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 randomized procedure, such QR and SVD. We structure the problem in a way reduces to single machine Cholesky and SVD factorizations on $k \times k$ matrices, greatly easing the computability of the problem.

1 Introduction

[1] presents many excellent techniques for utilizing map/reduce architectures to compute QR and SVD for the so-called tall-and-skinny matrices. The ideas are based on the fact that QR factorization can be turned into an A^TA computation problem which in turn reduces to a Cholesky decomposition performed on a single machine. First idea is,

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

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

$$A = LL^T$$

where L is an $n \times n$ lower triangular matrix., and R is upper triangular, then we can conclude if we factorize A into L and L^T, this implies $LL^T = RR^T$, we have a method of calculating QR using Cholesky factorization on A^TA. The key observation here is that after A^TA computation is completed we have an $n \times n$ matrix and if A is "skinny" then n is relatively small (in the thousands), and Cholesky decomposition can be executed on an small $n \times n$ matrix on a single computer. Calculating SVD is an additional step, based on QR. SVD decomposition is represented as

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

If we expand it with $A = QR$

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

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

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. Hence 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 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

$$B = Q^T A$$

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 approximate 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 Final Method

Our final implementation performs approximate k-rank SVD by using QR and SVD methods presented by Gleich, adding new map/reduce jobs when needed such as for random projection, or calculating $A^T Q$. Below we outline each map/reduce job.

Algorithm 1: Random Projection Job

function *MAP(key, value)*
    xxx
    return
function *REDUCE(key, value)*
    xxx
    return

Algorithm 2: Random Projection Job

```
begin
    for each row ∈ A do
        Tokenize row and pick out id value pairs
    end
end
```

References

[1] Gleich, Benson, Demmel, *Direct QR factorizations for tall-and-skinny matrices in MapReduce architectures*

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

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