國立臺灣大學電機資訊學院資訊工程學研究所

碩士論文

Department of Graduate Institute of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Dissertation

Newton Methods for Factorization Machines

黃郁庭

Yu-Ting Huang

指導教授：林智仁 博士

Advisor: Chih-Jen Lin, Ph.D.

中華民國 108 年 00 月

XX, 2019

中文摘要

...

關鍵詞: ...

ABSTRACT

...

KEYWORDS:

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

CHAPTER I

Introduction

Given a rating matrix R, matrix factorization (MF) is a technique to find two dense matrices $U \in \mathcal{R}^{d\times M}$ and $V \in \mathcal{R}^{d\times N}$ such that $r_{m,n} \simeq \boldsymbol{u}_m^T \boldsymbol{v}_n$, where $\boldsymbol{u}_m \in \mathcal{R}^d$ and $\boldsymbol{v}_n \in \mathcal{R}^d$ are respectively the mth column of U and the nth column of V, d is the pre-specified number of latent features, and the entry $r_{m,n}$ denotes the feedback of the mth user on the nth item. This task is achieved by solving the following non-convex problem

$$\min_{U,V} \quad \frac{1}{2}\sum_{(m,n)\in R}(r_{m,n} - \boldsymbol{u}_m^T \boldsymbol{v}_n)^2 + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2) \tag{1.1}$$

where $(m,n) \in R$ indicates that rating $r_{m,n}$ is available, λ is regularization coefficients for avoiding over-fitting, and $\|\cdot\|_F$ is the Frobenius norm. In fact, problem (1.1) is a special case of Factorization Machines (FM) (Rendle, 2010). The optimization problem solved by FM is as follows.

$$\min_{U,V} \quad f(U,V) \tag{1.2}$$

where

$$f(U,V) = \frac{1}{2}\sum_{i=1}^{l}\Big(y_i - (U\boldsymbol{w}_i)^T(V\boldsymbol{h}_i)\Big)^2 + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2) \tag{1.3}$$

and l is the total number of training instances. Problem (1.3) can be considered as a regression problem, where y_i is the label, and $\boldsymbol{h}_i \in \mathcal{R}^N$ and $\boldsymbol{w}_i \in \mathcal{R}^M$ are the feature vectors. If we let l be the number of non-zero entries in R, the value $y_i = r_{m,n}$ be the corresponding

rating from R,

$$\boldsymbol{w}_i = [\underbrace{0, \ldots, 0}_{m-1}, 1, 0, \ldots, 0]^T, \text{ and} \tag{1.4}$$

$$\boldsymbol{h}_i = [\underbrace{0, \ldots, 0}_{n-1}, 1, 0, \ldots, 0]^T, \tag{1.5}$$

then problem (1.2) is reduced to (1.1). Note that (1.2) is a variant of FM proposed in Blondel et al., (2016). One of its advantages is that the function in (1.3) is multi-block convex. That is, if U (or V) is fixed, then $f(U, V)$ is a convex function of V (or U). In this work, we focus on solving the optimization problem (1.2).

Many optimization methods have been proposed to minimize (1.2). We are particularly interested in Newton methods. Chin et al., (2018) have proposed an alternating Newton method and showed that it achieves state-of-the-art efficiency. The basic idea is to alternately switch between solving two sub-problems, each of which minimizes over one block of variables but fixes the other. Namely the first sub-problem minimizes over U while keeping V fixed, and the other minimizes over V while keeping U fixed.

Instead of alternately solving two sub-problems, naturally we ask why not minimizing $f(U, V)$ directly by the Newton method. To the best of our knowledge, no study has considered such a setting for factorization machines. In this work, we begin with developing a Newton method to solve problem (1.2). We then conduct detailed comparisons with the alternating Newton method.

CHAPTER II

Newton Method for Unconstrained Minimization

To derive the Newton method, we should re-write the function to have a vector of variables. To this end, the objective function in (1.3) is changed to

$$f(\mathrm{vec}(U), \mathrm{vec}(V)),$$

where $\mathrm{vec}(\cdot)$ stacks all columns of a matrix to a vector. For the standard Newton methods, at the kth iteration, we find a direction $\boldsymbol{s}_k$ minimizing the following second-order approximation of the function value:

$$\min_{\boldsymbol{s}} \quad \frac{1}{2}\boldsymbol{s}^T H_k \boldsymbol{s} + \boldsymbol{g}_k^T \boldsymbol{s} \tag{2.1}$$

where

$$H_k = \nabla^2 f(\mathrm{vec}(U_k), \mathrm{vec}(V_k)) \text{ and } \boldsymbol{g}_k = \nabla f(\mathrm{vec}(U_k), \mathrm{vec}(V_k))$$

are the Hessian matrix and the gradient of $f(U_k, V_k)$, respectively. If H_k is positive definite, then (2.1) is equivalent to solving the following linear system:

$$H_k \boldsymbol{s} = -\boldsymbol{g}_k \tag{2.2}$$

For non-convex objective functions, the Hessian matrix may not be positive definite. Then the solution of (2.2) may not lead to a descent direction. In Section IV we will show that this issue occurs for our optimization problem (1.2). To obtain a descent direction, we can consider a positive-definite approximation of the Hessian matrix.

Now assume that G_k is a positive definite approximation of the Hessian matrix. Instead of solving (2.2), we solve the following linear system to find a direction $\boldsymbol{s}_k$.

$$G_k \boldsymbol{s} = -\boldsymbol{g}_k \tag{2.3}$$

In practice, (2.3) may not need to exactly solved. Therefore, truncated Newton methods have been introduced to approximately solve (2.3). Iterative methods such as conjugate gradient method are often used for approximately solving (2.3), so at each iteration an inner iterative process is involved. In particular, if CG is is used, at each inner iteration a matrix-vector product between G_k and a vector must be conducted.

After a direction is found in our Newton method, we must decide the step size taken along that direction. A common setting is the backtracking line search. Namely, we find the largest step size $\theta \in \{1, \beta, \beta^2, \dots\}$ satisfying the following sufficient decrease condition. Let

$$\boldsymbol{s} = \operatorname{vec}(S) \tag{2.4}$$

and

$$S = \begin{bmatrix} S_u & S_v \end{bmatrix}. \tag{2.5}$$

$$f(U + \theta S_u, V + \theta S_v) - f(U, V) \leq \theta \nu \boldsymbol{g}^T \operatorname{vec}(S), \tag{2.6}$$

where $\nu \in (0, 1)$ and $\beta \in (0, 1)$ are pre-specified constants.

CHAPTER III

Newton Methods for Factorization Machines

To compute the gradient and Hessian of (1.2), we begin with the following properties of the Kronecker product among matrices A, B and X:

$$(B^T \otimes A)\,\mathrm{vec}(X) = \mathrm{vec}(AXB) \tag{3.1}$$

$$(A \otimes B)^T = A^T \otimes B^T. \tag{3.2}$$

Moreover, the predicted value can be represented as

$$(U\boldsymbol{w}_i)^T(V\boldsymbol{h}_i) = (\boldsymbol{h}_i^T \otimes (U\boldsymbol{w}_i)^T)\,\mathrm{vec}(V) \tag{3.3}$$

$$= (\boldsymbol{h}_i \otimes (U\boldsymbol{w}_i))^T\,\mathrm{vec}(V), \tag{3.4}$$

where (3.3) and (3.4) are from (3.1) and (3.2), respectively. The predicted value also can be represented as

$$\begin{aligned}(U\boldsymbol{w}_i)^T(V\boldsymbol{h}_i) &= (V\boldsymbol{h}_i)^T(U\boldsymbol{w}_i)\\ &= (\boldsymbol{w}_i^T \otimes (V\boldsymbol{h}_i)^T)\,\mathrm{vec}(U)\\ &= (\boldsymbol{w}_i \otimes (V\boldsymbol{h}_i))^T\,\mathrm{vec}(U).\end{aligned} \tag{3.5}$$

Then, we define

$$\boldsymbol{p}_i = U\boldsymbol{w}_i, \boldsymbol{q}_i = V\boldsymbol{h}_i, \tilde{y}_i = \boldsymbol{p}_i^T\boldsymbol{q}_i, b_i = \tilde{y}_i - y_i, \tag{3.6}$$

the loss function

$$\xi_i = b_i^2 = \left(\boldsymbol{p}_i^T \boldsymbol{q}_i - y_i\right)^2,$$

$$Q = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_l] \in \mathcal{R}^{d\times l}, P = [\boldsymbol{p}_1, \ldots, \boldsymbol{p}_l] \in \mathcal{R}^{d\times l},$$

$$W = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_l]^T \in \mathcal{R}^{l\times M}, H = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_l]^T \in \mathcal{R}^{l\times N}. \tag{3.7}$$

3.1 Gradient Calculation

The gradient of (1.2) is

$$\nabla f(\mathrm{vec}(U, V)) = \lambda\, \mathrm{vec}(U, V) + \frac{1}{2}\sum_{i=1}^{l} \frac{\partial \xi_i}{\partial \tilde{y}_i} \begin{bmatrix} \frac{\partial \tilde{y}_i}{\partial\, \mathrm{vec}(U)} \\ \frac{\partial \tilde{y}_i}{\partial\, \mathrm{vec}(V)} \end{bmatrix}. \tag{3.8}$$

From (3.5), (3.4) and (3.6),we define the following Jocobian of $\tilde{y}_i$ as

$$\begin{aligned} \boldsymbol{j}_i &= \begin{bmatrix} \frac{\partial \tilde{y}_i}{\partial\, \mathrm{vec}(U)} \\ \frac{\partial \tilde{y}_i}{\partial\, \mathrm{vec}(V)} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial (\boldsymbol{w}_i \otimes (V\boldsymbol{h}_i))^T\, \mathrm{vec}(U)}{\partial\, \mathrm{vec}(U)} \\ \frac{\partial (\boldsymbol{h}_i \otimes (U\boldsymbol{w}_i))^T\, \mathrm{vec}(V)}{\partial\, \mathrm{vec}(V)} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{w}_i \otimes \boldsymbol{q}_i \\ \boldsymbol{h}_i \otimes \boldsymbol{p}_i \end{bmatrix}. \end{aligned} \tag{3.9}$$

From (3.8) and (3.9), the gradient is

$$\begin{aligned}
\nabla f(\mathrm{vec}(U,V)) &= \lambda\,\mathrm{vec}(U,V) + \sum_{i=1}^{l} b_i \boldsymbol{j}_i \\
&= \lambda\,\mathrm{vec}(U,V) + \begin{bmatrix} \sum_{i=1}^{l} b_i(\boldsymbol{w}_i \otimes \boldsymbol{q}_i) \\ \sum_{i=1}^{l} b_i(\boldsymbol{h}_i \otimes \boldsymbol{p}_i) \end{bmatrix} && (3.10) \\
&= \lambda\,\mathrm{vec}(U,V) + \begin{bmatrix} \sum_{i=1}^{l} \mathrm{vec}\left(\boldsymbol{q}_i b_i \boldsymbol{w}_i^T\right) \\ \sum_{i=1}^{l} \mathrm{vec}\left(\boldsymbol{p}_i b_i \boldsymbol{h}_i^T\right) \end{bmatrix} && (3.11) \\
&= \lambda\,\mathrm{vec}(U,V) + \begin{bmatrix} \mathrm{vec}\left(\sum_{i=1}^{l} \boldsymbol{q}_i b_i \boldsymbol{w}_i^T\right) \\ \mathrm{vec}\left(\sum_{i=1}^{l} \boldsymbol{p}_i b_i \boldsymbol{h}_i^T\right) \end{bmatrix} \\
&= \lambda\,\mathrm{vec}(U,V) + \begin{bmatrix} \mathrm{vec}\Big(Q\big(\mathrm{diag}(\boldsymbol{b})W\big)\Big) \\ \mathrm{vec}\Big(P\big(\mathrm{diag}(\boldsymbol{b})H\big)\Big) \end{bmatrix}, && (3.12)
\end{aligned}$$

where $\mathrm{diag}(\boldsymbol{b})$ is a diagonal matrix in which the ith diagonal element is b_i, and for the second term in (3.11) we apply (3.1). Usually W and H is highly sparse matrices. The cost of gradient calculation is

$$\mathcal{O}\left(d \times (\mathrm{nnz}(W) + \mathrm{nnz}(H))\right),$$

where $\mathrm{nnz}(\cdot)$ is the number of non-zero elements in a sparse matrix.

3.2 Hessian and Gauss-Newton Matrices

The Hessian matrix of (1.2) is

$$\begin{aligned}
&\frac{\partial}{\partial \operatorname{vec}(U,V)^T}\frac{\partial f}{\partial \operatorname{vec}(U,V)} \\
=&\lambda I + \sum_{i=1}^{l} \frac{\partial}{\partial \operatorname{vec}(U,V)^T}(b_i \boldsymbol{j}_i) \\
=&\lambda I + \sum_{i=1}^{l} \Big(\frac{\partial}{\partial b_i^T}(b_i \boldsymbol{j}_i)\frac{\partial b_i}{\partial \operatorname{vec}(U,V)^T} + \frac{\partial}{\partial \boldsymbol{j}_i^T}(b_i \boldsymbol{j}_i)\frac{\partial \boldsymbol{j}_i}{\partial \operatorname{vec}(U,V)^T} \Big) \\
=&\lambda I + \sum_{i=1}^{l} \Big(\boldsymbol{j}_i \frac{\partial b_i}{\partial \tilde{y}_i}\frac{\partial \tilde{y}_i}{\partial \operatorname{vec}(U,V)^T} + b_i I \frac{\partial \boldsymbol{j}_i}{\partial \operatorname{vec}(U,V)^T} \Big) \\
=&\lambda I + \sum_{i=1}^{l} \boldsymbol{j}_i \boldsymbol{j}_i^T + \sum_{i=1}^{l} b_i \frac{\partial \boldsymbol{j}_i}{\partial \operatorname{vec}(U,V)^T},
\end{aligned} \tag{3.13}$$

where I is the identity matrix. This matrix may not be positive definite. We remove the last term in (3.13) and obtain the following Gauss-Newton matrix, which is positive definite.

$$G = \lambda I + \sum_{i=1}^{l} \boldsymbol{j}_i \boldsymbol{j}_i^T \tag{3.14}$$

To get the product between the Gauss-Newton matrix and a vector

$$\boldsymbol{s} = \operatorname{vec}(S) = \begin{bmatrix} \operatorname{vec}(S_u) \\ \\ \operatorname{vec}(S_v) \end{bmatrix},$$

from (3.14), (3.9) and (3.1), we first calculate

$$\begin{aligned}
z_i &= \boldsymbol{j}_i^T \operatorname{vec}(S) \\
&= \begin{bmatrix} \boldsymbol{w}_i^T \otimes \boldsymbol{q}_i^T & \boldsymbol{h}_i^T \otimes \boldsymbol{p}_i^T \end{bmatrix} \begin{bmatrix} \operatorname{vec}(S_u) \\ \\ \operatorname{vec}(S_v) \end{bmatrix} \\
&= \boldsymbol{q}_i^T S_u \boldsymbol{w}_i + \boldsymbol{p}_i^T S_v \boldsymbol{h}_i,\ i = 1, \ldots, l,
\end{aligned} \tag{3.15}$$

or equivalently

$$\boldsymbol{z} = \Big(Q^T \odot (W S_u^T) + P^T \odot (H S_v^T) \Big) \mathbf{1}_{d \times 1}, \tag{3.16}$$

where $\odot$ is the Hadamard product (i.e., element-wise product) of two matrices. Next, from (3.15), (3.14), (3.9), (2.4) and (3.1),

$$
\begin{aligned}
G\boldsymbol{s} &= \lambda \operatorname{vec}(S) + \sum_{i=1}^{l} z_i \boldsymbol{j}_i \\
&= \lambda \operatorname{vec}(S) + \sum_{i=1}^{l} z_i \begin{bmatrix} \boldsymbol{w}_i \otimes \boldsymbol{q}_i \\ \boldsymbol{h}_i \otimes \boldsymbol{p}_i \end{bmatrix} \\
&= \lambda \operatorname{vec}(S) + \sum_{i=1}^{l} \begin{bmatrix} \operatorname{vec}\left(z_i \boldsymbol{q}_i \boldsymbol{w}_i^T\right) \\ \operatorname{vec}\left(z_i \boldsymbol{p}_i \boldsymbol{h}_i^T\right) \end{bmatrix} \\
&= \lambda \operatorname{vec}(S) + \begin{bmatrix} \operatorname{vec}\Big(Q\big(\operatorname{diag}(\boldsymbol{z})W\big)\Big) \\ \operatorname{vec}\Big(P\big(\operatorname{diag}(\boldsymbol{z})H\big)\Big) \end{bmatrix}. \qquad (3.17)
\end{aligned}
$$

We investigate the complexity of the Hessian-vector product. The first term of (3.17) is a standard vector scaling that costs $\mathcal{O}\left((M+N) \times d\right)$. For the second term in (3.17), we separately consider two major steps:

1. The computation of WS_u^T and HS_v^T: $\mathcal{O}\left(d \times (\text{nnz}(W) + \text{nnz}(H))\right)$
2. The element-wise product between P^T and HS_v^T: $\mathcal{O}\left(d \times l\right)$
3. The product between dense vector Q and $\operatorname{diag}\left(\boldsymbol{z}\right)W$: $\mathcal{O}\left(d \times l\right)$

Therefore, the cost of one Hessian-vector is $\mathcal{O}\left(d \times (\text{nnz}(W) + \text{nnz}(H))\right)$.

3.3 Implementation of Line Search

Recalculating the function value at each $U + \theta S_u$ and $V + \theta S_v$ is expensive, where the main cost is on calculating $((U + \theta S_u)\boldsymbol{w}_i)^T((V + \theta S_v)\boldsymbol{h}_i), \forall i$. However, the following trick in Yuan et al., (2010) can be employed. Assume that

$$\tilde{y}_i = (U\boldsymbol{w}_i)^T(V\boldsymbol{h}_i), \qquad (3.18)$$

$$\Delta_i = (S_u \boldsymbol{w}_i)^T (V \boldsymbol{h}_i) + (U \boldsymbol{w}_i)^T (S_v \boldsymbol{h}_i) \tag{3.19}$$

and

$$\Delta_i' = (S_u \boldsymbol{w}_i)^T (S_v \boldsymbol{h}_i),\ i = 1, \ldots, l \tag{3.20}$$

are available. At an arbitrary θ, we can calculate

$$((U + \theta S_u)\boldsymbol{w}_i)^T ((V + \theta S_v)\boldsymbol{h}_i) = \tilde{y}_i + \theta \Delta_i + \theta^2 \Delta_i' \tag{3.21}$$

to get the new output value. Now we have

$$\begin{aligned} f(U + \theta S_u, V + \theta S_v) - f(U, V) &= \frac{\lambda}{2} \left(\|U + S_u\|_F^2 + \|V + S_v\|_F^2 - \|U\|_F^2 - \|V\|_F^2 \right) \\ &\quad + \frac{1}{2} \sum_{i=1}^{l} (y_i - \tilde{y}_i - \theta \Delta_i - \theta^2 \Delta_i')^2 - \frac{1}{2} \sum_{i=1}^{l} (y_i - \tilde{y}_i)^2 \\ &= \frac{\lambda}{2} \left(2\theta \langle U, S_u \rangle + 2\theta \langle V, S_v \rangle + \theta^2 \|S\|_F^2 \right) \\ &\quad + \frac{1}{2} \sum_{i=1}^{l} (y_i - \tilde{y}_i - \theta \Delta_i - \theta^2 \Delta_i')^2 - \frac{1}{2} \sum_{i=1}^{l} (y_i - \tilde{y}_i)^2, \end{aligned}$$

where $\langle \cdot, \cdot \rangle$ is the inner product of two matrices. If we further maintain

$$\langle U, S_u \rangle, \langle V, S_v \rangle, \|S\|_F^2, \text{ and } \boldsymbol{g}^T \boldsymbol{s}, \tag{3.22}$$

then the total cost of checking the condition in (2.6) is merely $\mathcal{O}(l)$.

3.4 Overall Procedure

The overall procedure is in Algorithm 1. For the sack of efficiency, if possible, for each operation, we use a matrix-based form rather than a vector-based representation. We discuss some important ones.

For the gradient, from the vector form in (3.12), we rewrite it as a matrix in Line 7. For the stopping condition, we follow past works on Newton methods such as Lin et al., (2008) to use

$$\|\nabla f(\text{vec}(U), \text{vec}(V))\| \leq \epsilon \|\nabla f(\text{vec}(U_{\text{init}}), \text{vec}(V_{\text{init}})\|, \tag{3.23}$$

where $0 < \epsilon < 1$ is a small stopping tolerance and $(U_{\text{init}}, V_{\text{init}})$ indicates the initial point of the model. By the matrix form in Line 7, this stopping condition is essentially

$$\|\nabla f(U, V)\|_F \leq \epsilon \|\nabla f(U_{\text{init}}, V_{\text{init}})\|_F, \tag{3.24}$$

where $\|\cdot\|_F$ is Frobenius norm of a matrix. For Δ_i, Δ_i' in (3.21), for line search, we aggregate all values together to have the form in Line 16. Besides, other values in (3.22) to be maintained are in Line 17. The check of the sufficient decrease condition (2.6) is by the calculation in Line 19.

For details of the CG procedure to approximately solve the linear system (2.3), we provide Algorithm 2. The main cost in each iteration of CG is the product between the Gauss-Newton matrix and a vector in Line 4 and 5, where we follow details in (3.16) and (3.17). Its computational complexity is $\mathcal{O}\left(d \times (\text{nnz}(W) + \text{nnz}(H)\right)$ from what we discussed in (3.17).

After $\boldsymbol{s}_k$ is obtained from the conjugate gradient method, a backtracking line search is applied to adjust the step size α. From the discussion in Section 3.3, the complexity of the line search is

$$\mathcal{O}\left((\text{\# of line search steps}) \times l\right).$$

The overall complexity per Newton iteration is

$$(\text{\# of CG iterations} + 2) \times \mathcal{O}\left(d \times (\text{nnz}(W) + \text{nnz}(H))\right) + (\text{\# of line search steps}) \times \mathcal{O}\left(l\right), \tag{3.25}$$

where the term $2 \times \mathcal{O}\left(d \times (\text{nnz}(W) + \text{nnz}(H))\right)$ is from the cost in Line 7 and Line 16 in Algorithm 1 to compute (3.12), (3.19) and (3.20). If matrix factorization is considered, (1.4) and (1.5) imply that

$$\text{nnz}(W) = \text{nnz}(H) = l. \tag{3.26}$$

Thus (3.25) can be written as

$$(\text{\# of CG iterations} + 2) \times \mathcal{O}\left(ld\right) + (\text{\# of line search steps}) \times \mathcal{O}\left(l\right).$$

Next we discuss the memory usage. The main bottleneck occurs when conducting Line 4 of Algorithm 2 or Line 16 of Algorithm 1. Take Line 4 of Algorithm 2 as an example. We must store the following matrix

$$P \in \mathcal{R}^{d\times l}, Q \in \mathcal{R}^{d\times l}, WD_u^T \in \mathcal{R}^{l\times d}, HD_v^T \in \mathcal{R}^{l\times d}, W \in \mathcal{R}^{l\times M}, \text{ and } H \in \mathcal{R}^{l\times N}. \quad (3.27)$$

Thus the total memory consumption is

$$4 \times \mathcal{O}(ld) + \text{nnz}(W) + \text{nnz}(H).$$

For the case of matrix factorization, from (3.26), nnz(W) and nnz(H) are smaller, so the four $l \times d$ matrices are the bottleneck.

3.5 Implementation for Matrix Factorization

From (1.4), (1.5) and (3.7), for matrix factorization W and H are special matrices when each row contains exactly one non-zero entry. We explore how to take such properties to reduce the memory consumption.

Consider the two main operations (3.16) and (3.17) for the Gauss-Newton matrix-vector product. From

$$P = UW^T \text{ and } Q = VH^T,$$

(3.16) is the same as

$$\boldsymbol{z} = \left((HV^T) \odot (WS_u^T) + (WU^T) \odot (HS_v^T)\right)\mathbf{1}_{d\times 1}.$$

Let $\boldsymbol{z}$ be indexed by (m, n) in R with $R_{m,n} \neq 0$, and

$$S_u = [\boldsymbol{s_u}^1, \ldots, \boldsymbol{s_u}^M] \in \mathcal{R}^{d\times M},$$
$$S_v = [\boldsymbol{s_v}^1, \ldots, \boldsymbol{s_v}^N] \in \mathcal{R}^{d\times N}.$$

Then each component in (3.16) is

$$\boldsymbol{z}_{(m,n)} = \boldsymbol{v}_n^T \boldsymbol{S_u}^m + \boldsymbol{u}_m^T \boldsymbol{S_v}^n.$$

Let $Z \in \mathcal{R}^{M \times N}$ be a sparse matrix to include these $\boldsymbol{z}_{(m,n)}$ elements.

We note that

$$H^T W = I\left[R^T\right],$$

where $I[\cdot]$ is an indicator function with values 1 or 0 to reflect non-zero position of the input matrix. Thus in the calculation of (3.17) we have

$$\begin{aligned} & Q\,\mathrm{diag}(\boldsymbol{z})W \\ =& V H^T\,\mathrm{diag}(\boldsymbol{z})W \\ =& V\left(Z^T \odot I\left[R^T\right]\right). \end{aligned}$$

Further

$$\begin{aligned} & P\,\mathrm{diag}(\boldsymbol{z})H \\ =& U W^T\,\mathrm{diag}(\boldsymbol{z})H \\ =& U\left(Z \odot I\left[R\right]\right). \end{aligned}$$

By the above settings, we never need to store any matrix in (3.27). Therefore the memory consumption is significantly reduced to

$$\mathcal{O}\left(\mathrm{nnz}(R) + (M+N)d\right).$$

For the computational complexity, it remains the same as the one shown in (3.25). The reason is that

$$\mathrm{nnz}(W) = \mathrm{nnz}(H) = \mathrm{nnz}(R).$$

We give details of other places in Algorithm 2 that also need to be changed. Let $B \in \mathcal{R}^{M \times N}$ contains elements $B_{m,n} = \boldsymbol{u}_m^T \boldsymbol{v}_n - r_{m,n}$. At Line 7 in Algorithm 1,we have

$$G \leftarrow \lambda \begin{bmatrix} U & V \end{bmatrix} + \begin{bmatrix} V\left(B^T \odot I\left[R^T\right]\right) & U\left(B \odot I\left[R\right]\right) \end{bmatrix}.$$

Then let $Z' \in \mathcal{R}^{M \times N}$ contains elements $Z'_{m,n} = (\boldsymbol{S_u}^m)^T \boldsymbol{S_v}^n$. At Line 16 in Algorithm 1, we have

$$\boldsymbol{\Delta} = Z,$$

$$\boldsymbol{\Delta}' = Z'.$$

Algorithm 1 Newton method to solve (1.2) by matrix-based operations.

Given $0 < \epsilon < 1$ and the initial (U, V).
Compute and cache $\tilde{\boldsymbol{y}} = ((VH^T) \odot (UW^T))^T \mathbf{1}_{d\times 1}$.
$\boldsymbol{b} \leftarrow \tilde{\boldsymbol{y}} - \boldsymbol{y}$
$f \leftarrow \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2) + \frac{1}{2}\boldsymbol{b}^T\boldsymbol{b}$.
for $k \leftarrow \{0, 1, \dots\}$ **do**
    $Q \leftarrow VH^T$, $P \leftarrow UW^T$
    Compute $G \leftarrow \lambda \begin{bmatrix} U & V \end{bmatrix} + \begin{bmatrix} Q\big(\mathrm{diag}(\boldsymbol{b})W\big) & P\big(\mathrm{diag}(\boldsymbol{b})H\big) \end{bmatrix}$.
    if $k = 0$ **then**
        $\|G^0\|_F \leftarrow \|G\|_F$
    end if
    if $\|G\|_F \leq \epsilon\|G^0\|_F$ **then**
        Output $\begin{bmatrix} U & V \end{bmatrix}$ as the solution of (1.2).
        break
    end if
    Run CG in Algorithm 5 to get an update direction $S = \begin{bmatrix} S_u & S_v \end{bmatrix}$.
    Calculate

$$\boldsymbol{\Delta} = \left(Q^T \odot (WS_u^T) + P^T \odot (HS_v^T)\right)\mathbf{1}_{d\times 1},$$
$$\boldsymbol{\Delta}' = \left((WS_u^T) \odot (HS_v^T)\right)\mathbf{1}_{d\times 1}.$$

    Parpare the following values

$$\langle U, S_u\rangle, \langle V, S_v\rangle, \|S\|_F^2, \text{ and } \langle G, S\rangle.$$

    for $\theta \leftarrow \{1, \beta, \beta^2, \dots\}$ **do**
        $\delta \leftarrow \frac{\lambda}{2}\left(2\theta\langle U, S_u\rangle + 2\theta\langle V, S_v\rangle + \theta^2\|S\|_F^2\right) + \frac{1}{2}\|\boldsymbol{b} - \theta\boldsymbol{\Delta} - \theta^2\boldsymbol{\Delta}'\|^2 - \frac{1}{2}\boldsymbol{b}^T\boldsymbol{b}$
        if $\delta \leq \theta\nu\langle G, S\rangle$ **then**
            $U \leftarrow U + \theta S_u$, $V \leftarrow V + \theta S_v$
            $f \leftarrow f + \delta$
            $\tilde{\boldsymbol{y}} \leftarrow \tilde{\boldsymbol{y}} + \theta\boldsymbol{\Delta} + \theta^2\boldsymbol{\Delta}'$
            $\boldsymbol{b} \leftarrow \tilde{\boldsymbol{y}} - \boldsymbol{y}$
            break
        end if
    end for
end for

Algorithm 2 A conjugate gradient method for solving (2.3) by matrix-based operations.

Given $0 < \eta < 1$ and G, the gradient matrix form of (3.12). Let $S = \mathbf{0}_{d\times(M+N)}$.
Calculate $R = -G$, $D = R$, and $\gamma^0 = \gamma = \|R\|_F^2$.
while $\sqrt{\gamma} > \eta\sqrt{\gamma^0}$ **do**
    $\boldsymbol{z} \leftarrow \left(Q^T \odot (WD_u^T) + P^T \odot (HD_v^T)\right)\mathbf{1}_{d\times 1}$
    $D_h \leftarrow \left(\lambda D + \left[Q\big(\mathrm{diag}(\boldsymbol{z})W\big)\, P\big(\mathrm{diag}(\boldsymbol{z})H\big)\right]\right)$
    $\alpha \leftarrow \gamma / \langle D, D_h\rangle$
    $S \leftarrow S + \alpha D$
    $R \leftarrow R - \alpha D_h$
    $\gamma^{\text{new}} \leftarrow \|R\|_F^2$
    $\beta \leftarrow \gamma^{\text{new}}/\gamma$
    $D \leftarrow R + \beta D$
    $\gamma \leftarrow \gamma^{\text{new}}$
end while
Output S as the solution.

CHAPTER IV

A Review of Alternating Newton Methods

In Chin et al., (2018), they solve the optimization problem (1.2) by a block coordinate descent method. For the two blocks U and V, each time one block is fixed while the other is updated. A simple illustration of the procedure is in Algorithm 3.

For each sub-problem to update one block, Chin et al., (2018) apply a truncated Newton method. Note that because (1.3) is multi-block convex function, each sub-problem is convex. We consider the sub-problem of U as an example to show details of the Newton method. Instead of copying results from Chin et al., (2018), here we show that materials can be extracted from the more sophisticated algorithm in Section III for minimizing both blocks together. To begin, from (3.8)-(3.12), we have

$$\nabla_U f(U,V) = \lambda U + Q\big(\mathrm{diag}(\boldsymbol{b})W\big). \tag{4.1}$$

For the Hessian matrix, from (3.13),

$$\frac{\partial}{\partial \,\mathrm{vec}(U)^T}\frac{\partial f}{\partial \,\mathrm{vec}(U)} = \lambda I + \sum_{i=1}^{l}(\boldsymbol{w}_i \otimes \boldsymbol{q}_i)(\boldsymbol{w}_i \otimes \boldsymbol{q}_i)^T + \sum_{i=1}^{l} b_i \frac{\partial(\boldsymbol{w}_i \otimes \boldsymbol{q}_i)}{\partial \,\mathrm{vec}(U)^T} \tag{4.2}$$

$$= \lambda I + \sum_{i=1}^{l}(\boldsymbol{w}_i \otimes \boldsymbol{q}_i)(\boldsymbol{w}_i \otimes \boldsymbol{q}_i)^T. \tag{4.3}$$

Note that the last term in (4.2) is zero because $\boldsymbol{w}_i \otimes \boldsymbol{q}_i$ is not a function of U. Therefore, the Hessian matrix is positive definite. Further, it is a constant matrix independent of U. The

Algorithm 3 Solving problem (1.2) via alternating minimization.

1: Given an initial solution (U, V)
2: **while** stopping condition is not satisfied **do**
3: $\quad U \leftarrow \arg\min_U f(U, V)$
4: $\quad V \leftarrow \arg\min_V f(U, V)$
5: **end while**

reason is that when V is fixed, the square loss function considered in (1.3) leads to a convex quadratic function of U.

The sub-problem of U becomes equivalent to solving the following linear system by the conjugate gradient method.

$$\left(\frac{\partial}{\partial \operatorname{vec}(U)^T}\frac{\partial f}{\partial \operatorname{vec}(U)}\right)\boldsymbol{s} = -\frac{\partial f}{\partial \operatorname{vec}(U)}. \tag{4.4}$$

To get the product between the Hessian matrix and a vector

$$\boldsymbol{s} = \operatorname{vec}(S_u), \tag{4.5}$$

from (4.3), (3.1), and (3.15), we first calculate

$$\begin{aligned} z_i &= \left[\boldsymbol{w}_i^T \otimes \boldsymbol{q}_i^T\right] \operatorname{vec}(S_u) \\ &= \boldsymbol{q}_i^T S_u \boldsymbol{w}_i,\ i = 1, \ldots, l, \end{aligned} \tag{4.6}$$

or equivalently

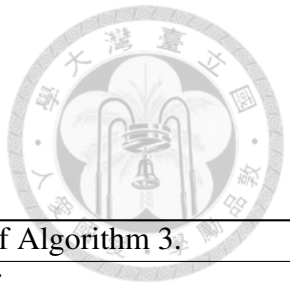
$$\boldsymbol{z} = \left(Q^T \odot (W S_u^T)\right)\mathbf{1}_{d\times 1}. \tag{4.7}$$

Let H be the Hessian in (4.3). From (4.6), (4.3), (4.5) and (3.1),

$$\begin{aligned} H\boldsymbol{s} &= \lambda \operatorname{vec}(S_u) + \sum_{i=1}^{l} z_i(\boldsymbol{w}_i \otimes \boldsymbol{q}_i) \\ &= \lambda \operatorname{vec}(S_u) + \sum_{i=1}^{l} \operatorname{vec}\left(z_i \boldsymbol{q}_i \boldsymbol{w}_i^T\right) \\ &= \lambda \operatorname{vec}(S_u) + \operatorname{vec}\Big(Q\big(\operatorname{diag}(\boldsymbol{z})W\big)\Big). \end{aligned} \tag{4.8}$$

The computational complexity is

$$(\#\text{ of CG iterations} + 2) \times \mathcal{O}\left(d \times \operatorname{nnz}(W)\right).$$



Algorithm 4 Newton method for solving a sub-problem in step 3 of Algorithm 3.

Given $0 < \epsilon < 1$, $0 < \eta < 1$, $\boldsymbol{y}$, W, H, and the current U, V, f.
$Q \leftarrow VH^T$
Compute and cache $\tilde{\boldsymbol{y}} = (Q \odot UW^T)^T \mathbf{1}_{d\times 1}$.
$\boldsymbol{b} \leftarrow \tilde{\boldsymbol{y}} - \boldsymbol{y}$
for $k \leftarrow \{0, 1, \dots\}$ **do**
    Compute $G \leftarrow \lambda U + Q\big(\mathrm{diag}(\boldsymbol{b})W\big)$
    if $k = 0$ **then**
        $\|G^0\|_F \leftarrow \|G\|_F$
    end if
    if $\|G\|_F \leq \epsilon \|G^0\|_F$ **then**
        Output U as the solution of step 3 in Algorithm 3.
        break
    end if
    Let $S_u = \mathbf{0}_{d\times M}$.
    Calculate $R = -G$, $D = R$, and $\gamma^0 = \gamma = \|R\|_F^2$.
    while $\sqrt{\gamma} > \eta\sqrt{\gamma^0}$ **do**
        $\boldsymbol{z} \leftarrow \big(Q^T \odot (WD^T)\big)\,\mathbf{1}_{d\times 1}$
        $D_h \leftarrow \big(\lambda D + Q\big(\mathrm{diag}(\boldsymbol{z})W\big)\big)$
        $\alpha \leftarrow \gamma / \langle D, D_h \rangle$
        $S_u \leftarrow S_u + \alpha D$
        $R \leftarrow R - \alpha D_h$
        $\gamma^{\text{new}} \leftarrow \|R\|_F^2$
        $\beta \leftarrow \gamma^{\text{new}} / \gamma$
        $D \leftarrow R + \beta D$
        $\gamma \leftarrow \gamma^{\text{new}}$
    end while
    Calculate $\boldsymbol{\Delta} = \big(Q^T \odot (WS_u^T)\big)\,\mathbf{1}_{d\times 1}$
    $\delta \leftarrow \frac{\lambda}{2}\left(2\langle U, S_u\rangle + \|S_u\|_F^2\right) + \frac{1}{2}\|\boldsymbol{b} - \theta\Delta\|^2 - \frac{1}{2}\boldsymbol{b}^T\boldsymbol{b}$
    $U \leftarrow U + S_u$
    $f \leftarrow f + \delta$
    $\tilde{\boldsymbol{y}} \leftarrow \tilde{\boldsymbol{y}} + \theta\boldsymbol{\Delta}$
    $\boldsymbol{b} \leftarrow \tilde{\boldsymbol{y}} - \boldsymbol{y}$
end for

The CG procedure stops if the CG iterate $\boldsymbol{s}$ satisfies

$$\| H\boldsymbol{s} + \frac{\partial f}{\partial \operatorname{vec}(U)} \| \leq \eta \| \frac{\nabla f}{\partial \operatorname{vec}(U)} \|. \tag{4.9}$$

Note that because the sub-problem is equivalent to a linear system (4.4), one single CG procedure is enough to obtain an approximate solution. Thus there is no need to have an outer Newton procedure. The CG procedure is presented in Algorithm 4. Clearly, these procedures are simplified from algorithms considering U and V together. Note that in Algorithm 4, we do not need to form $P \leftarrow UW^T$ as in Algorithm 1 because UW^T is used only once.

To solve the sub-problem over V, we apply the same procedure through the following values which are swapped.

$$U \leftrightarrow V$$

$$S_u \leftrightarrow S_v$$

$$\boldsymbol{w}_i \leftrightarrow \boldsymbol{h}_i$$

$$\boldsymbol{q}_i \leftrightarrow \boldsymbol{p}_i$$

$$W \leftrightarrow H$$

$$Q \leftrightarrow P$$

Regarding the memory cost, the bottleneck is at similar places to those in Algorithm 1. However, the needed memory is only about half of Algorithm 1. This can be clearly seen from comparing Line 4 of Algorithm 2 and Line 17 of Algorithm 4.

CHAPTER V

Experiments

We conduct experiments to compare the following two methods to solve (1.2).

- Newton method described in Section II.
- Alternating Newton method (Chin et al., 2018) described in Section III.

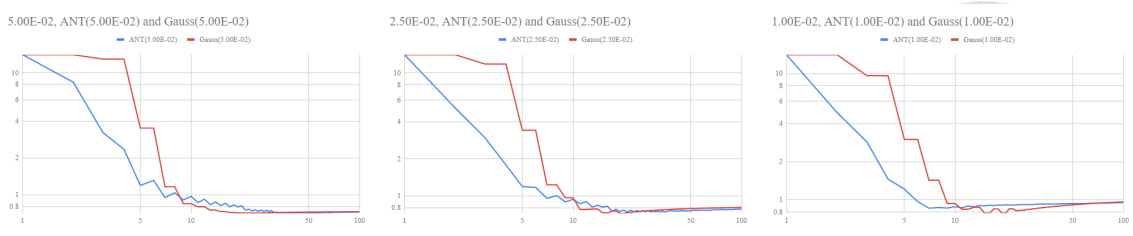
5.1 Experimental Settings

Table 5.1: Statistics and parameters for each data set.

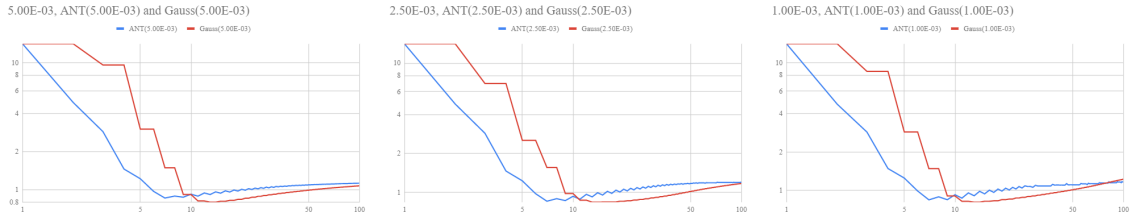
Data Set	movielens10m	netflix	movielens1m
M	71,567	2,649,429	6,040
N	65,133	17,770	3,952
#Training	9,301,274	99,072,112	*
#Test	698,780	1,408,395	*
d	40	20	40

We consider data sets listed in Table 5.1. For both methods, every element of initial U and V is randomly drawn from the interval $[-0.1/\sqrt{d}, 0.1/\sqrt{d}]$.

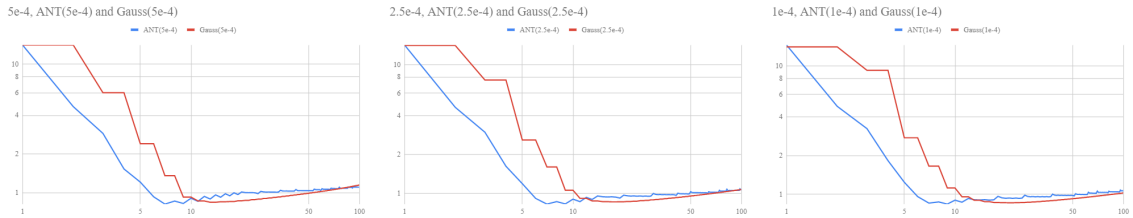
For the CG procedure, we set $\eta = 0.3$ as the stopping tolerance. Note that we consider the same η for the CG procedure in both Algorithms 2 and **??**. For the line search procedure, we set $\beta = 0.5$ as the ratio to decrease the step size and $\nu = 0.1$ for the sufficient decrease condition (2.6). For the latent dimension, we use d listed in Table 5.1.



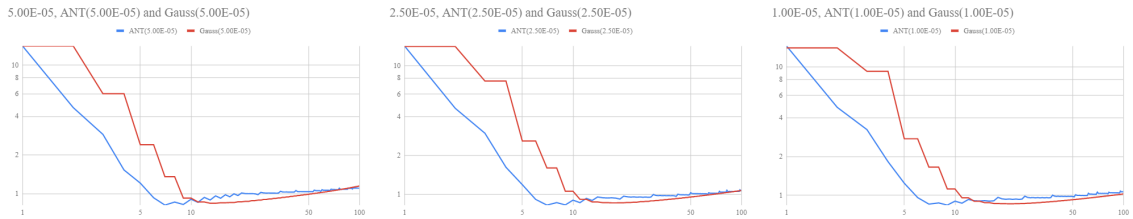
(a) ml-1m.5e-2 (b) ml-1m.2.5e-2 (c) ml-1m.1e-2



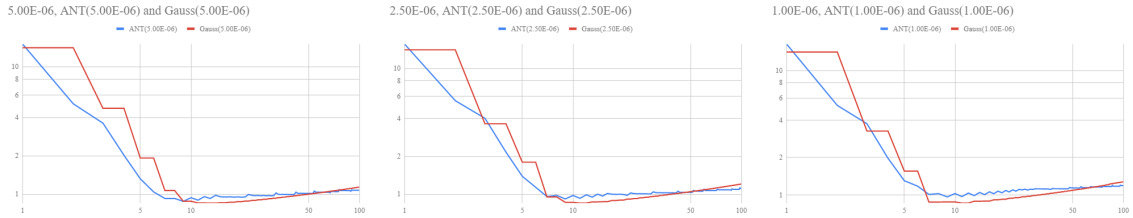
(d) ml-1m.5e-3 (e) ml-1m.2.5e-3 (f) ml-1m.1e-3



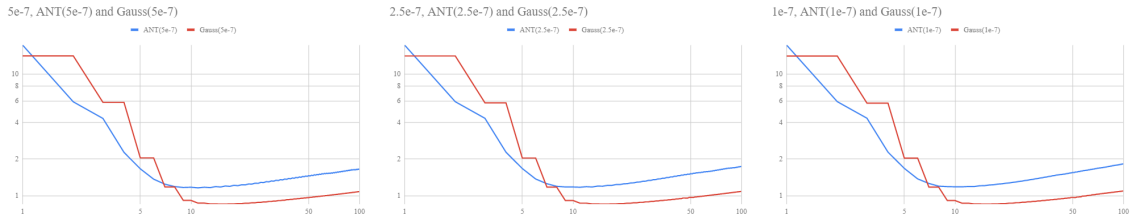
(g) ml-1m.5e-4 (h) ml-1m.2.5e-4 (i) ml-1m.1e-4



(j) ml-1m.5e-5 (k) ml-1m.2.5e-5 (l) ml-1m.1e-5



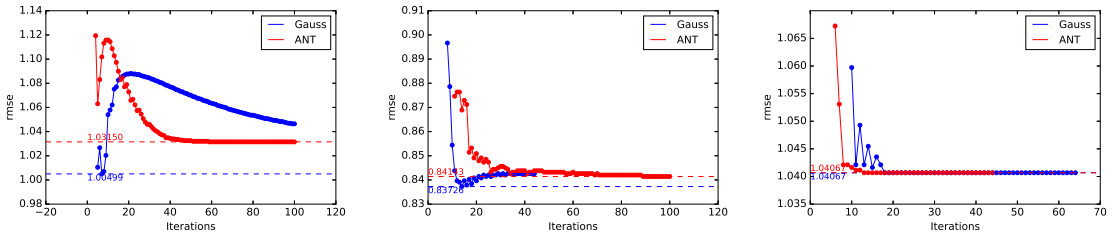
(m) ml-1m.5e-6 (n) ml-1m.2.5e-6 (o) ml-1m.1e-6



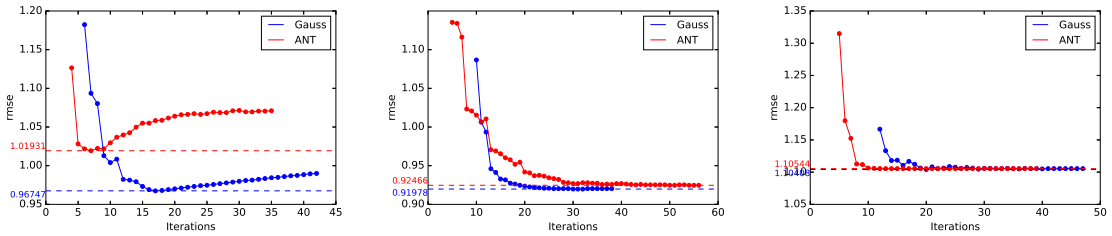
(p) ml-1m.5e-7 (q) ml-1m.2.5e-7 (r) ml-1m.1e-7

Figure 5.1: A comparison on the va-loss of different methods. ANT is worse than Gauss among smaller λ. The x-axis is the iteration, while the y-axis is the va-loss.

(a) ml-1m.5e-3 (b) ml-1m.5e-2 (c) ml-1m.5e-1

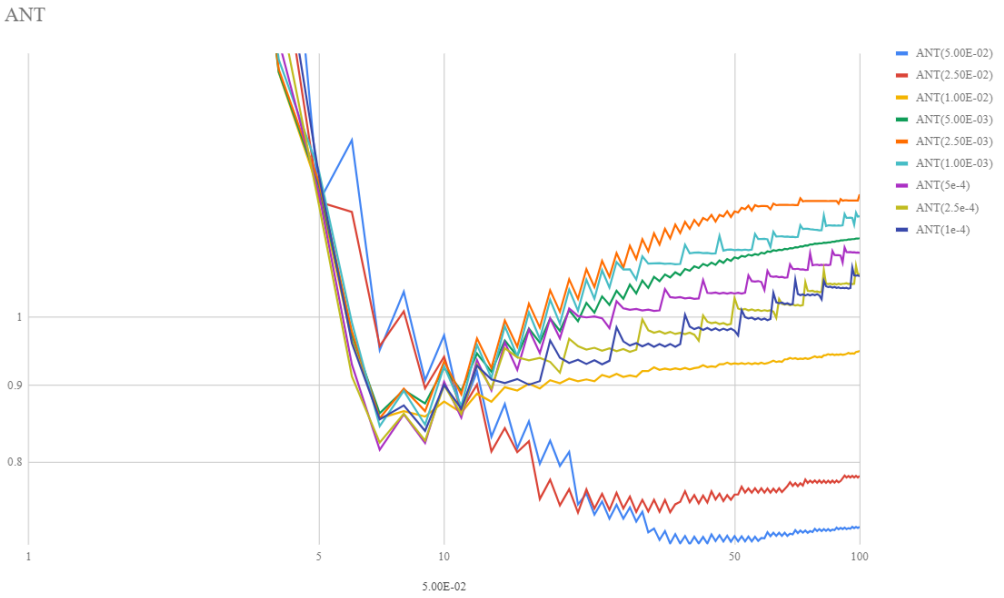


(d) ml-10m.5e-3 (e) ml-10m.5e-2 (f) ml-10m.5e-1

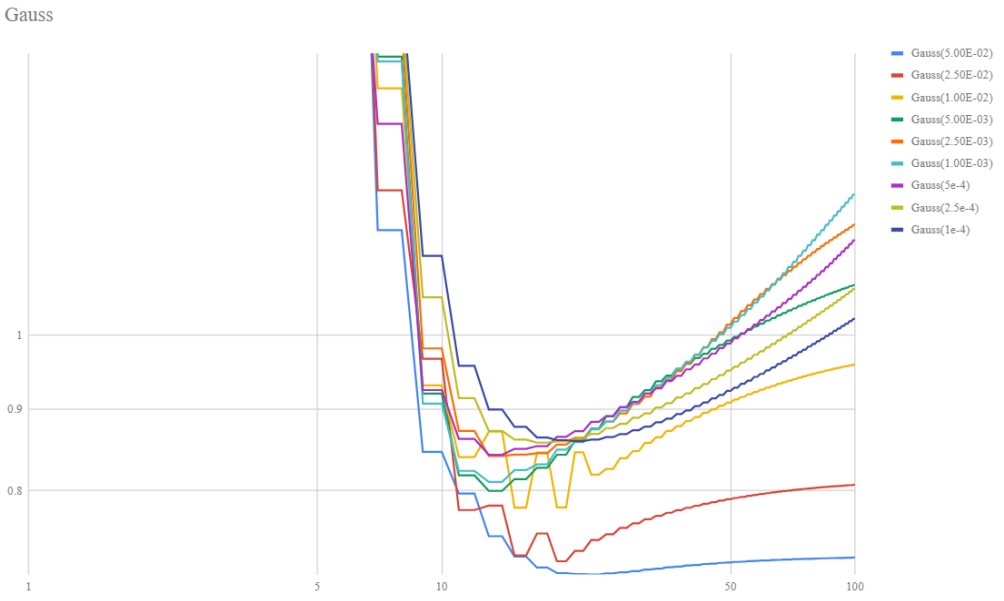


(g) nf.5e-3 (h) nf.5e-2 (i) nf.5e-1

Figure 5.2: A comparison on the va-loss of different methods. The x-axis is the iteration, while the y-axis is the va-loss.



(a) Ant



(b) Gauss

Figure 5.3: ml-1m. A comparison on the va-loss of different λ. The x-axis is the iteration, while the y-axis is the va-loss.

Figure 5.4: ml-1m. A comparison on the best va-loss of different methods. The x-axis is λ, while the y-axis is the va-loss.

Bibliography

Rendle, Steffen (2010). "Factorization machines". In: *Proceedings of IEEE International Conference on Data Mining (ICDM)*, pp. 995–1000.

Blondel, Mathieu et al. (2016). "Polynomial Networks and Factorization Machines: new Insights and Efficient Training Algorithms". In: *Proceedings of the 33rd International Conference on Machine Learning (ICML)*.

Chin, Wei-Sheng et al. (2018). "An Efficient Alternating Newton Method for Learning Factorization Machines". In: *ACM Transactions on Intelligent Systems and Technology* 9.6, 72:1–72:31. URL: https://www.csie.ntu.edu.tw/~cjlin/papers/fm/scalefm.pdf.

Yuan, Guo-Xun et al. (2010). "A Comparison of Optimization Methods and software for Large-scale L1-regularized Linear Classification". In: *Journal of Machine Learning Research* 11, pp. 3183–3234. URL: http://www.csie.ntu.edu.tw/~cjlin/papers/l1.pdf.

Lin, Chih-Jen, Ruby C. Weng, and S. Sathiya Keerthi (2008). "Trust region Newton method for large-scale logistic regression". In: *Journal of Machine Learning Research* 9, pp. 627–650. URL: http://www.csie.ntu.edu.tw/~cjlin/papers/logistic.pdf.

We consider a preconditioner $\tilde{M}$ to approximately factorize H such that $H \approx \tilde{M}\tilde{M}^T$, and then use CG to solve

$$\hat{H}\hat{\boldsymbol{s}} = \hat{\boldsymbol{g}}, \tag{5.1}$$

where $\hat{H} = \tilde{M}^{-1}H\tilde{M}^{-T}$ and $\hat{\boldsymbol{g}} = \tilde{M}^{-1}\boldsymbol{g}$. Once $\hat{\boldsymbol{s}}$ is found, the solution of (2.2) can be recovered by $\boldsymbol{s} = \tilde{M}^{-T}\hat{\boldsymbol{s}}$. Then, we consider the following diagonal preconditioner for solving (2.2) as an example.

$$\tilde{M} = \tilde{M}^T = \mathrm{diag}(\boldsymbol{h}), \tag{5.2}$$

where

$$\boldsymbol{h} = \sqrt{\lambda \mathbf{1}_{d(M+N)\times 1} + \sum_{i=1}^{l} \boldsymbol{j}_i \odot \boldsymbol{j}_i}.$$

Note that "$\sqrt{\cdot}$" element-wisely performs the square-root operation if the input argument is a vector or a matrix. From

$$\boldsymbol{j}_i = \begin{bmatrix} \boldsymbol{w}_i \otimes \boldsymbol{q}_i \\ \boldsymbol{h}_i \otimes \boldsymbol{p}_i \end{bmatrix} = \begin{bmatrix} \mathrm{vec}(\boldsymbol{q}_i \boldsymbol{w}_i^T) \\ \mathrm{vec}(\boldsymbol{p}_i \boldsymbol{h}_i^T) \end{bmatrix}, \tag{5.3}$$

we have

$$\begin{aligned} \sum_{i=1}^{l} \boldsymbol{j}_i \odot \boldsymbol{j}_i &= \sum_{i=1}^{l} \begin{bmatrix} \mathrm{vec}\big((\boldsymbol{q}_i \odot \boldsymbol{q}_i)(\boldsymbol{w}_i \odot \boldsymbol{w}_i)^T\big) \\ \mathrm{vec}\big((\boldsymbol{p}_i \odot \boldsymbol{p}_i)(\boldsymbol{h}_i \odot \boldsymbol{h}_i)^T\big) \end{bmatrix} \\ &= \begin{bmatrix} \mathrm{vec}\big((Q \odot Q)(W \odot W)\big) \\ \mathrm{vec}\big((P \odot P)(H \odot H)\big) \end{bmatrix}. \end{aligned} \tag{5.4}$$

Thus, the preconditioner can be obtained via

$$\tilde{M} = \tilde{M}^T = \mathrm{diag}\Big(\sqrt{\lambda \mathbf{1}_{d(M+N)\times 1} + \begin{bmatrix} \mathrm{vec}\big((Q \odot Q)(W \odot W)\big) \\ \mathrm{vec}\big((P \odot P)(H \odot H)\big) \end{bmatrix}}\Big). \tag{5.5}$$

Algorithm 5 A preconditioned conjugate gradient method for solving (2.2) by operations on matrix variables.

Given $0 < \eta < 1$ and G, the gradient matrix form of (1.2). Let $\hat{S} = \mathbf{0}_{d\times(M+N)}$.
Compute M via (5.5).
Calculate $R = -G\,./M$, $\hat{D} = R$, and $\gamma^0 = \gamma = \|R\|_F^2$.
while $\sqrt{\gamma} > \eta\sqrt{\gamma^0}$ **do**
    $\hat{D}_h \leftarrow \hat{D}\,./M$
    $\hat{\boldsymbol{z}} \leftarrow \Big(Q^T \odot (W\hat{D}_{hu}^T) + P^T \odot (H\hat{D}_{hv}^T)\Big)\,\mathbf{1}_{d\times 1}$
    $\hat{D}_h \leftarrow \Big(\lambda\hat{D}_h + \big[Q\big(\mathrm{diag}(\hat{\boldsymbol{z}})W\big)\; P\big(\mathrm{diag}(\hat{\boldsymbol{z}})H\big)\big]\Big)\,./M$
    $\alpha \leftarrow \gamma/\langle\hat{D}, \hat{D}_h\rangle$
    $\hat{S} \leftarrow \hat{S} + \alpha\hat{D}$
    $R \leftarrow R - \alpha\hat{D}_h$
    $\gamma^{\text{new}} \leftarrow \|R\|_F^2$
    $\beta \leftarrow \gamma^{\text{new}}/\gamma$
    $\hat{D} \leftarrow R + \beta\hat{D}$
    $\gamma \leftarrow \gamma^{\text{new}}$
end while
Output $S = \hat{S}\,./M$ as the solution.