Homework #11, EECS 598-006, W20. Due **Thu. Apr. 10**, by 4:00PM

---

1. [6] **Transform learning from heterogeneous data**

Suppose we want to learn a single **sparsifying transform** from two collections of data vectors: $\boldsymbol{X}_1 \in \mathbb{F}^{N \times L_1}$ and $\boldsymbol{X}_2 \in \mathbb{F}^{N \times L_2}$ where we expect the transforms of the training data in $\boldsymbol{X}_2$ to be less sparse than those of $\boldsymbol{X}_1$. A natural cost function in this situation is:

$$\mathop{\arg\min}_{\boldsymbol{T} \in \mathbb{F}^{N \times N} : \boldsymbol{T}'\boldsymbol{T} = \boldsymbol{I}_N} \min_{\boldsymbol{Z}_1, \boldsymbol{Z}_2} \Psi(\boldsymbol{T}, \boldsymbol{Z}_1, \boldsymbol{Z}_2), \quad \Psi(\boldsymbol{T}, \boldsymbol{Z}_1, \boldsymbol{Z}_2) \triangleq \frac{1}{2} |||\boldsymbol{T}\boldsymbol{X}_1 - \boldsymbol{Z}_1|||_{\mathrm{F}}^2 + \frac{1}{2} |||\boldsymbol{T}\boldsymbol{X}_2 - \boldsymbol{Z}_2|||_{\mathrm{F}}^2 + \beta_1 \|\boldsymbol{Z}_1\|_0 + \beta_2 \|\boldsymbol{Z}_2\|_0 ,$$

where $0 < \beta_2 < \beta_1$.

An 3-block alternating minimization approach is natural for this **transform learning** optimization problem.

(a) [3] Derive the update for $\boldsymbol{T}$. Hint: you may assume $N$ is small enough to allow for SVD operations.

(b) [3] Derive the update for $\boldsymbol{Z}_1$.

(c) [0] Is your approach a **BCM** or **BCD** algorithm?

---

2. [12] **Compressed sensing with analysis regularizer with PGM and BCD**

Ch. 6 discussed multiple approaches to solving this **analysis regularizer** optimization problem with $\boldsymbol{A} \in \mathbb{F}^{M \times N}$ and $\boldsymbol{T} \in \mathbb{F}^{K \times N}$:

$$\hat{\boldsymbol{x}} = \mathop{\arg\min}_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta R(\boldsymbol{x}), \quad R(\boldsymbol{x}) = \min_{\boldsymbol{z}} \frac{1}{2} \|\boldsymbol{T}\boldsymbol{x} - \boldsymbol{z}\|_2^2 + \alpha \|\boldsymbol{z}\|_1 .$$

(a) [3] One approach is to write the cost function as $\Psi(\tilde{\boldsymbol{x}})$, where $\tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z} \end{bmatrix}$ and then apply the **proximal gradient method** (**PGM**) to that $\Psi$. Determine the PGM update. Pay special attention to the **step size** and the soft **threshold**. (Use the best possible Lipschitz constant.)

(b) [6] Another approach is to apply **BCD** to the following two-block cost function:

$$\Psi(\boldsymbol{x}, \boldsymbol{z}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \left( \frac{1}{2} \|\boldsymbol{T}\boldsymbol{x} - \boldsymbol{z}\|_2^2 + \alpha \|\boldsymbol{z}\|_1 \right).$$

Suppose we apply one iteration of **GD** for the $\boldsymbol{x}$ update and exact minimization for the $\boldsymbol{z}$ update.

- Determine the $\boldsymbol{x}$ update, paying special attention to the **step size**. (Use the best possible Lipschitz constant.)
- Determine the $\boldsymbol{z}$ update, paying special attention to the soft **threshold**.

(c) [3] Suppose you know that $\boldsymbol{T}\boldsymbol{x}_{\text{true}}$ is a vector having a few significant values near or above some constant $c$, and the other values are zero or near zero. Discuss how you would set $\alpha$ for the two approaches considered above. Keep in mind that the presence of noise will cause the values of $\boldsymbol{T}\boldsymbol{x}_k$ to be spread out some.

(d) [0] Compare how easy or intuitive it is to set $\alpha$ for the two cases.

3. [30] **Low-rank matrix factorization: large scale**

Given a $M \times N$ data matrix $\boldsymbol{Y}$, this problem considers the **low-rank matrix approximation** problem:

$$\hat{\boldsymbol{X}} = \hat{\boldsymbol{U}}\hat{\boldsymbol{V}}, \quad (\hat{\boldsymbol{U}}, \hat{\boldsymbol{V}}) = \underset{\boldsymbol{U} \in \mathcal{V}_K(\mathbb{R}^M),\, \boldsymbol{V} \in \mathbb{F}^{K \times N}}{\arg\min} \Psi(\boldsymbol{U}, \boldsymbol{V}), \quad \Psi(\boldsymbol{U}, \boldsymbol{V}) \triangleq \frac{1}{2} \|\boldsymbol{Y} - \boldsymbol{U}\boldsymbol{V}\|_{\mathrm{F}}^2,$$

where $\mathcal{V}_K(\mathbb{F}^M)$ denotes the **Stiefel manifold** of $M \times K$ matrices having orthonormal columns.

(a) [10] Write a JULIA function that runs `niter` iterations of a two-block **BCM** algorithm for computing $\hat{\boldsymbol{U}}$ and $\hat{\boldsymbol{V}}$, as discussed in class. Your function must evaluated a user-defined function `fun` at the initial guess $(\boldsymbol{U}_0, \boldsymbol{V}_0)$ and after *each update*, so it will be called `2niter+1` times for `niter` iterations. Update $\boldsymbol{U}$ first, then update $\boldsymbol{V}$.
Your code must *not* use the **SVD** of $\boldsymbol{Y}$ or any other $M \times N$ matrix, but it may use the SVD of any matrix having one or more dimensions that are $K$ because $K$ is small.
Your file should be named `lrmf_uv.jl` and should contain the following function:

```
"""
    U,V,out = lrmf_uv(Y, U0, V0 ; niter=5)

Low-rank matrix factorization by solving `min_{U,V} |X - UV|_F^2`
where `rank(UV) = K << min(M,N)`
May use SVD only for small matrices (involving `K`)

in
- `Y M×N` data matrix
- `U0 M×K` initial guess for left factor
- `V0 K×N` initial guess for right factor

option
- `niter::Int` # of iterations; default 5
- `fun::Function` user function `fun(iter,U,V)`
 is evaluated after every update of `U` or `V`
default `(iter,U,V) -> undef`

out
- `U M×K` final left factor
- `V K×N` final right factor
- `out::Array{Any} [fun(0,U,V) ... fun(2*niter,U,V)]`
"""
function lrmf_uv(Y, U0, V0 ;
        niter::Int=5, fun::Function = (iter, U, V) -> undef)
```

Submit your solution to mailto:eecs556@autograder.eecs.umich.edu.

(b) [5] Write a JULIA script that applies your **BCM** algorithm to the data $\boldsymbol{Y}$ using the initial random estimates $\boldsymbol{U}_0$,$\boldsymbol{V}_0$ shown in the following code:

```
using Random: seed!
using LinearAlgebra: qr
fun = (x,p) -> p == 1 ? x == 1 : [x >= p; fun(x - p*(x >= p), p/2)]
tmp = [0 14 1 1 1 14 0 0 0 9 15 8 8 4 8 8 15 9 0]
tmp = hcat([Int.(fun(v, 2^3)) for v in tmp]...)
tmp = [zeros(Int, 1,19); tmp; zeros(Int, 1,19)]'
Xtrue = kron(10 .+ 80*tmp, ones(100,100)); @show (M,N) = size(Xtrue)
seed!(0); sig = 20; Y = Xtrue + sig * randn(size(Xtrue))
K = 7; U0 = qr(randn(M,K)).Q[:,1:K]; V0 = randn(K,N); # initial guesses of U,V
```

Note that the code uses $K = 7$ even though the true rank is lower than that, because in practice the rank is often unknown.

After *each update*, compute the cost function $\Psi$ above, and also compute the NRMSE $\|\boldsymbol{U}\boldsymbol{V} - \boldsymbol{X}_{\text{true}}\|_{\text{F}} / \|\boldsymbol{X}_{\text{true}}\|_{\text{F}}$. Also compute those two quantities for the initial guesses $\boldsymbol{U}_0$ and $\boldsymbol{V}_0$.
Your script should generate all the figures in the next parts.
Submit a screenshot of your test code to gradescope.

(c) [5] Show an image of $\hat{\boldsymbol{X}} = \hat{\boldsymbol{U}}\hat{\boldsymbol{V}}$. (It should look quite familiar, and fairly reasonable quality.)
(For yourself, also look at $\boldsymbol{Y}$ to see how much the noise was reduced.)

(d) [5] Make a plot of the cost $\Psi$ versus "half iteration" *i.e.*, versus `(0:2niter)/2` because a full iteration is an update of both $\boldsymbol{U}$ and $\boldsymbol{V}$ but we are evaluating the cost and NRMSE after every update (to make sure it is all working correctly).
Hint: the cost function converges surprisingly quickly.

(e) [5] Make a plot of NRMSE versus "half iteration" too.
Hint: The final NRMSE should be about 0.06, which is much less than the NRMSE of 0.46 of the noisy data image $\boldsymbol{Y}$.

(f) [0] Optional. Compare to the conventional SVD-based low-rank matrix approximation approach from EECS 551.