Binary Matrix Factorization

Ignacio Ramírez

December 23, 2016

1 introduction

We consider the problem of approximating a binary matrix $X \in \{0,1\}^{m\times n}$ as the product of two other binary matrices $U \in \{0,1\}^{m\times k}$ and $V \in \{0,1\}^{n\times p}$ plus a small residual,

$$X = UV^T + E. \tag{1}$$

Matrix Factorization The problem (1) has been addressed using various methods from different fields for over a hundred years. Two such fields are Signal Processing and Machine Learning under the so-called "Matrix Factorization" (MF) methods, (with non-negative matrix factorization (NMF) a very popular particular case,) which is very mature and has been developed largely for the case of real matrices.

Dictionary Learning A particular case of MF is the so-called "Dictionary Learning" problem which is particularly useful when $m \gg n$. In this case, contrary to the more general MF approach, the roles of U and V are quite different and, accordingly, both are estimated in quite different ways. More specifically, U is called a dictionary (and usually assigned the letter D), whereas $V^T = A$ is a matrix of "linear combination coefficients". The columns of the matrix D are called "atoms" and are supposed to embody typical patterns observed throughout the columns of X, while the columns of A, usually assumed sparse, specify the linear combination of columns of D that better approximates the corresponding columns of X.

Data Mining On the other hand, there is a large body of work from the Data Mining community on the subject of extracting general "patterns" from large matrices, in particular binary matrices. Some of those works are based on matrix factorization concepts, for example the Proximus algorithm [4],although most of them are quite heuristic.

Other ideas Initially I was interested in connecting the above techniques with other learning frameworks aimed at binary data. In particular, the Bi-directional Auto-associative Memory (BAM) model [3] was an early binary recurrent neural network.

Our approach The idea here is very simple: to apply a Dictionary Learning approach to the problem of binary matrix factorization for those cases where the matrix X is significantly "fat" $m \gg n$ (or, naturally, apply it to X^T when it is "tall", $m \ll n$).

2 The Dictionary Learning approach

The problem (1) is generally non-convex. There are very special cases in which it can be solved exactly, or it reduces to a tractable convex problem under particular conditions. Most DL approaches fall into the general setting where (1) is NP-hard and only local convergence (to a stationary point) can be guaranteed. This is usually achieved through *alternate (block) minimization* on D and A,

$$A^{(k+1)} = \arg\min_A \{f(D^{(k)} - A) + g(A)\} \tag{2}$$
$$D^{(k+1)} = \arg\min_D \{f(D - A^{(k+1)}) + g(A^{(k+1)})\}, \tag{3}$$

where $f(\cdot)$ and $g(\cdot)$ are *fitting* and *regularization* functions respectively. The first one is typically the squared ℓ_2 norm, $\|\cdot\|^2$, and the second one is an ℓ_p norm such as $\|\cdot\|_1$, with $0 \leq p \leq 2$ (when $p < 1$ it is not a norm). For that case, this algorithm is known as *Method of Directions* or MOD [2]:

$$A_j^{(k+1)} = \arg\min_{a \in \mathbb{R}^p} \{\|x_j - D^{(k)} a\|_2^2 + \|a\|_1,\ j = 1, \ldots, n\} \tag{4}$$
$$D_r^{(k+1)} = u_j / \|u_j\|_2,\ u_j = X(A^{(k+1)})^T \left(A^{(k+1)}(A^{(k+1)})^T\right)^{-1}\}, \tag{5}$$

where A_j and D_r are the j-th and r-th columns of A and D respectively. The first step corresponds to an ℓ_1-regularized least squares regression problem on each column of A, also known as LASSO [6]. In the second step, each atom in the dictionary corresponds to a normalized down version of the least squares solution u; note that here it is customary as well to apply some sort of regularization so that AA^T is invertible (non-singular).

K-SVD Another popular approach to the dictionary learning problem is the K-SVD method [1]. In this case, $g(\cdot)$ corresponds to the ℓ_0 pseudo-norm, which counts the number of non-zeros in a vector. The updates on A are similar to MOD, but use a greedy method known as OMP (Orthogonal Matching Pursuit) [5] to obtain an approximate solution,

Data: vector to encode x, dictionary D, maximum residual norm ϵ
Result: Optimal coefficients for x, a
Set iteration $k = 0$, residual $r^{(0)} = x$, coefficients $a^{(0)} = 0$;
while $\left\|r^{(k)}\right\| \geq \epsilon$ **do**
    $i = \arg\max \left\{ D_i^T r^{(k)} \right\}$;
    $a_i \leftarrow D_i^T r^{(k)}$;
    $r^{(k+1)} \leftarrow r^{(k)} - a_i D_i$;
    $k \leftarrow k + 1$;
end

What OMP does at each iteration is to project the residual onto the atom that is most correlated to it, and then remove the projection from the residual. For this to work well, the atoms must be normalized to have ℓ_2 norm 1.

The seconds stage, instead of being a block descent on D, updates both D and A (again) using rank-one updates as follows. For each atom D_j to be updated, a residual is first computed which does not take the contribution of D_j into account:

$$E_j^{(k+1)} = X - DA + D_j A^j, \tag{6}$$

where A^j is the j-th *row* of A. Then, both D_j and A^j are updated using the first pair of left and right eigenvectors of the SVD decomposition of $E_j^{(k+1)}$.

$$D_j = U_1, \quad A^j = V^1, \quad E_j^{(k+1)} = U\Sigma V, \tag{7}$$

The K-SVD dictionary step is significantly more costly than that of MOD, but usually requires significantly less iterations. MOD, however, is better suited for online dictionary adaptation, as fast approximations of the statistics AA^T (which can be thought of as the Hessian associated to minimizing D) and XA^T can be efficiently updated on a sample to sample basis.

3 Binary Dictionary Learning

The straightforward approach here is to adapt methods such as MOD or K-SVD to the case of binary matrices. One possibility is to simply use those methods as they are, possibly imposing constraints that enforce the entries in the results to be between 0 and 1. An alternative, more akin to works such as those done in Data Mining with the PROXIMUS algorithn, is to work directly on binary operands, using binary operations.

3.1 Coefficients update

First of all, the ℓ_0 norm and the ℓ_1 norms are the same for binary vectors, so both dictionary update methods from MOD and K-SVD can be treated together. (Actually, the ℓ_0 norm is nothing else than the *Hamming weight* or simply *weight* of a binary vector $h(x)$, so we may call it by that name hereafter). We thus consider the problem of minimizing $h(x_j - Da_j)$. Following the OMP idea of removing the closest atom at each iteration. The problem here is that we do not have normalized vectors, so dot product is not a good idea. Instead, we simply search for the atom that is closest in Hamming distance, and remove the candidate atom from the residual using modulo-2 arithmetic (we use $\oplus$ to denote modulo-2 or XOR addition). The algorithm goes as follows:

Data: vector to encode x, dictionary D
Result: Optimal coefficients for x, a
Set iteration $k = 0$, residual $r^{(0)} = x$, coefficients $a^{(0)} = 0$;
while $hr^{(k)} \geq \epsilon$ **do**
  $h_{\min} \leftarrow\leftarrow \min h(D_i, r^{(k)})$;
  if $h_{\min} > h(r^{(k)})$ **then**
    break
  end
  $i \leftarrow \arg\min h(D_i, r^{(k)})$;
  $a_i \leftarrow a_i \oplus 1$;
  $r^{(k+1)} \leftarrow r^{(k)} \oplus D_i$;
end

Besides the differences mentioned, the algorithm also stops when the Hamming distance between the atom that is closest to the current residual is larger than the Hamming weight of the residual itself, as in such case there is no possible improvement.

3.2 Dictionary update – MOD-like case

Here we want to update each atom so that the Hamming weight of the total residual matrix $E = X \oplus DA$ is minimum (note that minus and plus are the same thing in modulo-2 arithmetic). Say we want to update the r-th atom at iteration k. Clearly, the affected columns will only be those for which the coefficients in A corresponding to that atom are non-zero, that is $\{j : A_{rj} \neq 0\}$; let us call this set J_r and n_r its size. What we want is the update Δ that, when added to D_r, minimizes the weight of the residual E in those colums affected by D_r,

$$\Delta = \arg\min_d \sum_{j \in J_r} h(E_j^{(k)} \oplus d) \tag{8}$$

$$= \arg\min_d \sum_{j \in J_r} (\sum_i E_j^{(k)} + n_r d_i). \tag{9}$$

(note that the summation symbols are carried on using normal addition). The above objective is trivially separable in the elements of d,

$$\Delta_i = \arg\min_{u\in\{0,1\}} \sum_j E_{ij}^{(k)} \oplus d_i \tag{10}$$

$$= \arg\min_{u\in\{0,1\}} \sum_{j\in J_r} E_{ij}^{(k)} + n_r u. \tag{11}$$

From (11) we cleary obtain

$$\Delta_i = \begin{cases} 0, n_r \leq \sum_j E_{ij}^{(k)} \\ 1, n_r > \sum_j E_{ij}^{(k)} \end{cases} \tag{12}$$

In other words, the i-th element of D_r should be 0 if most of the elements in the rows of E affected by it are 0, and 1 otherwise. This is clearly an optimum solution (note that there could be many optima when n_r is even, in which case we simply choose one).

3.3 Dictionary update – K-SVD-like case

In this case, following the K-SVD concept, we want to obtain the best rank-one approximation to the residual E obtained after removing the contribution of D_r,

$$(D_r, A_r) = \arg\min_{u,v} h(E \oplus uv^T) \tag{13}$$

where

$$E = X_{J_r} \oplus D^{(k)} A_{J_r}^{(k)} \oplus D_r^{(k)} (A_{J_r}^{(k)})^r, \tag{14}$$

where X_{J_r} and A_{J_r} contain only the colums in $J_r = \{j : A_{rj} = 1\}$.

The problem (13) is, again, NP-hard, so an approximate solution must be sought. The idea here is to use alternate updates on D_r and A_r, in which case the updates have a simple closed form solution. This is exactly the solution proposed at each step of the PROXIMUS [4] algorithm. Say we want to minimize $h(E \oplus uv^T)$ using alternate updates on u and v. For fixed u, we have

$$\sum_{i,j} E \oplus uv^T = \sum_{j:v_j=1} h(E_j \oplus u).$$

We can solve this separately for each v_j simply by setting $v_j = 1$ is $h(E_j \oplus u) < h(E_j)$ and $v_j = 0$ otherwise. The update on u is identical. In both cases, the objective function can only decrease, the function is bounded and the domain is compact, so that the algorithm must converge to a local minimum. Because the set is finite, convergence is attained in a finite number of iterations. Usually this number is quite small.

4 Initialization

The above algorithms, which I will now call BDL and K-PROX (this is not a definitive name), are quite quite non-convex and nasty. Therefore, a clever initialization is always welcome. Here are some recipes, a few of them taken from the PROXIMUS paper [4],

neighbors Pending. see description on bsvd.cpp

graph grow Pending. see description on bsvd.cpp

partition Pending. see description on bsvd.cpp

random samples D is initialized using random saples, or a sum of a few random samples, from the set.

5 Variants

I implemented several variants of the above algorithms, most of them just change the order in which some updates are made. I'll describe them later.

References

[1] M. Aharon, M. Elad, and A. Bruckstein. k -svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, Nov 2006.

[2] K. Engan, S. Aase, and J. Husoy. Multi-frame compression: Theory and design. *Signal Processing*, 80(10):2121–2140, Oct. 2000.

[3] Bart Kosko. Bidirectional associative memories. *IEEE Trans. Syst. Man Cybern.*, 18(1):49–60, January 1988.

[4] Mehmet Koyutrk and Ananth Grama. PROXIMUS: A framework for analyzing very high dimensional discrete-attributed datasets. In *KDD 2003*, pages 147–156. IEEE Computer Society, 2003.

[5] S. Mallat and Z. Zhang. Matching pursuit in a time-frequency dictionary. 41(12):3397–3415, 1993.

[6] R. Tibshirani. Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society: Series B*, 58(1):267–288, 1996.