ST 790 - HW2

Brian Naughton

February 11, 2015

1. Prove the majorization

$$\left(x_{ij} - \sum_k v_{ik}w_{kj}\right)^2 \leq \sum_k \frac{a_{ikj}^{(t)}}{b_{ij}^{(t)}} \left(x_{ij} - \frac{b_{ij}^{(t)}}{a_{ikj}^{(t)}} v_{ik}^{(t)} w_{kj}^{(t)}\right)^2$$

Proof:

$$\begin{aligned}
\left(x_{ij} - \sum_k v_{ik}w_{kj}\right)^2 &= x_{ij}^2 - 2x_{ij}\sum_k v_{ik}w_{kj} + \left(\sum_k v_{ik}w_{kj}\right)^2 \\
&\leq x_{ij}^2 - 2x_{ij}\sum_k v_{ik}w_{kj} + \sum_k (v_{ik}w_{kj})^2 \\
&= \frac{\sum_k a_{ikj}^{(t)}}{b_{ij}^{(t)}} x_{ij}^2 - 2x_{ij}\sum_k \frac{a_{ikj}^{(t)}}{b_{ij}^{(t)}} \frac{b_{ij}^{(t)}}{a_{ikj}^{(t)}} v_{ik}w_{kj} + \sum_k (v_{ik}w_{kj})^2 \\
&= \sum_k \frac{a_{ikj}^{(t)}}{b_{ij}^{(t)}} \left(x_{ij}^2 - 2x_{ij}\frac{b_{ij}^{(t)}}{a_{ikj}^{(t)}} v_{ik}w_{kj} + \left(\frac{b_{ij}^{(t)}}{a_{ikj}^{(t)}}\right)^2 (v_{ik}w_{kj})^2\right) \\
&= \sum_k \frac{a_{ikj}^{(t)}}{b_{ij}^{(t)}} \left(x_{ij} - \frac{b_{ij}^{(t)}}{a_{ikj}^{(t)}} v_{ik}w_{kj}\right)^2.
\end{aligned}$$

The inequality above is due to the fact that the sum of squares is greater than the square of the sum (note: this could be more rigorously proven using the triangle inequality and induction).

Derive the multiplicitive update for v_{ik}:

$$\begin{aligned}
&\frac{\partial}{\partial v_{ik}} \sum_i \sum_j \sum_k \frac{a_{ikj}^{(t)}}{b_{ij}^{(t)}} \left(x_{ij} - \frac{b_{ij}^{(t)}}{a_{ikj}^{(t)}} v_{ik} w_{kj}^{(t)}\right)^2 \overset{set}{=} 0 \\
&\Rightarrow 2\sum_j \frac{a_{ikj}^{(t)}}{b_{ij}^{(t)}} \left(x_{ij} - \frac{b_{ij}^{(t)}}{a_{ikj}^{(t)}} v_{ik} w_{kj}^{(t)}\right) \left(-\frac{b_{ij}^{(t)}}{a_{ikj}^{(t)}} w_{kj}^{(t)}\right) = 0 \\
&\Rightarrow \sum_j x_{ij} w_{kj}^{(t)} = v_{ik} \sum_j \frac{(w_{kj}^{(t)})^2 b_{ij}^{(t)}}{a_{ikj}^{(t)}} = v_{ik} \sum_j \frac{w_{kj}^{(t)} b_{ij}^{(t)}}{v_{ik}^{(t)}} \\
&\Rightarrow v_{ik}^{(t+1)} = v_{ik}^{(t)} \frac{\sum_j x_{ij} w_{kj}^{(t)}}{\sum_j b_{ij}^{(t)} w_{kj}^{(t)}}
\end{aligned}$$

Derive the multiplicitive update for w_{kj}:

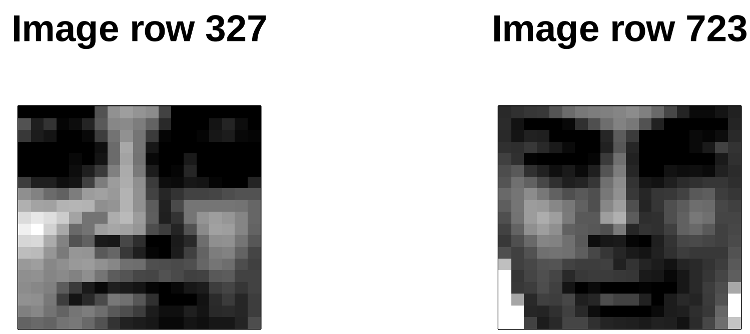
$$
\begin{aligned}
&\frac{\partial}{\partial w_{kj}} \sum_i \sum_j \sum_k \frac{a_{ikj}^{(t)}}{b_{ij}^{(t)}} \left(x_{ij} - \frac{b_{ij}^{(t)}}{a_{ikj}^{(t)}} v_{ik} w_{kj}^{(t)} \right)^2 \stackrel{set}{=} 0 \\
&\Rightarrow 2 \sum_i \frac{a_{ikj}^{(t)}}{b_{ij}^{(t)}} \left(x_{ij} - \frac{b_{ij}^{(t)}}{a_{ikj}^{(t)}} v_{ik}^{(t)} w_{kj} \right) \left(- \frac{b_{ij}^{(t)}}{a_{ikj}^{(t)}} v_{ik}^{(t)} \right) = 0 \\
&\Rightarrow \sum_i x_{ij} v_{ik}^{(t)} = w_{kj} \sum_i \frac{(v_{ik}^{(t)})^2 b_{ij}^{(t)}}{a_{ikj}^{(t)}} = w_{kj} \sum_i \frac{v_{ik}^{(t)} b_{ij}^{(t)}}{w_{kj}^{(t)}} \\
&\Rightarrow w_{kj}^{(t+1)} = w_{kj}^{(t)} \frac{\sum_i x_{ij} v_{ik}^{(t)}}{\sum_i b_{ij}^{(t)} v_{ik}^{(t)}}
\end{aligned}
$$

2. Implement algorithm in R:

```r
setwd('/home/bpnaught/st790-2015spr/hw2/')
# nnmf() implements the nonnegative matrix factorization algorithm and returns
#   the solution matrices V, W and the objective value that was minimized.
# X = (m x n) matrix
# r = rank of lower rank matrices V & W
# V = (m x r) matrix of starting values (optional)
# W = (r x n) matrix of starting values (optional)
# tol = convergence threshold of objective value (optional)
nnmf <- function(X, r, V=NULL, W=NULL, tol=10e-4) {
  m <- nrow(X)
  n <- ncol(X)
  # Use starting values of all 1's if not provided
  if (is.null(V)) V <- matrix(1, m, r)
  if (is.null(W)) W <- matrix(1, r, n)
  B = V %*% W
  notConverged = TRUE
  oldLoss <- 1e5
  iter <- 0
  while (notConverged) {
    iter   <- iter +1
    V <- V * tcrossprod(X,W) / tcrossprod(B,W)
    B <- V %*% W
    W <- W * crossprod(V, X) / crossprod(V, B)
    B <- V %*% W
    newLoss <- sum((X - B)^2) # Frobenius norm
    objectiveValue <- abs(newLoss - oldLoss) / (oldLoss + 1)
    if (objectiveValue <= tol) notConverged  <- FALSE
    oldLoss <- newLoss
  }
```

```r
  return (list(V=V, W=W, objValue=objectiveValue))
}
```

3. Read in `nnmf-2429-by-361-face.txt` and display a couple sample images



4. Report running times for $r = 10, 20, 30, 40, 50$ in seconds:

```
##      10     20     30     40     50
##   6.091 13.792 24.955 38.204 47.605
```

5. Choose $r = 30$, and initialize $\boldsymbol{V}^{(0)}$ and $\boldsymbol{W}^{(0)}$ with random samples from a Uniform(0,1) distribution.

```r
m <- nrow(X)
n <- ncol(X)
r <- 10
startGiven  <- nnmf(X, r, V[, 1:r], W[1:r, ]) # Using given starting values
set.seed(327)
Vrand <- matrix(runif(m*r), m, r)
Wrand <- matrix(runif(r*n), r, n)
startRandom <- nnmf(X, r, Vrand, Wrand) # Using random starting values

# Check to see if the results are the same
all.equal(startGiven$objValue, startRandom$objValue)

## [1] "Mean relative difference: 0.003701373"

startGiven$objValue
```

```
## [1] 0.0009916487

startRandom$objValue

## [1] 0.0009879782

all.equal(startGiven$V, startRandom$V)

## [1] "Mean relative difference: 0.8520246"

all.equal(startGiven$W, startRandom$W)

## [1] "Mean relative difference: 0.9110567"
```

The random starting values do not give the same results ($\boldsymbol{V}$, $\boldsymbol{W}$ and objective function) as the given starting values. The objective values are different, but clearly close due to convergence. The first few rows and columns of the W and V matrices are extremely different:

```
signif(startGiven$V[1:4, 1:4], 3)

##       [,1]   [,2]   [,3]   [,4]
## [1,] 0.117 0.0525 0.0950 0.0910
## [2,] 0.124 0.0107 0.0737 0.1270
## [3,] 0.138 0.0437 0.0532 0.0989
## [4,] 0.169 0.0950 0.0745 0.0335

signif(startRandom$V[1:4, 1:4], 3)

##         [,1]   [,2]     [,3]   [,4]
## [1,] 0.02250 0.0277 0.018400 0.0774
## [2,] 0.00759 0.0558 0.011900 0.0507
## [3,] 0.02680 0.0808 0.000715 0.0493
## [4,] 0.06830 0.0862 0.008030 0.1070

signif(startGiven$W[1:4, 1:4], 3)

##            V1       V2       V3       V4
## [1,] 7.89e-09 2.51e-09 6.57e-10 1.16e-09
## [2,] 9.70e-03 2.07e-03 6.90e-03 1.67e-01
## [3,] 1.11e-04 2.08e-05 4.87e-04 1.28e-02
## [4,] 2.35e-01 3.48e-02 7.75e-05 9.15e-04

signif(startRandom$W[1:4, 1:4], 3)
```

```
##            V1       V2       V3       V4
## [1,] 3.84e-06 0.002340 0.088700 2.13e-01
## [2,] 3.77e-04 0.003580 0.001060 7.06e-05
## [3,] 4.60e-04 0.000687 0.000611 1.46e-02
## [4,] 5.02e-01 0.417000 0.528000 6.45e-01
```

How does it compare to use $v_{ik}^{(0)} = w_{kj}^{(0)} = 1$ for all i, j, k?

```
startOnes <- nnmf(X, r) # nnmf default is to use ones
all.equal(startGiven$objValue, startOnes$objValue)

## [1] "Mean relative difference: 0.986395"

startGiven$objValue

## [1] 0.0009916487

startOnes$objValue

## [1] 1.349136e-05

all.equal(startGiven$V, startOnes$V)

## [1] "Mean relative difference: 0.7863629"

all.equal(startGiven$W, startOnes$W)

## [1] "Mean relative difference: 1.327656"

signif(startGiven$V[1:4, 1:4], 3)

##       [,1]   [,2]   [,3]   [,4]
## [1,] 0.117 0.0525 0.0950 0.0910
## [2,] 0.124 0.0107 0.0737 0.1270
## [3,] 0.138 0.0437 0.0532 0.0989
## [4,] 0.169 0.0950 0.0745 0.0335

signif(startOnes$V[1:4, 1:4], 3)

##        [,1]   [,2]   [,3]   [,4]
## [1,] 0.0274 0.0274 0.0274 0.0274
## [2,] 0.0272 0.0272 0.0272 0.0272
## [3,] 0.0276 0.0276 0.0276 0.0276
## [4,] 0.0264 0.0264 0.0264 0.0264
```

```r
signif(startGiven$W[1:4, 1:4], 3)

##             V1       V2       V3       V4
## [1,] 7.89e-09 2.51e-09 6.57e-10 1.16e-09
## [2,] 9.70e-03 2.07e-03 6.90e-03 1.67e-01
## [3,] 1.11e-04 2.08e-05 4.87e-04 1.28e-02
## [4,] 2.35e-01 3.48e-02 7.75e-05 9.15e-04

signif(startOnes$W[1:4, 1:4], 3)

##         V1    V2    V3    V4
## [1,] 0.37 0.461 0.511 0.474
## [2,] 0.37 0.461 0.511 0.474
## [3,] 0.37 0.461 0.511 0.474
## [4,] 0.37 0.461 0.511 0.474
```

Just as before the results from Q4 are very different than using all 1's as starting values. However, the interesting thing here is that each element of the columns of W are the same, and each element of the rows of V are the same.

6. Investigate the GPU capabilities, and report the speed gain.

   I implemented the same `nnmf()` function in `R`, but used CUBLAS functions to carry out matrix multiplications. The package `gputools` was used to provide wrapper functions for the CUBLAS routines. In order to use `gputools` on the teaching server, some changes need to be made to the package for it to work. Use the following instructions to install `gputools` on the server:

   (a) Download `gputools` package – gputools_0.28.tar.gz
   (b) Unpack it: `tar -zxvf gputools_0.28.tar.gz`
   (c) Open the file: ./gputools/src/Makefile
   (d) Remove the line "-gencode arch=compute_10,code=sm_10" from the file and save it.
   (e) Pack the library back together: `tar -pczf gputools2.tar.gz ./gputools`
   (f) Install it form the command line: `R CMD build gputools && R CMD INSTALL gputools2.tar.gz`

   However, Rstudio cannot find a shared CUDA file, so `gputools` had to be used in batch mode. The `R` script "nnmfGPU.R" runs the GPU version of `nnmf()`, and saves the results in an Rdata workspace file. Use the command `R CMD BATCH nnmfGPU.R` to run it.

   Load the timing results:

```r
load('nnmfGPU.Rdata')
timingsGPU

##     10     20     30     40     50
##  5.814  6.281  8.400 11.137 11.184
```

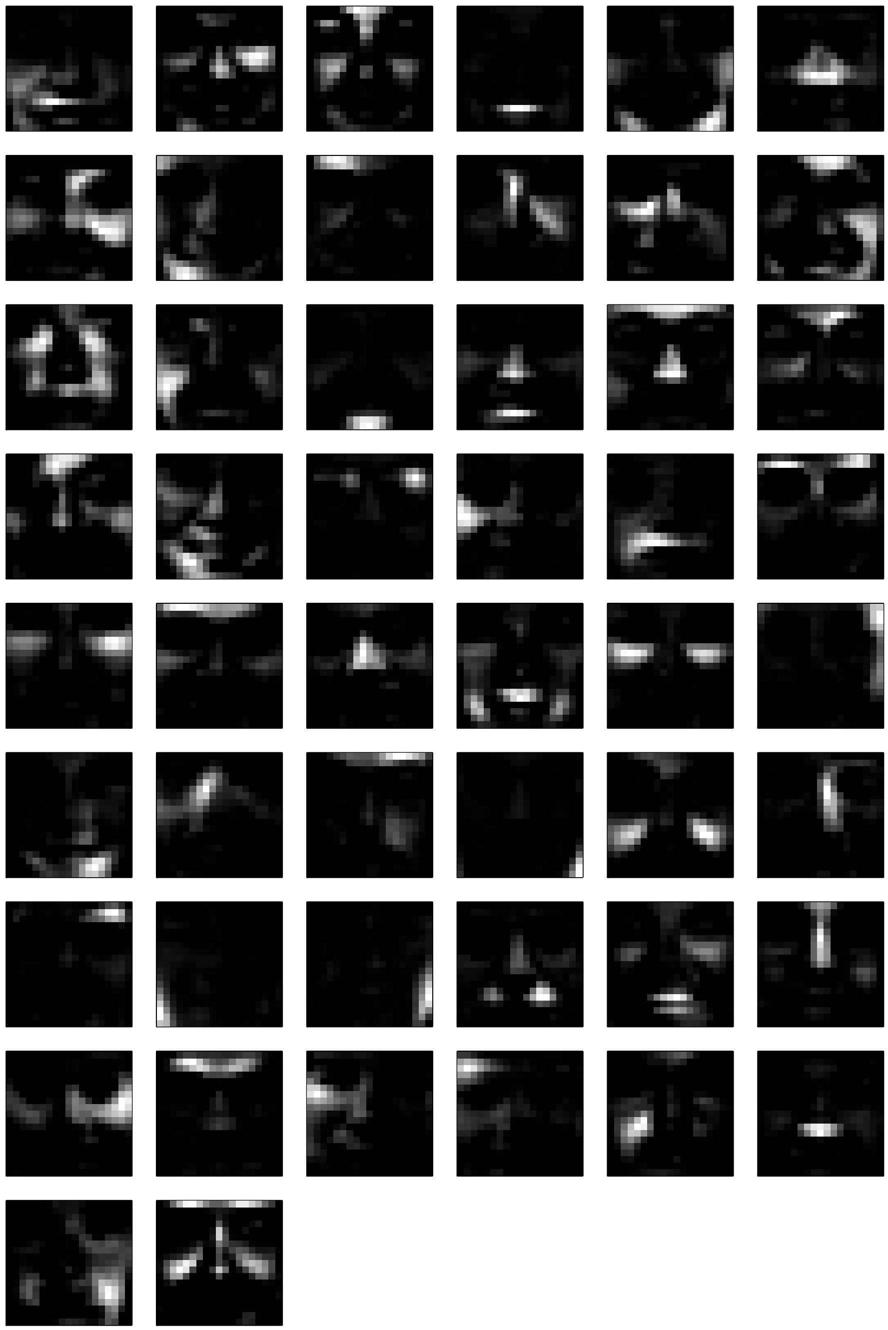
How do they compare to the timings on the CPU?

```
timings / timingsGPU

##       10       20       30       40       50
## 1.047644 2.195829 2.970833 3.430367 4.256527
```

This shows there is as much as an 5-fold speed up by using the GPU for matrix multiplications! It's also interesting that the timing does not increase that much as r increases for the CPU version

7. Plot the basis images (rows of $\boldsymbol{W}$) at rank $r = 50$.

```
r <- 50
rank50 <- nnmf(X, r, V[, 1:r], W[1:r, ])
par(mfrow=c(9,6))
par(mar=c(1,1,1,1)/2)
for (k in 1:50) {
    tmp <- matrix(rank50$W[k, 361:1], 19, 19, byrow = TRUE)
    image(z=tmp, col=grayColors, xaxt='n', yaxt='n')
}
sessionInfo()

## R version 3.1.2 (2014-10-31)
## Platform: x86_64-unknown-linux-gnu (64-bit)
##
## locale:
## [1] C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.9
##
## loaded via a namespace (and not attached):
## [1] evaluate_0.5.5 formatR_1.0    highr_0.4      stringr_0.6.2
## [5] tools_3.1.2
```



The images are all different features of the faces – eyes, mouths, cheeks, teeth, etc. This intuitively makes sense because then the rows of V are linear cominations of these features to approximate the original image.