COL 380 **Apr 01, 2019**

Lab 3

Instructor: Subodh Sharma **Due: Apr 10, 23:55 hrs**

Principal Component Analysis using GPUs

For this lab you have to perform *Principal component analysis*, or PCA for short, (refer Section 3) on a given dataset using CUDA. You must perform PCA via *Singular Value Decomposition*, or SVD for short (refer Section 2). For computing the eigenvalues and corresponding eigenvectors you must use *Jacobi Eigenvalue Algorithm* (refer Section 1).

1 Jacobi Eigenvalue Algorithm

The Jacobi eigenvalue algorithm is an iterative method for the calculation of the eigenvalues and eigenvectors of a real symmetric matrix (a process known as diagonalization).

Algorithm 1 is a description of the Jacobi method. The algorithm calculates a vector e which contains the eigenvalues and a matrix E which contains the corresponding eigenvectors, i.e. e_i is an eigenvalue and the column E_i an orthonormal eigenvector for e_i, for $i = 1, ..., n$.

Convergence criteria and error tolerance The computation converges when value of *state* becomes zero (condition on line 13 of function `JACOBI`) while allowing a tolerance 0.001 in computing equality and inequality of e_k and $prev\_e_k$ (lines 3 and 6 of function `UPDATE`).

2 Singular Value Decomposition (SVD)

The Singular Value Decomposition, or SVD for short, is a matrix decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler.

SVD of a matrix M is the factorization of M into the product of three matrices $M = U\Sigma V^T$ where the columns of U and V are orthonormal and the matrix Σ is diagonal with positive real entries. Given that M is a real m x n matrix that we wish to decompose, U is a m x m matrix, Σ is a m x n diagonal matrix, and V^T is the transpose of a n x n matrix.

SVD is the most well known and widely used matrix decomposition method. All matrices have an SVD, which makes it more stable than other methods, such as the eigendecomposition. As such, it is often used in a wide array of applications including compressing, denoising, and data reduction. Read the article `https://blog.statsbot.co/singular-value-decomposition-tutorial-52c695315254` for more information on SVD.

Computing SVD

1. Compute the Hermitian (conjugate transpose), M^T, of a matrix M.
2. Compute multiplication $M^T.M$.
3. Compute the eigenvalues and eigenvectors of $M^T.M$ using Jacobi algorithm.
4. Sort eigenvalues and corresponding columns of eigenvectors in descending order of eigenvalues.

Algorithm 1 Jacobi eigenvalue algorithm

```
1: function MAXIND($k$)
2:     $m := k + 1$
3:     for $i := k + 2$ to $N - 1$ do
4:         if $|S_{ki}| > |S_{km}|$ then
5:             $m := i$
6:         end if
7:     end for
8:     return $m$
9: end function
```

```
1: function UPDATE($k$, $t$))
2:     $prev\_e_k := e_k$, $e_k := prev\_e_k + t$
3:     if $changed_k$ and $prev\_e_k = e_k$ then
4:         $changed_k := false$
5:         $state := state - 1$
6:     else if not $changed_k$ and not $(prev\_e_k = e_k)$ then
7:         $changed_k := true$
8:         $state := state + 1$
9:     end if
10: end function
```

```
1: procedure JACOBI($S$)
2:     out: e, E
3:     global: $state, changed, ind, N$
4:     function ROTATE(k, l, i, j, c, s)
5:         $\begin{bmatrix} S_{kl} \\ S_{ij} \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} S_{kl} \\ S_{ij} \end{bmatrix}$
6:     end function
7:
8:     function JACOBI($S, N$)
9:         $E :=$ I, $state := N$
10:        for $k := 0$ to $N - 1$ do
11:            $ind_k :=$ MAXIND($k$), $e_k := S_{kk}$, $changed_k := true$
12:        end for
13:        while not $(state = 0)$ do
14:            $m := 0$
15:            for $k := 1$ to $N - 1$ do
16:                if $|S_{k\,ind_k}| > |S_{m\,ind_m}|$ then $m := k$ end if
17:            end for
18:            $k := m,\ l := ind_m,\ p := S_{kl},\ y := (e_l - e_k)/2$
19:            $d := |y| + \sqrt{p^2 + y^2},\ r := \sqrt{p^2 + d^2}$
20:            $c := d/r,\ s := p/r,\ t := p^2/d$
21:            if $y < 0$ then $s := -s,\ t := -t$ end if
22:            $S_{kl} := 0.0$, UPDATE($k, -t$), UPDATE($l, t$)
23:            for $i := 0$ to $k - 1$ do ROTATE($i, k, i, l, c, s,$) end for
24:            for $i := k + 1$ to $l - 1$ do ROTATE($k, i, i, l, c, s,$) end for
25:            for $i := l + 1$ to $N - 1$ do ROTATE($k, i, l, i, c, s,$) end for
26:            for $i := 0$ to $N - 1$ do
27:                $\begin{bmatrix} E_{ik} \\ E_{il} \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} E_{ik} \\ E_{il} \end{bmatrix}$
28:            end for
29:            $ind_k :=$ MAXIND($k$), $ind_l :=$ MAXIND($l$)
30:        end while
31:    end function
32: end procedure
```

5. Square root the eigenvalues to obtain the singular values of M.
6. Construct diagonal matrix Σ by placing singular values in descending order along its diagonal. Note that, the size of Σ if m x n, thus (i) place the top r singular values on the diagonal and discard the rest ($r = \texttt{min}(m, n)$) and (ii) pad with m-r rows and n-r columns containing `zeros`. Also compute Σ^{-1}
7. The sorted columns of eigenvectors represent V. Compute its transpose, V^T.
8. Compute the matrix U as $U = MV\Sigma^{-1}$.

SVD computation example

$$\text{let } M = \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix}$$

$$\text{since, } M^T = \begin{bmatrix} 4 & 3 \\ 0 & -5 \end{bmatrix} \text{ thus, } M^T.M = \begin{bmatrix} 25 & -15 \\ -15 & 25 \end{bmatrix}$$

$$\text{eigenvalues of } M^T.M \text{ using Jacobi algorithm} = \begin{bmatrix} 40 & 10 \end{bmatrix}$$

$$\text{corresponding eigenvectors: } V = \begin{bmatrix} -0.7071 & 0.7071 \\ 0.7071 & 0.7071 \end{bmatrix}$$

$$\text{Singular values: } \sigma_1 = \sqrt{40} = 6.3245,\ \sigma_2 = \sqrt{10} = 3.1622$$

$$\text{Thus, } \Sigma = \begin{bmatrix} 6.3245 & 0 \\ 0 & 3.1622 \end{bmatrix}, \text{ and, } \Sigma^{-1} = \begin{bmatrix} \frac{1}{6.3245} & 0 \\ 0 & \frac{1}{3.1622} \end{bmatrix} = \begin{bmatrix} 0.1581 & 0 \\ 0 & 0.3162 \end{bmatrix}$$

$$V^T = \begin{bmatrix} -0.7071 & 0.7071 \\ 0.7071 & 0.7071 \end{bmatrix}$$

$$\text{Thus, } U = MV\Sigma^{-1} = \begin{bmatrix} 0.4472 & 0.8944 \\ 0.8944 & -0.4472 \end{bmatrix}$$

Correctness Argument
Multiplying the matrices U, Σ and V^T gives back the original matrix M, i.e. $M = U\Sigma V^T$

For the example above,

$$U\Sigma V^T = \begin{bmatrix} 0.4472 & 0.8944 \\ 0.8944 & -0.4472 \end{bmatrix} \begin{bmatrix} 6.3245 & 0 \\ 0 & 3.1622 \end{bmatrix} \begin{bmatrix} -0.7071 & 0.7071 \\ 0.7071 & 0.7071 \end{bmatrix} = \begin{bmatrix} 3.9998 & 0 \\ 2.9999 & -4.9997 \end{bmatrix} \approx M$$

3 Principal Component Analysis (PCA)

Principal component analysis, or PCA for short, is a mathematical procedure that transforms a number of (possibly) correlated variables into a smaller number of uncorrelated variables called principal components. The objective of principal component analysis is to reduce the dimensionality (number of features) of the dataset, but retain as much of the original variability in the data as possible. The first principal component accounts for the majority of the variability in the data, the second principal component accounts for the majority of the remaining variability, and so on.

PCA can be thought of as a projection method where data with n-features is projected onto a subspace with n (or fewer)-featuress, while retaining most of the information.

The article `https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c` has a nice intuitive explanation of PCA. The article `http://setosa.io/ev/principal-component-analysis/` contains some interesting examples.

Computing PCA

1. Input a dataset D with s samples of f features, *ie* dimension of D is s x f.
2. Obtain eigenvalues by Jacobi algorithm and U by Singular Vector Decomposition on D (or alternatively D^T).
3. Choose the k columns of U corresponding to the k largest eigenvalues. Here, k is the number of dimensions of the new feature subspace ($k \leq f$)
4. Construct the projection matrix W of principal components from the selected k eigenvectors.
5. Transform the original dataset D via W to obtain a k-dimensional feature subspace $\widehat{D}$.

Note that, values in the input dataset D would be standardized.

PCA computation example Consider the "Iris" dataset deposited on the UCI machine learning repository (`https://archive.ics.uci.edu/ml/datasets/Iris`). The dataset contains 150 instances of Iris plants with 4 features.

Step 1 The dimension on input dataset D = 150 x 4 (s = 150, f = 4)

Step 2 Computing SVD of D^T (after standardization),

$$\Sigma = \begin{bmatrix} 20.8955 & 0 & 0 & 0 & ... \\ 0 & 11.7551 & 0 & 0 & ... \\ 0 & 0 & 4.7013 & 0 & ... \\ 0 & 0 & 0 & 1.7582 & ... \end{bmatrix} \text{ and, } U = \begin{bmatrix} -0.5223 & -0.3723 & 0.7210 & 0.2619 \\ 0.2633 & -0.9255 & -0.2420 & -0.1241 \\ -0.5812 & -0.0210 & -0.1408 & -0.8011 \\ -0.5656 & -0.0654 & -0.6338 & 0.5235 \end{bmatrix}$$

Step 3

Eigenvalues in descending order: $436.6227 > 138.1831 > 22.1029 > 3.0911$
Sum of eigenvalues = 599.9998
Explained variance of eigenvalues: $436.6227/599.9998 = 0.7277$, $138.1831/599.9998 = 0.2303$, $22.1029/599.9998 = 0.0368$ and $3.0911/599.9998 = 0.0051$.

It is clearly seen that most of the variance (72.77% of the variance to be precise) can be explained by the first principal component alone. The second principal component still bears some information (23.03%) while the third and fourth principal components can safely be dropped without losing too much information. Together, the first two principal components contain 95.8% of the information.

Hence, the reduced dimension, $k = 2$.

Step 4 Although, the name "projection matrix" has a nice ring to it, it is basically just a matrix of the concatenated top k eigenvectors (columns of U). The top k eigenvectors represent the principal components.

$$W = \begin{bmatrix} -0.5223 & -0.3723 \\ 0.2633 & -0.9255 \\ -0.5812 & -0.0210 \\ -0.5656 & -0.0654 \end{bmatrix}$$

Step 5 Use the 4 x 2-dimensional projection matrix W to transform the samples onto the new subspace via the equation, $\widehat{D} = D$ x W. The resulting $\widehat{D}$ is a 150 x 2 matrix of transformed samples.

4 Lab submission details

4.1 Problem Statement

- Your task is to implement a parallel PCA algorithm over rectangular datasets using CUDA.
- You will be provided a dataset, D (such that #samples $\geq$ #features, *ie* $s \geq f$), and minimum percentage data retention, R. You will be required to perform parallel PCA through parallel SVD, as follows:

    1. Perform SVD of D (or alternatively D^T). Use Jacobi algorithm to compute eigenvalues and eigenvectors. Return U, Σ and V^T.
    2. Recognize the minimum value of dimension k such that the sum of top k eigenvalues is $\geq R$. Return k.
    3. Transform D to $\widehat{D}$. Return $\widehat{D}$.

4.2 Correctness check

- Multiplication of $U.\Sigma.V^T$ must give back $M = D$ (or alternatively D^T).
- Your returned k must match our correct k.
- Provided that you have computed k correctly, your matrix $\widehat{D}$ must match our corresponding matrix.

Note: We will keep it in consideration that real values may not be exact matches. We will accept values within a $\leq$0.001 error tolerance.

4.3 Input format

The input file format and the main file to calculate the time taken by your implementation can be cloned from `https://github.com/dvynjli/col380_lab3_suite`. Read the `README` file for details.

4.4 Using GPU HPC

We will test your submissions on gpu.hpc using the CUDA module `compiler/cuda/9.2/compilervars`. (Online documentation of CUDA 9.2 is available at `https://docs.nvidia.com/cuda/archive/9.2/`).

Follow these steps for using gpu.hpc

1. Visit `http://supercomputing.iitd.ac.in/` and apply for an account.
    - Goto 'Apply for account/Renew Account'.
    - Use your kerberos credentials to login.
    - Enter 'svs' for 'Faculty supervisor (uid)' and '01 May 2019' for 'Requested Expiry date of access'
2. Access gpu.hpc as `ssh <kerberos id>@gpu.hpc.iitd.ac.in`. Use your kerberos password to login.
3. Copy your files to hpc client node
    ```
    scp <path-to-file/file> <kerberos is>@gpu.hpc.iitd.ac.in
    ```

4. Write a PBS script to submit a job to hpc queue. Follow: `http://supercomputing.iitd.ac.in/?pbs`.
   Make sure you load the CUDA module in your PBS script as
   `module load compiler/cuda/9.2/compilervars`.
5. Submit a job as `qsub <pbs-script> -q low`. `qsub` will return a job-id.
6. Check status of your submitted job as `qstat <job-id>`

4.5 Plagiarism policy

Make sure the code is not plagiarized. We have a repository of online codes. If you are found copying code from online sources or your peers, you will be penalized with a grade-drop + zero in the lab.

4.6 Late submission policy

10% penalty for each day over the deadline. Maximum of 2 days allowed after which summary zero will be awarded (unless there is a medical emergency – for which you will have to provide a letter from the doctor).

4.7 Word of advice on time management

We believe that this will be an involving task. You are strongly advised to start working on the assignment at the earliest possible.

4.8 Evaluation scheme

1. Correct computation of U, Σ, V^T	**45 Marks**
2. Correct computation of k	**eligibility for component 3**
3. Correct computation of $\widehat{D}$	**25 Marks & eligibility for component 4**
4. Performance	**30 Marks**

Note: The grading for component 4 (*Performance*) will be relative.
The grading of components 1, 2 and 3 will be absolute.