MASTER 2 MICAS

Machine Learning Communications and Security

Course : MICAS931 - Introduction to Optimization.

School : Institut Polytechnique de Paris

Teacher : HADI GHAUCH

Subject : Homework 2 - Submit before 22/11/2020

Etudiant : Panongbene Jean Mouhamed Sawadogo.
Email : panongbene.sawadogo@telecom-paris.fr

SOMMAIRE :

Part I : Closed-form solutions vs iterative algorithms

Consider the following linear regression problem :

$$w^* = \underset{w \in \mathbb{R}^d}{min} \frac{1}{N} \sum_{i \in [N]} \|w^T x_i - y_i\|_2^2 + \lambda \|w\|_2^2 \qquad (1)$$

where $\{(x_i, y_i)\}_{i=1}^N$ is the training set

a) Derive a closed-form solution for $w^* \in (1)$.

Answer : Define, the f function by for all w :

$$f(w) = \frac{1}{N} \sum_{i \in [N]} \|w^T x_i - y_i\|_2^2 + \lambda \|w\|_2^2$$

$$\Longrightarrow \boxed{f(w) = \frac{1}{N} \|Xw - Y\|_2^2 + \lambda \|w\|_2^2}$$

The $\|.\|_2^2$ function is strictly convex and the function w $\rightarrow Xw - Y$ is also strictly convex we can deduce that the f function is strictly convex.

So we have $w^* \in \{w, \nabla f(w) = 0\}$

We have :

$$\nabla f(w) = \nabla_w \left(\frac{1}{N} \|Xw - Y\|_2^2 + \lambda \|w\|_2^2 \right)$$

$$\Longrightarrow \nabla f(w) = \frac{2}{N} X^T (Xw - Y) + 2\lambda w$$

$$\Longrightarrow \nabla f(w) = 0 \Longleftrightarrow \frac{2}{N} X^T (Xw - Y) + 2\lambda w = 0$$

$$\Longleftrightarrow \frac{2}{N} X^T X w - \frac{2}{N} X^T Y + 2\lambda w = 0$$

$$\Longleftrightarrow \left(\frac{1}{N} X^T X + \lambda I_d \right) w = \frac{1}{N} X^T Y$$

$$\iff w = \frac{1}{N}\left(\frac{1}{N}X^TX + \lambda I_d\right)^{-1} X^TY$$

$$\boxed{w^* = \frac{1}{N}\left(\frac{1}{N}X^TX + \lambda I_d\right)^{-1} X^TY}$$

Derive an approximation of the computational complexity (in terms of $\mathcal{O}(-)$ analysis) for this solution.

Answer :

To caculate the value of w^*, we will need to calculate the sum $B = X^TY$ and the matrix $A = \frac{1}{N}X^TX + \lambda I_d$. We also need to calculate the inverse of A matrix. So we can say that the complexity of calculatind w^* is :

$$\mathcal{O}\left(d^3 + d^2 + N^2\right)$$

where :

- d is the number of columns of dataset

- N is the number of items in the dataset

b) Consider "Communities and Crime" dataset (N = 1994, d = 128) and find the optimal linear regressor from the closed-form expression.

Answer : We have for $\lambda = 0.5$:

w^* = [-1.40840651e-03 -5.11872434e-06 1.01032221e-06 7.95785113e-06 2.78852086e-03 1.05531972e-02 2.57762552e-03 3.99994736e-02 -2.31810744e-02 1.41382231e-03 8.65773116e-03 4.02389149e-03 7.23309452e-03 3.82837025e-03 8.88591548e-03 1.07765823e-02 2.25854474e-02 -8.06565756e-03 -2.10583930e-03 -3.76908468e-03 -1.05013705e-02 1.00340974e-02 2.48300741e-02 3.96969660e-03 -7.84824569e-03 -3.59596793e-03 2.62525884e-03 -3.63875369e-03 8.10871436e-04 2.26280796e-03 2.23882777e-03 6.91253831e-04 1.23620786e-02 2.23510481e-02 1.49220439e-02 1.95676308e-02 -4.31351548e-03 2.01789345e-02 -1.64395460e-03 1.53924609e-03 3.92115245e-03 1.27004244e-02 -2.28373507e-03 2.64610644e-02 1.39745186e-02 2.56499307e-02 2.72057915e-02 8.59397992e-03 -1.95864847e-02 -2.13905159e-02 -1.93682369e-02 -1.73307834e-02 6.27519534e-03 2.57005346e-03 1.15313171e-

02 3.86079296e-02 4.59245348e-03 7.62923281e-03 9.08148484e-03 1.10929395e-02 1.32058224e-02 7.15869179e-03 8.05328088e-03 8.71357150e-03 9.24764935e-03 6.36504775e-03 7.96288155e-03 1.41303032e-02 1.01972618e-02 3.23512331e-03 1.17866314e-03 1.12700408e-02 -9.11520466e-03 1.56318485e-02 1.95741609e-02 -8.74499306e-03 1.57488587e-02 -4.79808362e-03 -6.36390831e-03 2.46148659e-02 7.10064589e-03 3.52510796e-03 2.40277456e-02 1.40513626e-02 -4.98136646e-03 -4.03685119e-03 -2.71479517e-03 -4.46621223e-03 -2.42568224e-03 -2.68317760e-03 -1.70930745e-03 1.50638722e-02 7.95093836e-03 7.39972366e-03 9.05585918e-03 8.24050036e-03 6.95205990e-03 4.55904920e-03 3.30060805e-03 1.20473124e-02 7.18419389e-03 3.90017723e-03 4.97944701e-03 9.92914000e-03 5.42316116e-03 4.83470907e-03 6.11487878e-03 7.32782823e-03 4.98269145e-03 3.20927101e-03 4.84287020e-03 1.10571678e-02 3.31105231e-03 4.63511158e-03 9.04893380e-03 3.44909658e-03 8.33772871e-03 4.35170483e-03 6.17359734e-03 1.17204760e-02 1.03565264e-02 7.76194363e-03 3.92557290e-03 7.83145788e-03 7.62747129e-03 1.87035062e-02 4.31859274e-03]

The optimal linear regression is

$$\boxed{f(w^*) = 0.07036504500486998}$$

What is the complexity in that case for this value of N ?

Answer : In this case, we have N=1994 and d = 128 so the complexity is given by :

$$\boxed{\mathcal{O}\left(128^3 + 128^2 + 1994^2\right) = \mathcal{O}\left(6089572\right)}$$

c) Repeat b) for "Individual household electric power consumption" dataset (N = 2075259, d = 9). Observe the scalability issue of the closed-form expression.

Answer : we have :

w^* =[-4.91684585e-04 6.74100998e-06 4.52048650e-02 4.78788670e-02 4.77524664e-02 4.86878169e-02 4.83839119e-02]

The optimal linear regression is

$$\boxed{f(w^*) = 0.05322384502321367}$$

SCALABILITY : We observe scalability of the parameter w.

The complexity in that case ?

Answer : In this case, we have N=2075259 and d = 9 so the complexity is given by :

$$\boxed{\mathcal{O}\left(9^3 + 9^2 + 2075259^2\right) = \mathcal{O}\left(4306699917891\right)}$$

d) Derive a simple gradient algorithm to solve (1), iteratively..

Answer : We can consider the following gration descent algorithm :

$$w_{k+1} = w_k - \alpha \nabla f(w_k), \qquad where \qquad \alpha > 0$$

$$\Longrightarrow w_{k+1} = w_k - \frac{2\alpha}{N} X^T \left(X w_k - Y\right) + 2\alpha\lambda w_k$$

$$\Longrightarrow w_{k+1} = w_k - 2\alpha \left(\frac{1}{N} X^T X + \lambda I_d\right) w_k + \frac{2\alpha}{N} X^T Y$$

$$\boxed{w_{k+1} = w_k - 2\alpha \left(\frac{1}{N} X^T X + \lambda I_d\right) w_k + \frac{2\alpha}{N} X^T Y}$$

Algorithm :

step 1- Choose $\epsilon > 0$, the stop condition and $\alpha > 0$

step 2- Initialize $w_k = w_0$

step 3- While $\|\nabla f(w_k)\|_2 \leqslant \epsilon$

$w_k = w_k - \alpha \nabla f(w_k)$

step 5- return w_k

How do you select the 'best' stepsize ? argue your choice.

Answer : we can select the step size $\alpha = \frac{2}{L+\mu}$ where L and μ is respectively the max and the min of eigenvalues of A=$\frac{1}{N} X^T X + \lambda I_d$ matrix.

As we demonstrated in the hw1 this step is the best to have a good time of convergence.

Derive an approximation of the computational complexity (in terms of $\mathcal{O}(-)$ analysis) for this method. Compare it with that of the closed form solution.

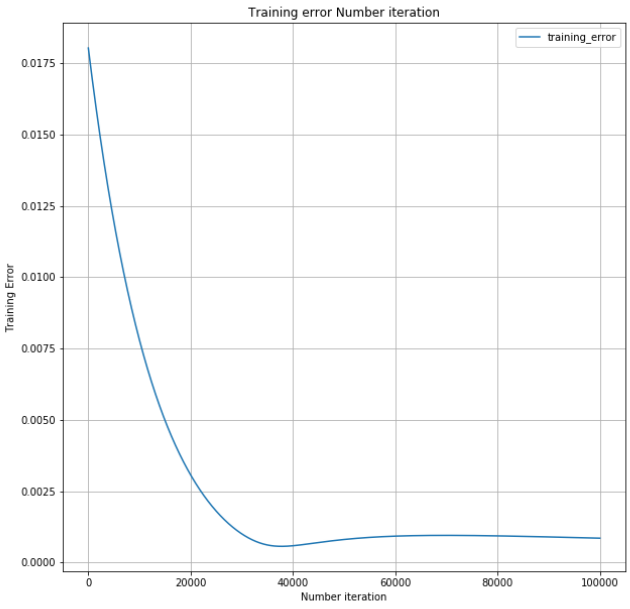
Answer : For a given step, the calculation of w requires to calculate the matrix A=$\frac{1}{N}X^TX + \lambda I_d$ and the sum B=$\frac{1}{N}X^TY$ but unlike the previous one we do not need a matrix inversion. therefore, the complexity in this case is equal to :

$$\mathcal{O}(N^2 + d^2)$$

This complexity is smaller than in the previous case.

Implement this algorithm using the "Individual household electric power consumption"
Plot the training error (using l_2 loss function) vs the number of iterations, against the optimal closed-form solution.

Answer : we have :



Verify that the gradient descent eventually converges to the optimal solution. Argue the reason for that.

Answer : It is easily verified that the descent of the gradient minimizes the error and stabilizes towards the optimal solution. Indeed, the direction of descent of the gradient points towards the minimum of the loss function.

Part II :Deterministic vs Stochastic algorithms

Now consider logistic ridge regression

$$\underset{w\in\mathbb{R}^d}{arg\,min} = \frac{1}{N}\sum_{i\in[N]} f_i(w) + \lambda\|w\|_2^2,\ where\ f_i(w) = log\left(1 + exp\left\{-y_i w^T x_i\right\}\right)$$

for the "Individual household electric power consumption" dataset (N = 2075259, d = 9).
This is supervised learning problem, specifically, a classification problem, i.e., the training set is defined as :$\left\{(x_i, y_i)_{i=1}^N\right\}$, where $x_i \in \mathbb{R}^d$ is a feature vector, and $y_i \in \mathbb{B}$ is a binary label. Solve the optimization problem using GD, plain stochastic GD, and SVRG.

0-a) Is f Lipschitz continuous ? If so, find a small Lipschitz constant L ?

Answer : We have for all w :

$$f(w) = \frac{1}{N}\sum_{i\in[N]} f_i(w) + \lambda\|w\|_2^2$$

$$\Longrightarrow f(w) = \frac{1}{N}\sum_{i\in[N]} \left(f_i(w) + \lambda\|w\|_2^2\right)$$

Let $g_i(w) = f_i(w) + \lambda\|w\|_2^2$

then we have :

$$f(w) = \frac{1}{N}\sum_{i\in[N]} g_i(w)$$

NB : we consider the functions g_i thus defined in the other questions

Let B > 0 and define $\mathcal{H} = \left\{ w \in \mathbb{R}^d \ \|w\|_2 \leq B \right\}$

For all i $\in [N]$ and $w \in \mathcal{H}$ we have :

$g_i(w) = f_i(w) + \lambda \|w\|_2^2$

$\implies g_i(w) = log\left(1 + exp(-y_i w^T x_i)\right) + \lambda \|w\|_2^2$

$$\implies \nabla g_i(w) = \frac{-y_i x_i exp(-y_i w^T x_i)}{1 + exp(-y_i w^T x_i)} + 2\lambda w$$

we have :

$exp(-y_i w^T x_i) \leqslant 1 + exp(-y_i w^T x_i)$

$$\implies \frac{exp(-y_i w^T x_i)}{1 + exp(-y_i w^T x_i)} \leqslant 1$$

$\implies \|\nabla g_i(w)\|_2 \leqslant \|y_i x_i\|_2 + 2\lambda \|w\|_2$

$\implies \|\nabla g_i(w)\|_2 \leqslant \|y_i x_i\|_2 + 2\lambda B$

$$\implies \frac{1}{N} \sum_{i \in [N]} \|\nabla g_i(w)\|_2 \leqslant \frac{1}{N} \sum_{i \in [N]} \|y_i x_i\|_2 + 2\lambda B$$

$$\implies \|\nabla f(w)\|_2 \leqslant \frac{1}{N} \sum_{i \in [N]} \|y_i x_i\|_2 + 2\lambda B$$

So f is lipschtz continous and we can take :

$$\boxed{L = \frac{1}{N} \sum_{i \in [N]} \|y_i x_i\|_2 + 2\lambda B}$$

0-b) Is f strongly convex ? If so, find a large μ ?

Answer : Consider this function : $h(w) = \lambda \|w\|_2^2$ then :

$\nabla h(w) = 2\lambda w$

$\implies \nabla^2 h(w) = 2\lambda I_d$

$\Longrightarrow \nabla^2 h(w) \succ 2\lambda I_d$

So, the h function is 2λ-strongly convex (**)

$\nabla f_i(w) = \nabla_w \left(log(1 + exp(-y_i w^T x_i))\right)$

$\Longrightarrow \nabla f_i(w) = \dfrac{-y_i x_i exp(-y_i w^T x_i)}{1 + exp(-y_i w^T x_i)}$

$\Longrightarrow \nabla^2 f_i(w) = \dfrac{y_i^2 x_i x_i^T exp(-y_i w^T x_i)}{(1 + exp(-y_i w^T x_i))^2}$

$\Longrightarrow \nabla^2 f_i(w) \succcurlyeq 0$

so we can say that the function f_i is convex (***)

(**) and (***) $\Longrightarrow g_i(w) = f_i(w) + h(w)$ is 2λ-strongly convex

So, $f(w) = \dfrac{1}{N} \sum_{i \in [N]} g_i(w)$ is 2λ-strongly convex and we can choose

$$\boxed{\mu = 2\lambda}$$

0-c) Can you use the same "trick" as Part III of HW1 to find L and μ ?

Answer : we ca,=n determinate the value of L and μ using the same strick in Part III of HW1 i.e :

L= max $\{\lambda_i\}_{i \in [N]}$

μ= min $\{\lambda_i\}_{i \in [N]}$

where $\{\lambda_i\}$ is the eigenvalues of the matrix $\nabla^2 f(w)$

0-d) What can you say about the ratio $\frac{L}{\mu}$? is it a 'good' or 'bad' setup for a plain GD method, for the dataset considered here ?

Answer : we have :

$$\nabla^2 f(w) = \frac{1}{N} \sum_{i \in [N]} \left(\frac{y_i^2 x_i x_i^T exp(-y_i w^T x_i)}{(1 + exp(-y_i w^T x_i))^2} + 2\lambda I_d \right)$$

So we can see to compute $\nabla^2 f(w)$ depend of the size os dataset. So, determinated the eigenvalues of $\nabla^2 f(w)$ depends also of size of dataset : more the dataset is large, more the computational complexity for to determinate eigenvalues is difficults.

So for this case, the ratio $\frac{L}{\mu}$ is bad for a plain GD method, for the dataset considered here.

a) Write/Derive the update equations for each algorithm, and implement them

Answer : For each equation, we consider that the step is constant equal α.

- For GD :
The update equation for gradient descent is given by :

$$w_{k+1} = w_k - \alpha \nabla f(w_k), \qquad where \qquad \alpha > 0$$

We have $f(w) = \frac{1}{N} \sum_{i \in [N]} g_i(w)$

$$\Longrightarrow \nabla f(w) = \frac{1}{N} \sum_{i \in [N]} \nabla g_i(w)$$

$$\Longrightarrow \nabla f(w) = \frac{1}{N} \sum_{i \in [N]} \left(\frac{-y_i x_i exp(-y_i w^T x_i)}{1 + exp(-y_i w^T x_i)} + 2\lambda w \right)$$

$$\Longrightarrow \nabla f(w_k) = \frac{1}{N} \sum_{i \in [N]} \left(\frac{-y_i x_i exp(-y_i w_k^T x_i)}{1 + exp(-y_i w_k^T x_i)} + 2\lambda w_k \right)$$

So, the update equation for GD is :

$$\Longrightarrow \boxed{w_{k+1} = w_k - \frac{\alpha}{N} \sum_{i \in [N]} \left(\frac{-y_i x_i exp(-y_i w_k^T x_i)}{1 + exp(-y_i w_k^T x_i)} + 2\lambda w_k \right)}$$

- For Plain Stochastic GD :
The update equation for plain stochastic gradient descent is given by :

$$w_{k+1} = w_k - \alpha \hat{\nabla} f(w_k), \qquad where \qquad \alpha > 0$$

Where :
$\alpha > 0$

$\hat{\nabla} f(w) = \frac{1}{|\mathcal{C}_k|} \sum_{i \in \mathcal{C}_k} \nabla g_i(w)$ where $\mathcal{C}_k$ is an i.i.d random of subset of $[N]$.

- For SVRG :

The update equation for SVRG is given by :

$$w_{k,t} = w_{k,t-1} - \alpha \left(\nabla f_{\mathcal{C}_k}(w_{k,t-1}) - \nabla f_{\mathcal{C}_k}(\tilde{w}_k) + \tilde{\nabla} f \right),$$

Where :

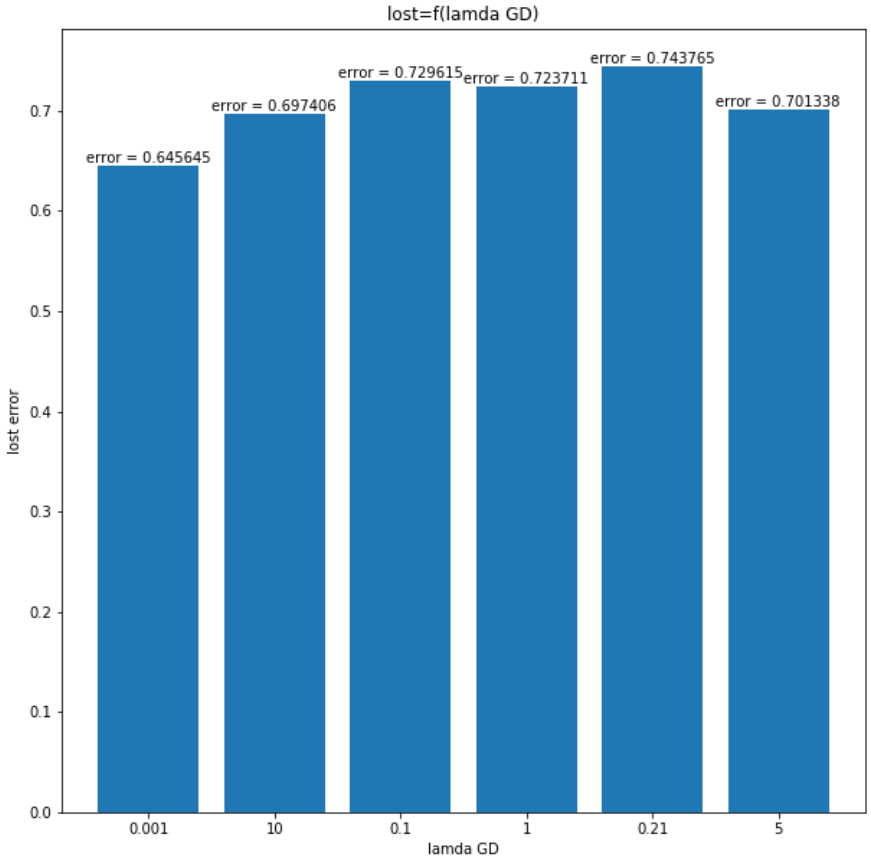
$\alpha > 0$

$\nabla f_{\mathcal{C}_k}(w) = \frac{1}{|\mathcal{C}_k|} \sum_{i \in \mathcal{C}_k} \nabla g_i(w)$ where $\mathcal{C}_k$ is an i.i.d random of subset of $[N]$.

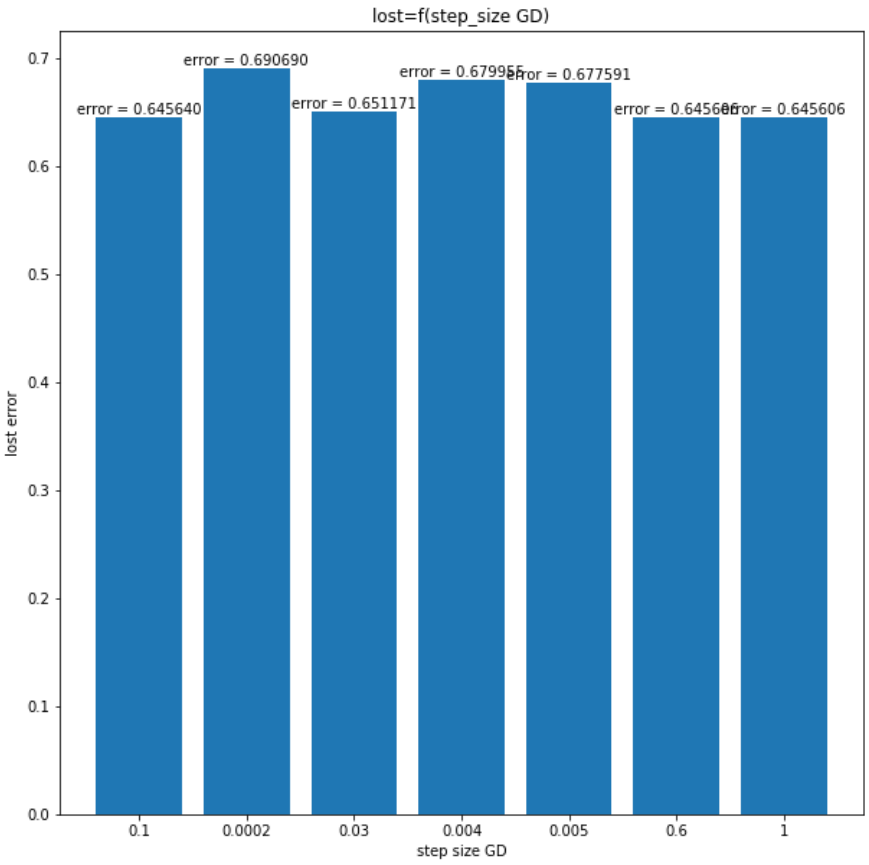
$\tilde{\nabla} f = \frac{1}{N} \sum_{i \in [N]} \nabla g_i(w)$

b) Tune the hyper-parameters for algorithm (including λ), using a held out cross validation test set.

- Gradient Descent : For the gradient descent we need to determine two parameters : the λ and the step size α.

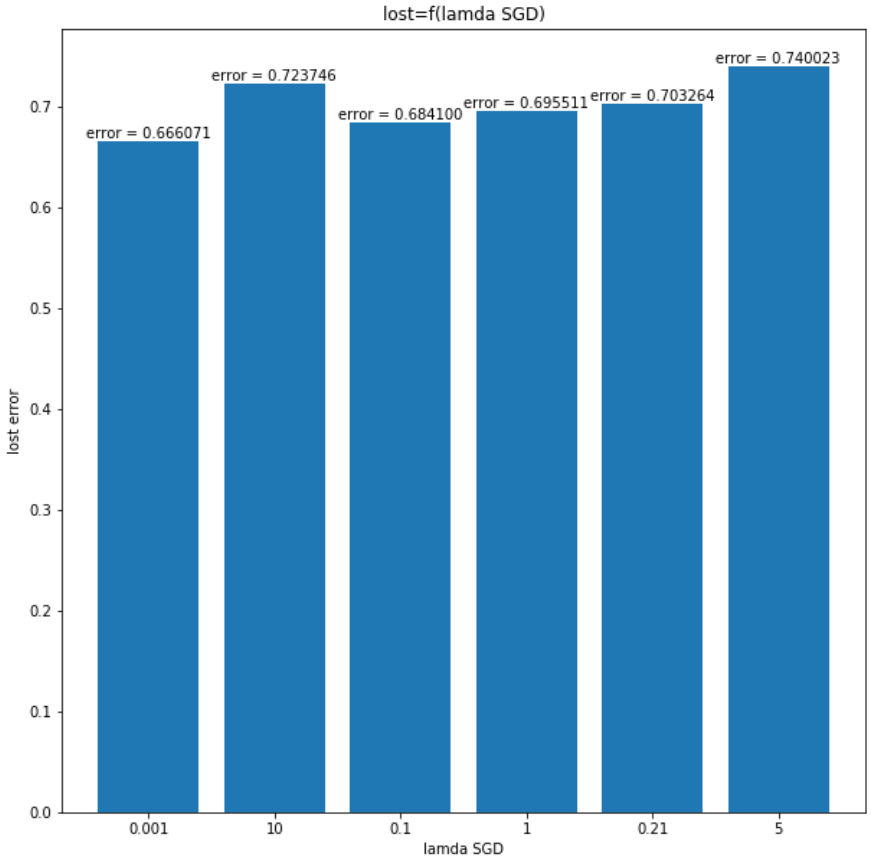


By fixing the size step at 0.01 and by carrying out a cross validation on all the candidate parameters of λ with the GD algorithm, we obtain the histogram above. We can therefore conclude that the best λ for our implementation is 0.001.

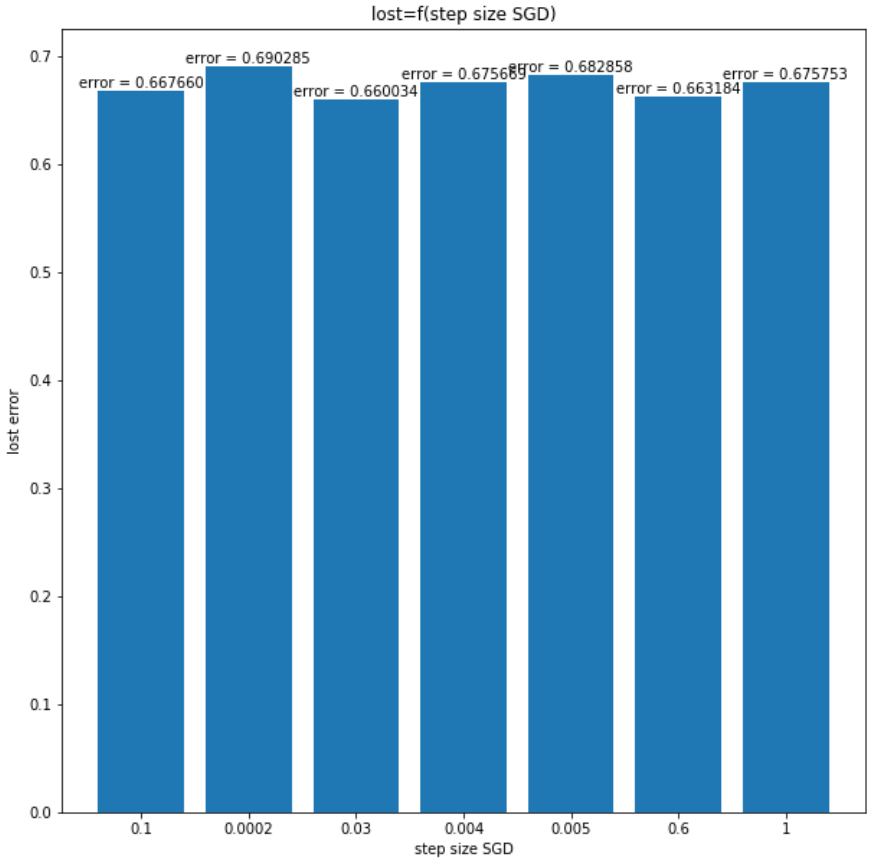


By fixing λ at 0.001 and by carrying out a cross validation on all the candidate parameters of step size with the GD algorithm, we obtain the histogram above. We can therefore conclude that the best step size for our implementation is 0.1.

- Stochastic Gradient Descent : For the stochastic gradient descent we need to determine three parameters : the lamda, the step size and the optimal batch size.



By fixing the size step at 0.01 and the batch size at 1 and by carrying out a cross validation on all the candidate λ parameters with the SGD algorithm, we obtain the histogram above. We can therefore conclude that the best λ for our implementation is 0.001.

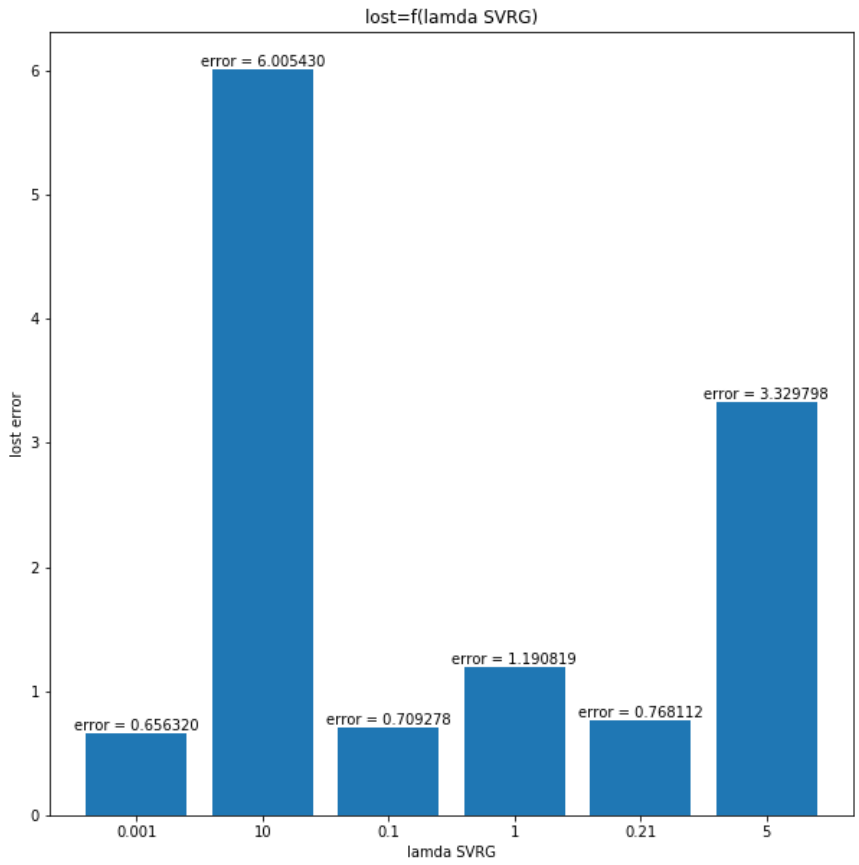


By fixing the λ at 0.01, the size batch at 1 and by carrying out a cross validation on all the candidate step size parameters with the SGD algorithm, we obtain the histogram above. We can therefore conclude that the best step size for our implementation is 0.03.

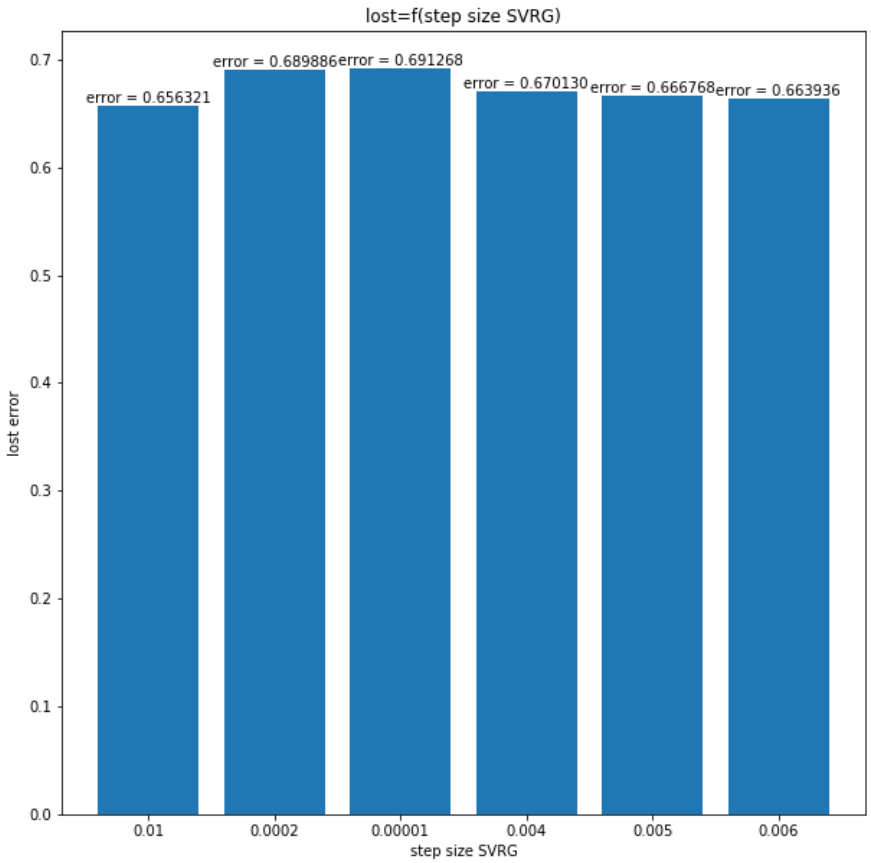
The larger the batch used to train the SGD model the better the performance, and if we take a batch of data equal to the size of the data, we find the GD algorithm. Thus, the best batch correspond to the size of the data.

- SVRG(Stochastic Variance reduced Gradient) : For the SVRG, we need to determine four parameters : the λ, the step size, the epoch and the optimal batch size.

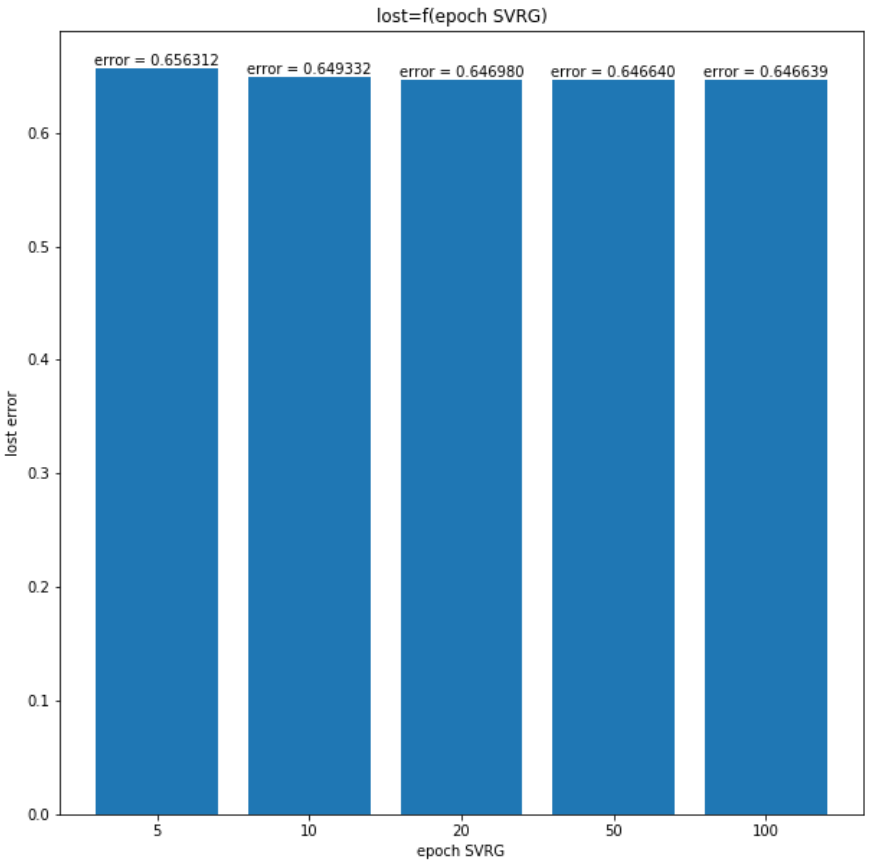
By setting the size step to 0.01, the batch size to 1 and the epoch size to



5 and performing a cross validation on all the candidate lamda parameters with the SVRG algorithm, we obtain the histogram below. above. We can therefore conclude that the best lamda for our implementation is 0.001.



By setting lamda to 0.01, the batch size to 1, and the epoch size to 5 and performing cross validation on all candidate step size parameters with the SVRG algorithm, we get the histogram shown. -above. We can therefore conclude that the best step size for our implementation is 0.01.



By setting lamda to 0.01, the size step to 0.01 and the batch size to 1 and performing a cross validation on all candidate epoch size parameters with the SVRG algorithm, we get the histogram above. We can therefore conclude that the best epoch size for our implementation is 100.

For the lot size, it is the same reasoning as for the SGD algorithm.

c) Compare all these methods in terms complexity of hyper-parameter tunning, convergence time, convergence rate (in terms of # outer-loop iterations), and memory requirement.

Answer :

→ complexity of hyper-parameter :

The only hyper parameter that is needed for the GD algorithm is the learning rate α and λ which can be obtained using cross validation.

On the other hand for the stochastic GD, it is necessary to determine in addition to the parameters learning rate and λ, the size of the batch which makes one more parameter. To simplify as in our implementation, instead of taking a variable batch size with each iteration, we set the starting batch size.

For the SVRG algorithm it is necessary to add in addition to the parameters necessary for the SGD, a new parameter of epoch T which brings the number of parameters to 4.

We can therefore say that the SVRG algorithm requires more parameter with 4 parameters and the BG algorithm requires less parameter with 2 parameters.

→ Convergence time : To compare the convergence time, we will use the complexity of the algorithms, that is to say the number of calculations required for each algorithm.

For the GD algorithm we need at each iteration to calculate the sum of the gradients of the functions g_i on all of our training data. However, for the SGD we also have to do the same calculation if we choose the batch size equal to the data size. That is to say in the worst case, the SGV requires as many calculations as the GD.

On the other hand, the SVRG algorithm requires more calculations due to the iteration to be done for the time.

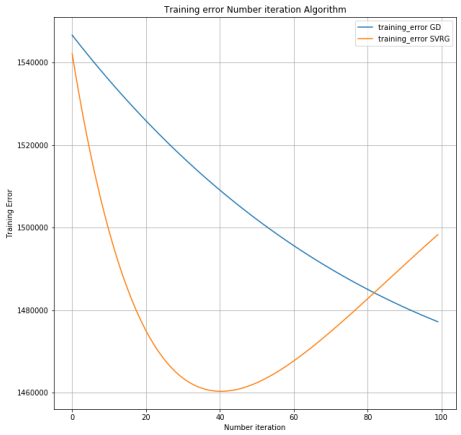
Thus, we can say that in terms of calculation time, the SGD algorithm requires less calculation time with a calculation time equal to that of the GD in the case of a size of the epochs equal to the size of our data. The SVRG algorithm is the one that needs more calculation.

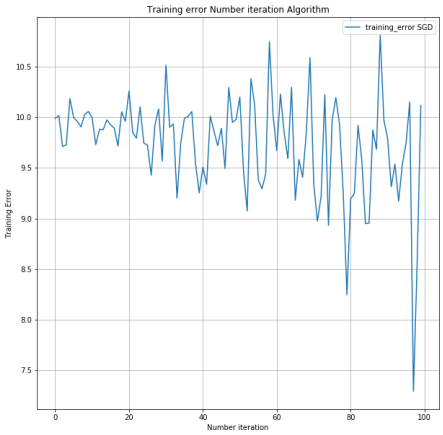
→ convergence rate : The convergence rate is faster for the GD algorithm followed by the SGD algorithm with equality in the case where the batch size is equal to the size of our data.

→ memory requirement : The SVRG algorithm requires more intermediate memory during model training operations. It is followed by the GD algorithm and at the end the SGD algorithm with equal memories in the worst case (when choosing a batch size equal to the size of the data).

d) Compare the training error (using l_2 loss function) of all these algorithms. Comment on these results : are they expected or is there any discrepancy ?

Answer :

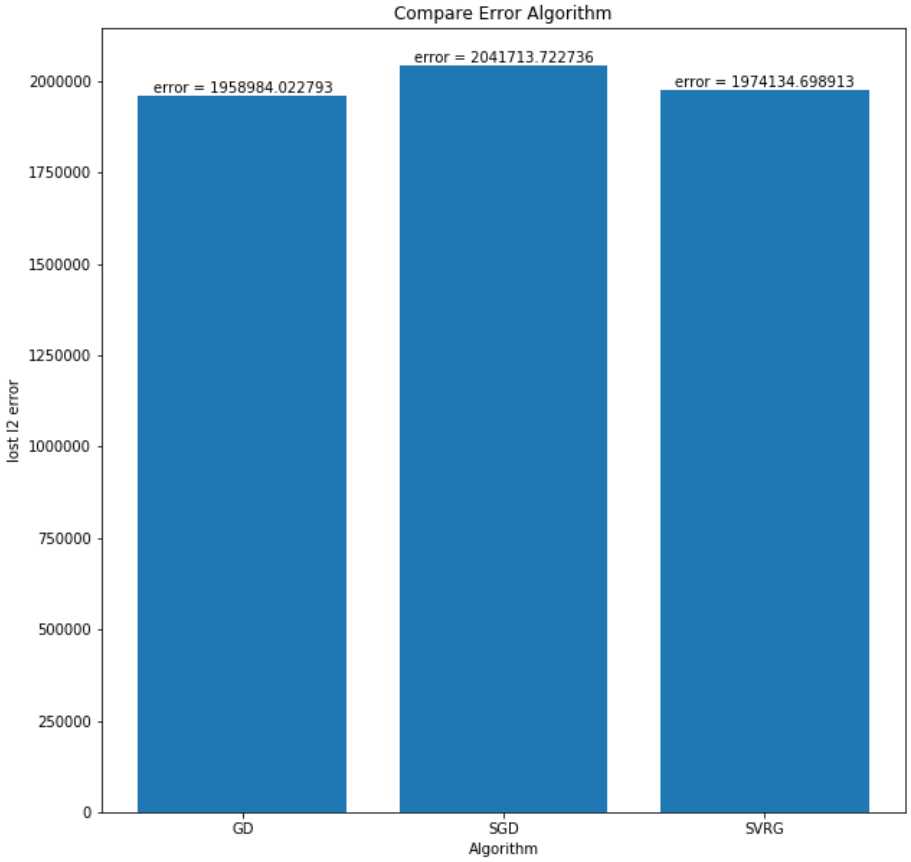




We can see from the figure above that the svrg algorithm converges more easily but it is unstable because when reaching its minimum, the error tends to increase unlike the GD algorithm. On the other hand, the SGD presents a curve with several irregularities but whose tendency is downward. These curves effectively reflect the mathematical demonstrations that have been made. Indeed, the instability of the SGD algorithm is due to the random effect of the selection of the elementary functions used to update the weights. The GD behaves normally : convergence is stable but slow.

e) Compare the test error (using l_2 loss function) of all these algorithms. Comment on these results : are they expected or is there any discrepancy ?

Answer : We see that the performance of the algorithms on the test data is



almost similar with the descent gradient which gives the best results followed by the SVRG algorithm. This seems to be the intuition one would expect indeed SVRG and SGD algorithms are approximations of the GD algorithm, so it is only natural that the SG gives better results.

Part III : Matrix Factorization for Recommendation System

Consider the low-rank matrix factorization (MF) for recommendation systems. Read carefully the problem setting in second lecture on non-convex optimisation (lecture 7)

- Data matrix $\mathbf{X}$ of size M $\times$ N . M = number of users, N = number of items

- $[X]_{u,i}$ is rating (score) that user u gave to item i

- In recommendation sys model rating as : $[X]_{u,i} = p_i^T q_u$,
$p_i \in \mathbb{R}^d$ vector of item characteristics ,$q_u \in \mathbb{R}^d$ vector of user preferences

- factorize $\mathbf{X}$ with low-rank matrices : $\mathbf{P} \in \mathbb{R}^{M\times d}, \mathbf{Q} \in \mathbb{R}^{N\times d}, d \ll (M, N)$ such that $\mathbf{X} = \mathbf{P}\mathbf{Q}^\mathbf{T}$

- every col. of $\mathbf{Q}^\mathbf{T}$ (resp. rom of $\mathbf{P}$) is user prefs (resp item characteristics)

We will build on that example by considering a realistic implementation using popular movie recommendation datasets, e.g., the MovieLens dataset. You will be solving the problem posed in Lecture 7, namely, the "Low-rank MF (supervised learning)" slide : please make sure you read the problem that is formulated in that slide.

We denote by $\mathcal{K}$ the set of **ratedentries** in $\mathbf{X}$ i.e, ntries of the rating matrix which have a score/rating by at least one user. Then the low-rank MF model is learned from the set of rated entries, $\mathcal{K}$, e.i.,

$$\underset{\{p_i,q_u\}_{(u,i)\in\mathcal{K}}}{argmin} \sum_{(u,i)\in\mathcal{K}} \left[\Big([X]_{u,i} - p_i^T q_u \Big)^2 + \lambda_i \|p_i\|_2^2 + \mu_u \|q_u\|_2^2 \right]$$

In general, $\{\lambda_i > 0, \mu_u\}_{(u,i)\in\mathcal{K}}$ are regularization parameters to prevent overfitting, tuned via cross-validation. However, here we will focus on the training part (rather than the testing) and assume that all these regularization parameters are **strictly positive** and **given** to us. Due to the presence of coupling in the cost function, Block Coordinate Descent (BCD) methods are prevalent for solving (2). We thus propose to use the following standard methods to solve (2) : Block-Coordinate Descent (BCD) with closed-form solution, and BCD with Gradient Descent (GD).

0a) Starting from (2), apply the BCD method to derive the subproblem for each block of coordinates, In other words, derive the subproblem corresponding to each of the optimization variables in (2), $\{p_i, q_u\rangle_{(u,i)\in\mathcal{K}}$

Answer : Let f function define by :

$$f(p_i, q_u)_{|(i,u)\in\mathcal{K}} = \sum_{(u,i)\in\mathcal{K}} \left[\left([X]_{u,i} - p_i^T q_u \right)^2 + \lambda_i \|p_i\|_2^2 + \mu_u \|q_u\|_2^2 \right]$$

So, the subproblems corresponding to the optimization are given by :

for each i, $\hat{p}_i = \underset{p_i}{argmin} \sum_{(u,i)\in\mathcal{K}} \left[\left([X]_{u,i} - p_i^T q_u \right)^2 + \lambda_i \|p_i\|_2^2 + \mu_u \|q_u\|_2^2 \right]$

for each u, $\hat{q}_u = \underset{q_u}{argmin} \sum_{(u,i)\in\mathcal{K}} \left[\left([X]_{u,i} - p_i^T q_u \right)^2 + \lambda_i \|p_i\|_2^2 + \mu_u \|q_u\|_2^2 \right]$

0b) show that each subproblem is strongly convex.

Answer : The function $x \to \|x\|_2^2$ is strongly convex and the functions $p_i \to \left([X]_{u,i} - p_i^T q_u \right)^2$, $q_u \to \left([X]_{u,i} - p_i^T q_u \right)^2$ are strongly convex.
Each subproblem is a linear combination of these functions. we can therefore say that each subproblem is strictly convex.

1) **BCD with closed − form solution** : also called alternating least-squares - derive the update for each of the subproblems (part 0a)) in **closed − form solution**

Answer : For each i, we have :

$$\nabla_{p_i} f = \nabla_{p_i} \sum_{(u,i)\in\mathcal{K}} \left[\left([X]_{u,i} - p_i^T q_u \right)^2 + \lambda_i \|p_i\|_2^2 + \mu_u \|q_u\|_2^2 \right]$$

$$\Longrightarrow \nabla_{p_i} f = -2q_u \left([X]_{u,i} - p_i^T q_u \right) + 2\lambda_i p_i$$

$$\Longrightarrow \nabla_{p_i} f = -2q_u [X]_{u,i} + 2q_u p_i^T q_u + 2\lambda_i p_i$$

$$\Longrightarrow \nabla_{p_i} f = -2q_u [X]_{u,i} + 2q_u q_u^T p_i + 2\lambda_i p_i$$

$$\Longrightarrow \nabla_{p_i} f = -2q_u [X]_{u,i} + 2 \left(q_u q_u^T + \lambda_i I_d \right) p_i$$

So we have :

$$\nabla_{p_i} f = 0 \iff -2q_u \left[X\right]_{u,i} + 2\left(q_u q_u^T + \lambda_i I_d\right) p_i = 0$$

$$\iff q_u \left[X\right]_{u,i} = \left(q_u q_u^T + \lambda_i I_d\right) p_i$$

$$\iff p_i = \left[X\right]_{u,i} \left(q_u q_u^T + \lambda_i I_d\right)^{-1} q_u$$

$$\boxed{p_i = \left[X\right]_{u,i} \left(q_u q_u^T + \lambda_i I_d\right)^{-1} q_u}$$

Similarly, we have for each u :

$$\nabla_{q_u} f = \nabla_{q_u} \sum_{(u,i)\in\mathcal{K}} \left[\left(\left[X\right]_{u,i} - p_i^T q_u\right)^2 + \lambda_i \|p_i\|_2^2 + \mu_u \|q_u\|_2^2\right]$$

$$\implies \nabla_{q_u} f = -2p_i \left(\left[X\right]_{u,i} - p_i^T q_u\right) + 2\mu_u q_u$$

So we have : $\implies \nabla_{q_u} f = 0 \iff -2p_i \left(\left[X\right]_{u,i} - p_i^T q_u\right) + 2\mu_u q_u = 0$

$$\iff p_i \left[X\right]_{u,i} = p_i p_i^T q_u + \mu_u q_u$$

$$\iff p_i \left[X\right]_{u,i} = \left(p_i p_i^T + \mu_u I_d\right) q_u$$

$$\iff \left(p_i p_i^T + \mu_u I_d\right) q_u = p_i \left[X\right]_{u,i}$$

$$\iff q_u = \left(p_i p_i^T + \mu_u I_d\right)^{-1} p_i \left[X\right]_{u,i}$$

$$\boxed{q_u = \left(p_i p_i^T + \mu_u I_d\right)^{-1} p_i \left[X\right]_{u,i}}$$

- show that algorithm which results from these updates **converges monotonically to a stationary point** of (2)

Answer : The problem (2) being strictly convex, it therefore admits a single global minimum which cancels out the derivative of f.
The solution resulting from the convergence of the BCD algorithm forms a local minimum of f and since f admits a single local minimum which happens to be the global minimum of f therefore the solution of the BCD algorithm

converges monotonically towards the stationary point of f (2).

- derive an estimate of the **computational complexity** per BCD iteration (in $\mathcal{O}()$ notation)

Answer : The calculation of the solutions of the subproblems of the BCD algorithm in our problem, requires to calculate the inverse K of matrix of size d followed by a product with a vector and sum.
Thus, we can give an estimate of the complexity which will be at least equal to

$$\mathcal{O}(d^3 + d^2 + N)$$

2) **BCD with GD** : also called block-gradient descent
- In part 1) you derived a closed-form solution for each of the subproblems. For this method, do not solve each subproblem optimally, but rather take T **GD steps** (T is a small positive integer) with respect to the block of coordinates being optimized for that subproblem. Derive the update equations that result from this strategy, as a function T.
Answer :

- what happens as $T \to \infty$?
- can you show the **convergence to a stationary point** of the resulting algorithm using the framework of BCD ?
- derive an estimate of the **computational complexity** per BCD iteration, as a function T (in $\mathcal{O}()$ notation)

3 Implement all your algorithms using the MovieLens 100K dataset.

- pick a value for model complexity, d, and plot the training error (empirical risk in (2)) using the ‘ 2 loss function, for each of the algorithm.

- plot the training error of BCD with GD for increasing value of T . what are the pros and cons of increasing T ?