LEAST SQUARES OPTIMIZATION IN MODEL UNMIXING

Mentors: Dr. Rick Archibald, Dr. Kwai Wong, Dr. Stanimire Tomov, and Dr. Azzam Haidar

PROJECT BY:

HELEN ZHOU

ZHEN ZHANG

MICHAELA SHOFFNER

INTRODUCTION TO PROJECT

- Our project concerns image unmixing using linear algebra and three baseline models.
- Each of us is working on a distinct part: using machine learning to better predict the correct weights, improving on the least squares method for initial calculations, and porting the project over to c in preparation of adapting it to run in parallel on a gpu.

OVERVIEW OF PROBLEM SET UP

- What we have:
  - 3 true modes: M1, M2, and M0
  - 16 images, x, each consists of a combination of these 3 modes
  - 3 4x4 true weight matrices

What we want: the true model of the composition of the image, such that the calculated weights of the 3 modes equal the true weights.

Current model: linear least squares optimization:

$X = \alpha M1 + \beta M2 + \gamma M0$, with α, β, and γ being the weights of the three modes and x being the resulting image.

LEAST SQUARES SOLUTION

- $X = \alpha M1 + \beta M2 + \gamma M0 \leftrightarrow ||x-(\alpha M1+\beta M2+\gamma M0)||\_2=0$.

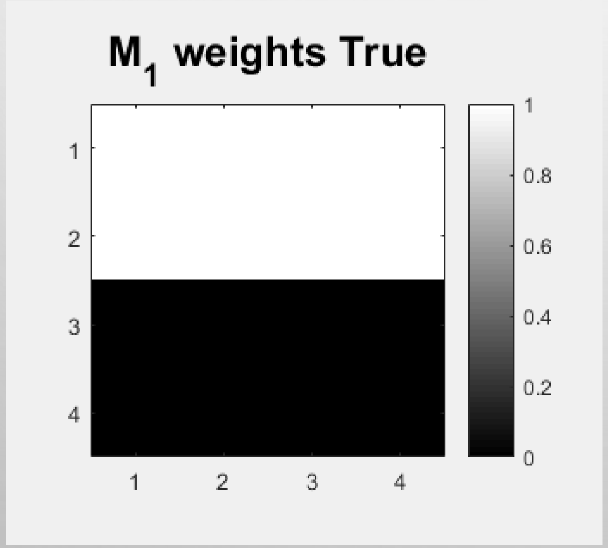
A=[M1 M2 M0] : 7,225,344-by-3 : data of the 3 modes.

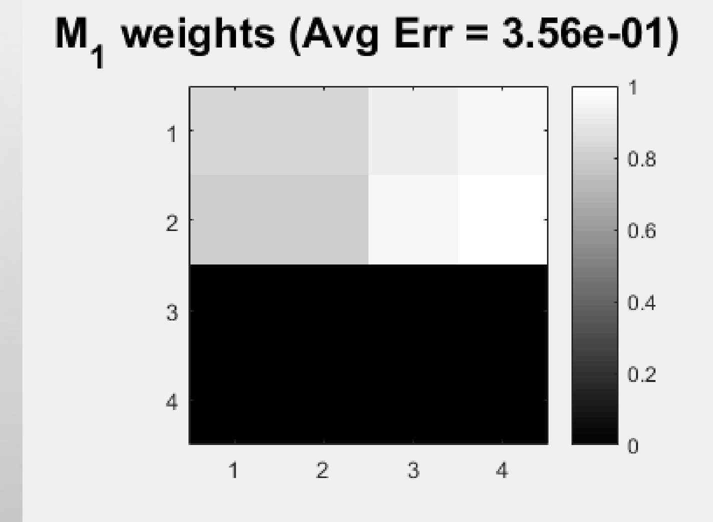
X: 7,225,344-by-1 : data of the resulting image.

Formulation: let w=[α; β; γ] (3-by1), find w that minimizes ||Aw-x||_2.

4-by-4 unit cells→4-by-4 weights for each mode.

Weight matrix :





L1-REGULARIZED LEAST SQUARES SOLUTION

- Problem: solve Aw=x and meanwhile, minimize |grad(w)|.
- Here |grad(w)|= $\sum_{i=1}^{3}|w(i+1,:)-w(i,:)|+\sum_{j=1}^{3}|w(:,j+1)-w(:,j)|$
- Formulation: minimize ||Aw-x||_2 + λ |grad(w)|.
- Goal: find w that minimizes $\sum|grad(w)|_1+\sum M\left(||Aw-x||_2\right)^2$,
- ⇒ Split bregman method.
- Model: min | ϕ(u)|+H(u)

L1-REGULARIZED LEAST SQUARES SOLUTION
SPLIT BREGMAN ITERATION

Model: min | Φ(u)|+H(u)

E1, E2: 36-by-48 matrices for gradient calculation. For example, the first row of E1 is [1 0 0 -1 0 0 ⋯ 0], then the first row of E1*u is (u1-u4).

⇒ Φ1(u)=E1*u, Φ2(u)=E2*u.

A is diagonal in block sense, 16 diagonal blocks of [M1 M2 M0].

X contains all data in 16 units cells of the resulting image.

⇒H(u)=(||Au-x||_2)^2.

Split bregman iteration: use d1, d2 to approximate E1*u, E2*u.

Now the goal is: find w, d1, d2 that minimize

|d1|+|d2|+M(||Aw-x||_2)^2+(λ/2)(||E1*w-d1||_2)^2+(λ/2)(||E2*w-d2||_2)^2

⇒ Iteration

FINDING THE TRUE MODEL

From Dr. Archibald: x might be linear terms + some combination of the gradients of the 3 modes.

- Assume: $x = \alpha * M1 + \beta * M2 + \gamma * M0 + a * g1 + b * g2 + c * g0$
- ⇒Least square
- ⇒Closer to true weights

CONVERTING TO C

- The original program was made on matlab.
- Converting it to c, and doing matrix arithmetic with lapack, was done in order to improve speed.
- Lapack is a library of functions used for matrix calculations, primarily used in c and fortran. The most useful to me will be dgemm and dgels.

MAKING PARALLEL

Once the program is working in its new form, we plan to further increase its speed and efficiency by running it in parallel.

Since quite a bit of the time the program spends running is doing large matrix calculations, something easy to do in parallel, adapting it to run on a gpu should see a significant increase in speed.

Should be a further speed boost over only lapack.

MAKING PARALLEL CONTINUED

- Unfortunately, a lot of the time elapsed also goes to getting the data input and setting up the initial matrices.
- If, as I suspect, this cannot be easily done is parallel, that will put a significant limit on how much speed up we can expect to gain from the parallel algorithm.
- Binary files are also a possibility, but they're also more difficult to be sure they are implemented correctly. As well, the speed up may not be enough to be noticeable.

MACHINE LEARNING: GOAL

- Let M_0, M_1, M_2 be three modes, and I be a target image. We want to find a representation of I with M_0, M_1, M_2
- Each I is provided with three fixed coefficients, indicating the linear part of the dependence.
- The target is to find the representation of the nonlinear part.

MACHINE LEARNING: METHOD

- An ideal network should take an image as input and output the linear coefficients.
- But the problem is that, we do not know exactly the mathematical form of the bias, i.E.,

$$I - \alpha M_0 - \beta M_1 - \gamma M_2$$

- The method is to assume that for each pixel (x, y) in I, the bias for this pixel is

$$B_{x,y}(\alpha, \beta, \gamma)$$

- We can find this bias function with interpolation, provided that we have 16 sets of I
- with (α, β, γ) already given.

MACHINE LEARNING: METHOD

- When the form of bias is already known, we can generate as many synthetic data as we want.
- More thinking: essentially what this neural network is doing is to solve an equation.
- Then why don't we extend this idea further? Maybe we can solve a big linear system with neural network.
- Or maybe even other linear algebra problem may be solved with nn!

SOLVE LINEAR SYSTEM

- Cost function for solving linear system $Ax = b$:
- $$\Sigma_i ||A\Theta b_i - b_i|| \ \ or \ \ \Sigma_i ||\Theta x_i - x_i||$$
- Gradient: $\Sigma_i A^T(A\Theta b_i - b_i)b_i^T \ or \ \Sigma_i(\Theta b_i - x_i)x_i^T$
- For the first cost function, I prove that the spectral radius is:
- $max_{m,j}|1 - \Delta t \lambda^{(j)} \sigma_m^2|$

Δt is the time step,

$\lambda^{(j)}$ are eigenvalues of $\Sigma_i b_i b_i^T$

σ_m are singular values of A

EIGENVALUE/VECTOR

- I have not worked on this problem in detail.
- But the cost function may look like:

$$var(Ao_i./o_i)$$

o_i Is the output vector.

- If this function is minimized to 0, we will have an eigenvector of A.

Q & A

ANY QUESTIONS?