Binary Matrix Factorization

Ignacio Ramírez

December 22, 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,

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

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]:

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

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. 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.

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.