SVD Factorization for Tall-and-Fat Matrices on Map/Reduce Architectures

Burak Bayramlı

October 15, 2013

Abstract

We demonstrate an implementation for an approximate rank-k SVD factorization, combining well-known randomized projection techniques with previously implemented map/reduce solutions in order to compute steps of the random projection based SVD procedure, such QR and SVD. We structure the problem in a way that it reduces to Cholesky and SVD factorizations on $k \times k$ matrices computed on a single machine, greatly easing the computability of the problem.

1 Introduction

(intro)

gleich
[1] presents many excellent techniques for utilizing map/reduce architectures to compute QR and SVD for the so-called tall-and-skinny matrices. QR factorization is turned into an A^TA computation problem to be computed in parallel using map/reduce, and its key element the Cholesky decomposition, can be performed on a single machine. Let's use $C = A^TA$ and, since

$$C = A^TA = (QR)^T(QR) = R^TQ^TQR = R^TR$$

and because Cholesky factorization of an $n \times n$ symmetric positive definite matrix is

$$C = LL^T$$

where L is an $n \times n$ lower triangular matrix, and R is upper triangular, we can conclude if we factorize C into L and L^T, this implies $C = LL^T = RR^T$, we have a method of calculating R of QR using Cholesky factorization on A^TA. The key observation here is A^TA computation results an $n \times n$ matrix and if A is "skinny" then n is relatively small (in the thousands), then Cholesky decomposition can be executed on a small $n \times n$ matrix on a single computer utilizing an already available function in a scientific computing library. Q is computed simply as $Q = AR^{-1}$. This again is relatively cheap because R is $n \times n$, the inverse is computed locallly, matrix multiplication with A can be performed through map/reduce.

SVD is an additional step. SVD decomposition is

$$A = U\Sigma V^T$$

If we expand it with $A = QR$

$$QR = U\Sigma V^T$$

$$R = Q^TU\Sigma V^T$$

Let's call $\tilde{U} = Q^T U$

$$R = \tilde{U}\Sigma V^T$$

This means if we run a local SVD on R (we just calculated above with Cholesky) which is an $n \times n$ matrix, we will have calculated $\tilde{U}$, the real Σ, and real V^T.

Now we have a map/reduce way of calculating QR and SVD on $m \times n$ matrices where n is small.

1.1 Approximate rank-k SVD

Switching gears, we look at another method for calculating SVD. The motivation is while computing SVD, if n is large, creating a "fat" matrix which might have columns in the billions would require reducing the dimensionality of the problem. According to [2], one way to achieve is through random projection. First we draw an $n \times k$ Gaussian random matrix Ω. Then we calculate

$$Y = A\Omega$$

We perform QR decomposition on Y

$$Y = QR$$

Then form $k \times n$ matrix

$$B = Q^T A(bt)$$

Then we can calculate SVD on this small matrix

$$B = \hat{U}\Sigma V^T$$

Then form the matrix

$$U = Q\hat{U}$$

The main idea is based on

$$A = QQ^T A$$

if replace Q which comes from random projection Y,

$$A \approx \tilde{Q}\tilde{Q}^T A$$

Q and R of the projection are close to that of A. In the multiplication above R is called B where $B = \tilde{Q}^T A$, and,

$$A \approx \tilde{Q}B$$

then, as in [1], we can take SVD of B and apply the same transition rules to obtain an approximate U of A.

This approximation works because of the fact that projecting points to a random subspace preserves distances between points, or in detail, projecting the n-point subset onto a random subspace of $O(\log n/\epsilon^2)$ dimensions only changes the interpoint distances by $(1 \pm \epsilon)$ with positive probability [3]. It is also said that Y is a good representation of the span of A.

1.2 Combining Both Methods

Our idea was using approximate k-rank SVD calculation steps where $k << n$, and using map/reduce based QR and SVD methods to implement those steps. By utilizing random projection, we would be able to work in a smaller dimension which would translate to local Cholesky, and SVD calls on $k \times k$ matrices that can be performed in a speedy manner. Below we outline each map/reduce job.

```
random_projection_map($A$)
    Tokenize $value$ and pick out id value pairs
    result = zeros(1,k)
    for each $j^{th}$ $token \in value$
        Initialize seed with j
        j = generate $k$ random numbers
        $result = result + r \cdot token[j]$
    emit key, result
```

Reduce is a no-op.

Each value of A will arrive to the algorithm as a key and value pair. Key is line number or other identifier per row of A. Value is a collection of id value pairs where id is column id this time, and value is the value for that column. Sparsity is handled through this format, if an id for a column does not appear in a row of A, it is assumed to be zero. The resulting Y matrix has dimensions $m \times k$.

```
$A^TA$cholesky_job_map(key k,value a)
    for $i, row$ in enumerate$a^T a$
        emit $i, row$
```

```
cholesky_job_reduce($key, value$)
    emit $k$, sum($v_j^k$)
```

The cholesky_job_final_local_reduce step is a function provided in most map/reduce frameworks, it is a central point that collects the output of all reducers, naturally a single machine which makes it ideal to execute the final Cholesky call on by now a very small ($k \times k$) matrix. The output is R.

References

[1] Gleich, Benson, Demmel, *Direct QR factorizations for tall-and-skinny matrices in MapReduce architectures*, arXiv:1301.1071 [cs.DC], 2013

[2] N. Halko, *Randomized methods for computing low-rank approximations of matrices*, University of Colorado, Boulder, 2010

[3] S. Dangupta, A. Gupta *An Elementary Proof of a Theorem of Johnson and Lindenstrauss*, Wiley Periodicals, 2002

```
cholesky_job_final_local_reduce($key, value$)
    $result = $ cholesky($A_{sum}$)
    emit $result$
```

```
Q_job_map($key, value$)
    During initialization $R_{inv} = R^{-1}$, store it once for each mapper
    for $row$ in $Y$
        emit $key, row \cdot R_{inv}$
```

[4] M. Kurucz, A. A. Benczúr, K. Csalogány, *Methods for large scale SVD with missing values*, ACM, 2007

[5] B. Bayramli, *Sasha Framework*, `git@github.com:burakbayramli/sasha.git` Github, 2013

[6] B. Bayramli, *Map/Reduce Code for Netflix SVD Analysis*, `http://github.com/burakbayramli/classnotes/tree/master/stat/stat_mr_rnd_svd/sasha`, Github, 2013