Homework #12, EECS 598-006, W20. Due **Sun. Apr. 19**, by 4:00PM

---

1. [15] **Exam2 problem: submit to Canvas by Sat. Apr. 18 at 1PM sharp**

Make a clearly stated problem that is suitable for an exam in this course, based on the material from Ch. 4 and beyond (HW 7-12), and provide your own solution to the problem.
If you make a good problem (not too trivial, not too hard) then it might be used on an actual exam. Problems requiring JULIA solutions could be especially interesting. To earn full credit:

- Problems and solutions that do not involve JULIA must fit on a single page with a horizontal line that clearly separates the question (above) from the answer (below). For problems involving JULIA the 1-page limit does not apply because it can take more space to give thorough problem specifications and solutions involving plots can take quite a few lines of code.
- Your solution to the problem must be correct.
- Your problem should not be excessively trivial (nor beyond the scope of the course).
  It must use ideas from this course that go beyond what was covered in EECS 551.
- Your problem must differ from homework, clicker, and old exam problems from 551 and this course.
- Upload your problem/solution to gradescope as usual for grading.
  Also submit your problem/solution in pdf format to Canvas by the **bold deadline above**, for distribution to the class.
  We can export from Canvas easily but not from gradescope.
- Do not submit any other parts of your HW assignment to Canvas; only this part.
- Do not put your name on the problem/solution that you upload to Canvas because we plan to export all problems/solutions to distribute as practice problems.

---

2. [9] **Dictionary learning**

Given $N \times L$ data matrix $\boldsymbol{X}$, consider the following **dictionary learning** optimization problem:

$$\hat{\boldsymbol{D}} = \underset{\boldsymbol{D}\in\mathcal{D}}{\arg\min} \underset{\boldsymbol{Z}\in\mathbb{F}^{K\times L}}{\min} \Psi(\boldsymbol{D},\boldsymbol{Z}), \quad \Psi(\boldsymbol{D},\boldsymbol{Z}) \triangleq \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{D}\boldsymbol{Z}\|_{\mathrm{F}}^2 + \beta \left\|\mathsf{vec}(\boldsymbol{Z})\right\|_0,$$

where $\mathcal{D}$ is the set of $N \times K$ matrices having unit-norm columns. The course notes discuss two approaches to this optimization problem: a **BCD** approach where one alternates between updating all of $\boldsymbol{D}$ and then all of $\boldsymbol{Z}$, and a **BCM** approach where one updates one column (atom) of $\boldsymbol{D}$ and one row of $\boldsymbol{Z}$. The drawback of the atom-wise BCM approach is that it is not conducive to parallel computing.

(a) [3] Write down a **majorize-minimize** (**MM**) (or equivalent **proximal gradient method** (**PGM**)) update for $\boldsymbol{D}$ in a form that is explicit enough you could implement it in JULIA if asked. (This update is mentioned in the course notes without any formula given.)
Hint. One possible approach uses the **vec trick** of HW1#13 to determine a Lipschitz constant.
Hint. You also might need to complete a square.

(b) [3] An alternative is to make a **BCD** approach where one updates, say, two atoms of $\boldsymbol{D}$ together. Thinking of the partition $\boldsymbol{D} = [\boldsymbol{D}_{1:2}\ \boldsymbol{D}_{3:K}]$ write a concise expression for an update of $\boldsymbol{D}_{1:2}$ that is guaranteed to decrease Ψ.

(c) [0] Your answer to (b) should depend on the **spectral norm** of a certain matrix. Often we think of a spectral norm as being impractical for large-scale problems. Discuss whether it is practical in this case when L, the number of training samples, is large. As part of that discussion, describe the computation requirements for that spectral norm, using **big O notation**.

(d) [3] Discuss what advantage the approach of (b) has over (a).

---

3. [3] Please complete the online course evaluation for this class. Afterwards, submit a statement to gradescope saying that you did the evaluations. The Honor Code applies. Your evaluation is very important to me for improving the course next time I teach it. Your evaluation is entirely anonymous.

---

4. [51] **Learning a binary classifier for handwritten digits via SGM and ADMM**

This problem continues the **machine learning** problem of designing a **binary classifier** for two classes of handwritten digit images. Here we use the **hinge loss function** $h(t) = \max(1-t, 0)$ and the 1-norm regularizer in the following cost function:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \Psi(\boldsymbol{x}), \quad \Psi(\boldsymbol{x}) = \mathbf{1}'h.(\boldsymbol{A}\boldsymbol{x}) + \beta \left\|\boldsymbol{x}\right\|_1,$$

where $\boldsymbol{A}$ is the training data matrix where each row consists of the product of a feature vector and the corresponding label (± 1). The dot subscript is the JULIA-style notation. This 1-norm regularizer encourages the classifier weights $\hat{\boldsymbol{x}}$ to be sparse, aiming for a parsimonious model in hopes of better generalization performance. By using the hinge loss instead of the Huber hinge loss, we need not select a parameter δ.

This cost function is challenging because it is not differentiable.
Nevertheless, its **subgradient** is well defined, and the problem is also amenable to an **ADMM** approach.

(a) [10] Download the template JULIA code `class-hinge-l1-how.jl` from Canvas.
Add code to the notebook to implement the **subgradient method** (**SGM**). (We could have done **stochastic gradient descent** (**SGD**), but the data size here is small enough that I decided to make the problem simpler.)
Use $\beta = 0.3$ and run 8000 iterations using step sizes $\dfrac{4}{1+k}$ for $k = 1, \ldots,$.
I found that initializing with $\mathbf{0}$ led to very slow convergence. (You can try it yourself to see.)
Instead, for $\boldsymbol{x}_0$ use the final (300th) iterate of POGM for the Huber hinge loss (your solution to a previous HW problem). The variable is `x_pogm` in the code. We saw previously that it gave quite good classification accuracy. Using a good initialization, when available, is good practice, especially for slow methods like SGM.
Submit a screenshot of your added SGM code to gradescope.

(b) [5] Plot the cost function Ψ versus iteration k for yourself; you will see that it does not decrease monotonically. In fact it goes dramatically uphill initially!
For a more useful graphic, add `ylim=[40,60]` to your plot command and submit that version to gradescope.

(c) [5] Make an image of the (sparse!) classifier feature weights $\hat{\boldsymbol{x}}$ for the SGM-based classifier. Submit to gradescope.
Hint. The SGM $\hat{\boldsymbol{x}}$ should be fairly similar to your previous POGM solution. And the classification accuracy is also pretty similar (see below).

(d) [3] Derive the **proximal operator** for $c \cdot h$ where $c > 0$ and h is the hinge loss.
Sketch (or plot) the result for $c = 2$ and submit your equation and figure to gradescope.

(e) [10] Implement **ADMM** for this cost function using the variable split $\boldsymbol{w} = \boldsymbol{A}\boldsymbol{x}$, to make the scaled augmented Lagrangian

$$L(\boldsymbol{x}, \boldsymbol{w}; \boldsymbol{\eta}) = \mathbf{1}'h.(\boldsymbol{w}) + \beta \left\|\boldsymbol{x}\right\|_1 + \frac{\mu}{2}\left(\left\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{w} + \boldsymbol{\eta}\right\|_2^2 - \left\|\boldsymbol{\eta}\right\|_2^2\right).$$

For grading consistency, update $\boldsymbol{x}$ then $\boldsymbol{w}$ then $\boldsymbol{\eta}$. The $\boldsymbol{x}$ update is a sparse coding problem. Use one iteration of the **proximal gradient method** (**PGM**) with step size α for this (inexact) update. Do not use $\|\cdot\|_2$, so that your method can scale to large problems. Do not use `pogm_restart` here.
Your file should be named `hinge1_admm.jl` and should contain the following function:

```
"""
    x, out = hinge1_admm(A, α ; x0, niter, μ, fun)

ADMM for hinge+l1 cost function `Ψ(x) = 1'h.(A*x) + β ||x||_1`
for binary classifier design

in
- `A::AbstractMatrix{<:Number} M × N` where each row is a feature vector
- `α::Real` step size for PGM for `x` update
 (i.e., 1/L for a practical Lipschitz constant)

option
- `x0::AbstractVector{<:Number}` initial guess, default `zeros(N)`
- `niter::Int` # of iterations; default `1`
- `μ::Real` Lagrange penalty parameter; default `1`
```

```
- `β::Real` regularization parameter; default `0`
- `fun::Function = (iter,x) -> undef`

out
- `x` final guess, a vector in `R^N`
- `out [fun(0,x0), fun(1,x1), ..., fun(niter, x)]`
"""
function hinge1_admm(A::AbstractMatrix{<:Number}, α::Real ;
    x0::AbstractVector{<:Number} = zeros(size(A,2)),
    niter::Int=1,
    β::Real = 0,
    μ::Real = 1,
    fun::Function = (iter,x) -> undef,
)
```

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

(f) [5] Apply your ADMM method to the handwritten digit data. Experiment with $\mu > 0$ and find a value that seems to lead to fast convergence. I am having you find this value yourself instead of specifying it for you because having to choose μ is a primary practical drawback of AL/ADMM methods! Again initialize $\boldsymbol{x}_0$ with `x_pogm`, initialize $\boldsymbol{w}_0 = \boldsymbol{A}\boldsymbol{x}_0$, and $\boldsymbol{\eta}_0 = \boldsymbol{0}$. For my chosen μ value, the cost function had pretty much flattened ouot by 8000 iterations. My plot uses `ylim=[40,60]` to match the range used for SGM above.
Plot the cost function $\Psi(\boldsymbol{x}_k)$ versus iteration k and submit.

(g) [5] Show the classifier weights $\hat{\boldsymbol{x}}$ for ADMM and submit.

(h) [5] Use the test data to make a table that reports the classification accuracies of the nearest subspace classifier, the linear regression classifier, the Huber hinge + 1-norm classifier using POGM, and both the SGM and ADMM classifiers for both digits, as well as the overall classification accuracy. Hint. Most of these numbers are in the notebook already.
Hint: if your code is working, the overall classification test accuracy on the test data with SGM and ADMM is pretty close to that of POGM.

(i) [3] Discuss how you would solve this optimization problem by a version of ADMM where all of the updates are exact, meaning can be solved perfectly with no inner iterations.