight parameters $\alpha, \gamma, \lambda$ (the latter for SFPreg only).

Result: a feasible solution $x \in \mathbb{R}^n$

Solve (4)/(9) for $x, b \in \mathbb{R}^n$ using a linear programming solver

Use rounding procedure to obtain vector $b$

while number of iterations $\leq \text{Iter}$ do

Solve the problem (5)/(10) for the vectors $x$ and $b$

if $b$ is integer then

return $x$

end

Use rounding procedure to obtain vector $\bar{b}$

if cycle is detected then

Use perturbation (7) on $\bar{b}$

end

Update the value of $\alpha$ using (6)

end

Algorithm 1: Sparse feasibility pump: SFP uses (4) and (5); SFPreg uses (9) and (10).

Although this strategy usually leads towards a binary $b$, it can also lead to cycling, which we consider to occur at iteration $k + 1$ if

(i) the point $b^{k+1}$ has already been visited in previous iterations;

(ii) the distance $\Delta(b^{k+1}, b^k)$ is under a certain threshold.

To break the cycle, a weak perturbation $v \in [0, 1]^n$ is used for the coefficients of $b$:

$$b^{k+1} \leftarrow \frac{1}{2} b^{k+1} + v.$$  (7)

The resulting $\bar{b}$ has real values and is used in the linear program for the next iteration. This is done to induce a switching of the coefficients in the next vector $\bar{b}$, which will have binary values.

We name Sparse Feasibility Pump (SFP) the algorithm as described above.

Because the matrix $D$ can be ill conditioned, a second implementation, named Sparse Feasibility Pump with regularization (SFPreg), is proposed, where the objective has an $l_1$ regularization term added. Regularization terms in the Feasibility Pump are also considered in [20]. For the initial step, instead of (3) we aim to solve

$$\begin{align*}
\text{minimize} & \quad 1_m^T w + \lambda 1_n^T x_r \\
\text{subject to} & \quad 1_n^T b \leq K \\
& \quad -w \leq y - Dx \leq w \\
& \quad x_r \leq Mb \\
& \quad -x_r \leq x \leq x_r
\end{align*}$$  (8)

where $\lambda > 0$ is a given constant. Similar to the transformation of (3) into (4), we implement (8) by introducing a new variable $w$ to bound the error and a variable $x_r$ to bound $x$ and by relaxing the binary constraint on $b$, thus obtaining

$$\begin{align*}
\text{minimize} & \quad 1_m^T w + \lambda 1_n^T x_r \\
\text{subject to} & \quad 1_n^T b \leq K \\
& \quad -w \leq y - Dx \leq w \\
& \quad x_r \leq Mb \\
& \quad -x_r \leq x \leq x_r
\end{align*}$$  (9)

Regularization also changes the form of the iteration step. The parameter $\alpha$ is now used as a weight value for the regularized error. The regularized version of (5) is

$$\begin{align*}
\text{minimize} & \quad (1 - \alpha) \Delta(b, \bar{b}) + \alpha \left(1_m^T w + \lambda 1_n^T x_r\right) \\
\text{subject to} & \quad 1_n^T b \leq K \\
& \quad -w \leq y - Dx \leq w \\
& \quad x_r \leq Mb \\
& \quad -x_r \leq x \leq x_r
\end{align*}$$  (10)

Adding the regularization term does not change the nature of the problem and the same algorithms as for SFP are used.

We note that, unlike the Branch and Bound algorithms from [17], Algorithm 1 is not guaranteed to provide the exact solution of (1). However, it is significantly faster.

3. Results

For our tests, we use randomly generated dictionaries of size $80 \times 200$ with condition numbers of 1000 and 100000; like in
[17], we focus on not so well conditioned matrices because they are more difficult to handle by sparse representation algorithms. The test signals are obtained with $y = Dx_{\text{true}} + u$, where the solutions $x_{\text{true}}$ have $K \in \{5, 7, 9, 11\}$ nonzero coefficients generated randomly following a Gaussian distribution with zero mean and unit variance, in random positions; the noise $u$ is Laplacian and its variance is chosen such that the signal to noise ratios (SNR) have values 10, 20, 30, and $\infty$.

For each condition number, sparsity level $K$, and SNR, we generate 10 distinct dictionaries and solutions $x_{\text{true}}$.

For the computation of the representation error, the relative error

$$ e = \frac{\|Dx - y\|_1}{\|y\|_1} $$

is used (where now $x$ is the computed solution), in accordance with the formulation of the initial problem (1). The name of the algorithm may be added as subscript, like in $e_{RLAD}$.

We have implemented SFP and SFPreg as shown in Algorithm 1. The initial weight $\alpha$ is set to 0.5 and is multiplied by an update factor $\gamma = 0.95$ at each iteration; in [18, 21] the choices $\alpha = 1$ and $\gamma = 0.9$ are used; by taking $\alpha = 0.5$ we aim to start with equal emphasis on the number of nonzero coefficients and on the representation error; this choice appears also to speed up the convergence. The number of iterations $\text{Iter}$ is set to 1000. We run SFPreg with 50 equally spaced values of the regularization parameter $\lambda$ from 0 to 1. The value for which the error is the smallest is considered the best.

Our two algorithms are compared with the regularized least absolute deviation (RLAD) model [2]

$$ \min_{x \in \mathbb{R}^n} \|Dx - y\|_1 + \bar{\lambda} \|x\|_1 $$

(12)

solved with \texttt{linprog}. Note that this is a lasso-style relaxed problem, adapted to the $\ell_1$ error norm. Problem (12) is solved for different values of $\bar{\lambda}$ until the solution $x$ has a number of nonzero coefficients equal or close to $K$. After that, problem (12) is solved without the regularization term, keeping fixed the support already found. Hence, the solution is optimal for that support. This linear programming approach is generically named RLAD.

The algorithms are implemented in MATLAB and tested on a computer with a 6-core 3.4 GHz processor and 32 GB of RAM.

In Figure 1 we show the effect of the regularization factor $\lambda$ in SFPreg on error (11). The values for SFP (which corresponds to $\lambda = 0$) are not shown, since they are much larger (with the exception of the noiseless case); in increasing SNR order, they are 0.51, 0.232, 0.108, and $2.7 \cdot 10^{-12}$. It can be seen that as the SNR is smaller the error reduction is more significant. Although it is hard to give a general recipe on how $\lambda$ should be chosen, it is visible that the error is small for a fairly large range of values. Hence, it is enough to try only few of them and take the one with best error.

Figures 2 and 3 display the difference in relative errors between RLAD and SFPreg for different matrix condition numbers. It can be seen that the differences are significant. In order to obtain the same number of coefficients, RLAD is forced by the regularization to focus more on sparsity than on the error reduction. Both algorithms recover the solution exactly in the noiseless case.
Figure 2: Difference in relative representation errors between RLAD and SFPreg for matrix conditioning of 1000. A positive value means that SFPreg is better.

Figure 3: Difference in relative representation errors between RLAD and SFPreg for matrix conditioning of 100000.
The relative errors (11) for the SFPreg algorithm are displayed in Figures 4 and 5. Obviously, the errors are dependent on the SNR. The error values are consistently at about the noise level, showing that SFPreg has a good behavior.

The relative recovery errors computed with

$$e_{\text{rec}} = \frac{\|x - x_{\text{true}}\|_2}{\|x_{\text{true}}\|_2}$$

(13)
for the SFPreg algorithm are displayed in Figures 6 and 7. The recovery errors are also dependent on the SNR. For low perturbation the signal is recovered with very good accuracy.

The means of the relative errors (11) obtained by running the tests are displayed in Tables 1 and 2. Table 1 shows the results for matrices with a condition number of 1000 and Table 2 for matrices with condition number 100000. The first value in each cell is the relative error for SFP, the second (with larger font) is for SFPreg, and the last is for RLAD. It can be seen that SFPreg has always smaller mean errors
Table 1: Mean relative errors (II) for SFP (top in each cell), SFPr eg (middle), and RLAD (bottom) for matrix conditioning of 1000.

<table>
<thead>
<tr>
<th>K</th>
<th>5</th>
<th>7</th>
<th>9</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNR = 10</td>
<td>0.262</td>
<td>0.379</td>
<td>0.427</td>
<td>0.471</td>
</tr>
<tr>
<td></td>
<td>0.255</td>
<td>0.255</td>
<td>0.248</td>
<td>0.241</td>
</tr>
<tr>
<td></td>
<td>0.447</td>
<td>0.450</td>
<td>0.500</td>
<td>0.471</td>
</tr>
<tr>
<td>SNR = 20</td>
<td>0.084</td>
<td>0.143</td>
<td>0.378</td>
<td>0.338</td>
</tr>
<tr>
<td></td>
<td>0.084</td>
<td>0.084</td>
<td>0.082</td>
<td>0.082</td>
</tr>
<tr>
<td></td>
<td>0.201</td>
<td>0.276</td>
<td>0.410</td>
<td>0.415</td>
</tr>
<tr>
<td>SNR = 30</td>
<td>0.027</td>
<td>0.027</td>
<td>0.095</td>
<td>0.157</td>
</tr>
<tr>
<td></td>
<td>0.027</td>
<td>0.026</td>
<td>0.025</td>
<td>0.025</td>
</tr>
<tr>
<td></td>
<td>0.068</td>
<td>0.231</td>
<td>0.270</td>
<td>0.464</td>
</tr>
<tr>
<td>SNR = ∞</td>
<td>1.814 \cdot 10^{-11}</td>
<td>4.556 \cdot 10^{-12}</td>
<td>1.037 \cdot 10^{-12}</td>
<td>1.161 \cdot 10^{-12}</td>
</tr>
<tr>
<td></td>
<td>5.801 \cdot 10^{-14}</td>
<td>2.646 \cdot 10^{-14}</td>
<td>1.985 \cdot 10^{-14}</td>
<td>1.132 \cdot 10^{-14}</td>
</tr>
<tr>
<td></td>
<td>5.603 \cdot 10^{-10}</td>
<td>6.772 \cdot 10^{-10}</td>
<td>9.165 \cdot 10^{-10}</td>
<td>8.621 \cdot 10^{-10}</td>
</tr>
</tbody>
</table>

compared to RLAD. The SFPr eg approach gives an important improvement, in some cases reducing the error by almost an order of magnitude compared to SFP. Also, SFP has many cases in which the representation error is much larger than for RLAD.

In fact, the SFPr eg algorithm has smaller representation errors in all cases with noise compared to the RLAD algorithm, for both conditioning numbers. It can be seen that the conditioning of the matrix does not affect the methods based on regularization.

Turning now to the complexity of the algorithms, the mean number of iterations is 3.77 for SFP and 3.14 for SFPr eg. So, regularization not only helps the algorithm to give better results but also leads to slightly faster convergence.

The number of iterations depends on the selection of α. A small α makes SFP and SFPr eg converge faster but can cause a higher error. Also the update factor γ affects the speed and accuracy of the algorithms. The values reported above seem to provide a good compromise.

The mean execution time of SFPr eg is 0.416 seconds, while SFP requires 0.348 seconds. This is an important improvement compared to the formulation (4) solved with the Branch and Cut algorithm. As reported in [17], Branch and Cut require between 3 to 10000 seconds to find the solution for the problem. So, SFPr eg is significantly faster and still obtains results that are nearly optimal.

4. Conclusions

In this paper an adaptation of the Feasibility Pump has been proposed to address the sparse representation problem with $l_1$ norm of the error. As seen from the experimental results, the addition of the regularization term offers a considerable improvement to the algorithm. The Feasibility Pump gives a good solution in a very small number of iterations. Thus, the Feasibility Pump proves to be a good algorithm for finding sparse representations of signals. Due to regularization and the big-$M$ trick, it limits the magnitude of the values of the coefficients and is thus appropriate also for ill-conditioned problems. Future research will focus on improving the randomization step, using different regularization terms, and adapting the algorithm for other sparse problems reformulations.

Data Availability

The data used for experiments have been generated as described in the paper, using random numbers. We expect
similar results if the data are generated again using the same method.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References


