SAG-re: Faster Prototyping of Recommender Systems using Stochastic Average Gradient

James Lo
Computer Science
University of British Columbia
Vancouver, Canada
tklo@cs.ubc.ca

ABSTRACT

In the age of agile software engineering and shorter product lifecycles, data-scientists would ultimately face the challenge of running many experiments and producing high-quality results, with less time. In this paper, we motivate the problem of adopting the stochastic average gradient method (SAG) for prototyping model-based recommender systems. We motivate that, by taking advantage of SAG's fast convergence rate and low iteration cost, data-scientists are able to achieve better optimizations for their recommender systems in a shorter amount of time. However, adopting SAG in prototyping model-based recommender systems is not trivial because the asymptotic space-complexity of using SAG can be prohibitively high. We propose SAG-RE as our approach to resolve the space-complexity challenge. SAG-RE preserves all the benefits and advantages of using SAG, and SAG-RE achieves asymptotic space complexity as compact as any memory-less approach. We both prove in theory and extensively evaluate in practice that, SAG-RE yields a better quality optimization within a shorter amount of time, than the two main gradient methods in the state-of-the-art of prototyping recommender systems, namely full deterministic gradient, and stochastic gradient.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information Filtering*

General Terms

Asymptotic Time Complexity, Asymptotic Space Complexity, Prototyping, Experimentation

Keywords

Recommender systems, collaborative filtering, matrix factorization, stochastic gradient, agile software engineering

1. INTRODUCTION

Shopping, text advertising, display advertising, renting movies, listening to music... recommender systems are prevalent and ubiquitous in our daily lives. Matrix factorization (MF) is a popular technique in model-based recommender systems. MF has been utilized extensively in past research for handling both explicit [6, 11, 9] ratings, and implicit [3, 4, 10, 5, 11] feedback.

In recommender systems that utilize matrix factorization, most optimize an objective function. In the state of the art, full deterministic gradient (FG) and stochastic gradient (SG) are the two main gradient methods for optimization. All of the recommender systems that we cite above utilize either full deterministic gradient, or stochastic gradient.

Unfortunately, both full deterministic gradient and stochastic gradient have pitfalls when it comes to prototyping recommender systems. Full deterministic gradient can offer high quality optimizations. However, FG is slow because at each iteration of optimization, FG has to sample through all the entries in the dataset. Stochastic gradient is relatively fast; its iteration cost is low because each iteration of SG sample only one or a few entries. However, the trade-off with stochastic gradient is that it often provides low quality optimizations. By chance, stochastic gradient may *eventually* yield a good quality optimization. If it ever happens, it is after a tremendous number of iterations. Thus stochastic gradient is also slow in terms of yielding a good quality optimization within a reasonable amount of time.

High quality optimizations within a short amount of time is important when building recommender systems. The first reason is that data scientists often have to run repeated experiments: e.g. with different objective functions, different metrics, different datasets, and different optimization parameters. The high level goal to run multiple experiments is that, through experimentation and comparing results of multiple trials, data scientists can ultimately get a sufficiently good mix of objective function and hyper parameters for fitting a dataset. The second reason is that product life cycles are shortening in the age of agile software engineering. Thus data scientists are facing or will ultimately face the challenge of running more experiments and producing high quality results with less time.

In this paper, we study the challenge from the perspective of convex-optimization. We propose and hypothesize using the stochastic average gradient (SAG) method [8, 7] as a viable alternative to using FG and SG during the prototyping process. SAG has the distinctive advantage that its optimization quality is proven to be much better than SG;

at the same time SAG's iteration cost is asymptotically as low as *SG*. However, applying and adapting SAG to matrix factorization is not trivial because SAG requires previously-computed gradients; and storing these gradients can lead to very high asymptotic space complexity. We explore the challenge with space-complexity, and resolve it by proposing a re-computation approach (SAG-RE) that re-computes the previously-computed gradients on-the-fly, on-demand. SAG-RE preserves the fast convergence rate and low iteration cost of SAG. Moreover, the asymptotic space complexity of SAG-RE is as compact as memory-less gradient methods such as *FG* and *SG*.

To the best of our knowledge, we are the first to

- Identify pitfalls associated with using full deterministic gradient and stochastic gradient when data-scientists prototype model-based recommender systems.
- Propose Stochastic Average Gradient (SAG) as a viable alternative for yielding higher quality optimizations while enjoying a low iteration cost.
- Extend SAG into SAG-RE for matrix factorization, resolve the space complexity challenge in adapting SAG from the domain of large-scale supervised-machine-learning into the domain of prototyping recommender algorithms.
- Prove in theory, that SAG-RE has a convergence rate as fast as the original SAG; SAG-RE has asymptotic time complexity as efficient as any gradient method with the lowest iteration cost, and SAG-RE has asymptotic space complexity as compact as any memoryless gradient method.
- Extensively evaluate and compare SAG-RE with *FG* and *SG* across multiple RecSys objective functions and diverse datasets.
- Demonstrate in practice that, even without any optimization or fine-tuning on the implementation, SAG-RE still yields the best optimization within the shortest time despite the additional time of re-computation, and that SAG-RE uses memory at a level similar to full deterministic gradient and stochastic gradient, both of which are memory-less.
- Provide follow-up evidence that both full deterministic gradient and stochastic gradient takes much longer to reach a quality of optimization similar to SAG-RE.

2. BACKGROUND AND TERMINOLOGY

To motivate our paper and the space complexity challenge, we first introduce the background and the terminology that we use.

Matrix Factorization. Model-based recommender systems approximate the *user-item* matrix A through the dot-product of the *user*-matrix U and the *item*-matrix V: $\hat{A} = U * V$.

The *user-item* matrix A is a *nRows*-by-*nCols* matrix. A can be *sparse*; thus we use N to indicate the number of non-zero entries in A.

The *approximation* matrix $\hat{A}$ also has *nRows* rows, and *nCols* columns. $\hat{A}$ is not a sparse matrix. The goal of model-based recommendation is to use the non-zero entries to approximate the missing entries in A. When multiplying U and V, the latent dimensions *nDims* cancels-out in the dot product. This is why the *approximation* matrix has identical dimensions as the original *user-item* matrix.

The *user* matrix U is *nRows*-by-*nDims*: U has *nRows* rows, and *nDims* columns. *nDims* is the number of latent dimensions. The *item* matrix V is *nDims*-by-*nCols*.

Optimizing an Objective Function. The goal of matrix factorization is to find the best U and the best V whose dot product optimizes an objective function:

$$\arg\min_{U,V}(\text{or} \arg\max) \left[f(U,V) = \sum_{i=1}^{nRows} \sum_{j=1}^{nCols} f(\bar{u}_i, \bar{v}_j) \right] \quad (1)$$

When we take the gradient of the objective function with respect to a row in the *user* matrix U (e.g. $\bar{u}_i$), we sum up the gradient of all the entries in $\hat{A}$ that belong to the same row $\bar{u}_i$.

$$\frac{\mathrm{d}f(U,V)}{\mathrm{d}\bar{u}_i} = \sum_{j=1}^{nCols} \frac{\mathrm{d}f(\bar{u}_i, \bar{v}_j)}{\mathrm{d}\bar{u}_i} \quad (2)$$

Similarly, when we take the gradient with respect to a column of V (e.g. $\bar{v}_j$), we sum up the gradients across different rows that belong to the same column:

$$\frac{\mathrm{d}f(U,V)}{\mathrm{d}\bar{v}_j} = \sum_{i=1}^{nRows} \frac{\mathrm{d}f(\bar{u}_i, \bar{v}_j)}{\mathrm{d}\bar{v}_j} \quad (3)$$

Both $\frac{\mathrm{d}f(\bar{u}_i,\bar{v}_j)}{\mathrm{d}\bar{u}_i}$ and $\frac{\mathrm{d}f(\bar{u}_i,\bar{v}_j)}{\mathrm{d}\bar{v}_j}$ are vectors of length *nDims*, the number of latent dimensions. Specifically, $\frac{\mathrm{d}f(\bar{u}_i,\bar{v}_j)}{\mathrm{d}\bar{u}_i}$ is a *1*-by-*nDims* row vector; $\frac{\mathrm{d}f(\bar{u}_i,\bar{v}_j)}{\mathrm{d}\bar{v}_j}$ is a *nDims*-by-*1* column vector. Similarly, the summed-up gradient $\frac{\mathrm{d}f(U,V)}{\mathrm{d}\bar{u}_i}$ is a row vector, and $\frac{\mathrm{d}f(U,V)}{\mathrm{d}\bar{v}_j}$ is a column vector, of length *nDims*.

In SAG, storing only the summed-up gradients is not sufficient for matrix factorization. The reason is that, each iteration of SAG requires the fine-grain gradients of individual entries (e.g. $\frac{\mathrm{d}f(\bar{u}_i,\bar{v}_j)}{\mathrm{d}\bar{u}_i}$ and $\frac{\mathrm{d}f(\bar{u}_i,\bar{v}_j)}{\mathrm{d}\bar{v}_j}$) that we previously sampled at an iteration before t. As we will prove, when directly applied to matrix factorization without using our SAG-RE approach, SAG will have a asymptotic space complexity of $\theta(nDims * (min(M,N) + nRows + nCols))$. M is the number of *distinct* entries that we have previously sampled. At any iteration t,

$$M \propto \sum_{l=1}^{t} B_l \quad (4)$$

B_r is the batch size at ieration l; $l \le t$. Usually the batch size B is constant for all iterations; then M is proportional to and is less than or equal to $B * t$.

Here, we want to point out that SAG-RE preserves the low asymptotic time complexity as SAG; and SAG-RE reduces asymptotic space complexity to $\theta(N + nDims * (nRows + nCols))$. We will prove that this asymptotic space complexity is as compact as any memory-less approach.

Gradient Methods in Matrix Factorization. Gradient methods are iterative methods of optimization. When we increase the number of iterations, we expect the quality of optimization to also increase over time. At each iteration, gradient methods sample a batch of B entries, calculate the gradients of these entries, and use the calculated gradients to update U and V for the next iteration:

$$U^{t+1} = U^t + \frac{\alpha^t}{B} \left(\sum_{b=1}^{B} \frac{\mathrm{d}f(\bar{u}_{entry(b).i}, \bar{v}_{entry(b).j})}{\mathrm{d}\bar{u}_{entry(b).i}} \right) \quad (5)$$

$$V^{t+1} = V^t + \frac{\alpha^t}{B}\left(\sum_{b=1}^{B} \frac{\mathrm{d}f(\bar{u}_{entry(b).i}, \bar{v}_{entry(b).j})}{\mathrm{d}\bar{v}_{entry(b).j}}\right) \quad (6)$$

At iteration t, U^t is the current approximation of U. We use the gradients of the sampled batch of entries to update U^t into U^{t+1} for iteration *t+1*.

α^t is the *learning-rate* or *step-size*, at iteration t. When the goal of our optimization is to maximize an objective function, we apply *gradient-ascent* on U and V; thus we set $\alpha^t > 0$. When we try to minimize an objective function, we apply *gradient-descent* and set $\alpha^t < 0$.

entry(b) is the b-th entry in our batch of samples. *entry(b).i* is the *row* number of the entry; *entry(b).j* is the *column* number of the entry sampled from A.

Full deterministic gradient (FG) takes all N samples at each iteration; $B = N$ in FG. Stochastic gradient (SG) takes only one or a few samples per iteration: B is usually a constant much less than N.

Stochastic Average Gradient. Stochastic Average Gradient (SAG) requires a memory of previously-computed gradients: e.g. $\bar{m}_U^t$ and $\bar{m}_V^t$ for matrix factorization. Each iteration of SAG uses a sampled batch of entries to update the memory. After the update, SAG then applies the updated memory $\bar{m}_U^{t+1}$ and $\bar{m}_V^{t+1}$ respectively on calculating U^{t+1} and V^{t+1}:

$$\bar{m}_{entry(b).i}^{t+1} = \frac{\mathrm{d}f(\bar{u}_{entry(b).i}, \bar{v}_{entry(b).j})}{\mathrm{d}\bar{u}_{entry(b).i}} \quad (7)$$

$$\bar{m}_U^{t+1} = \bar{m}_U^t + \sum_{b=1}^{B}\left[\bar{m}_{entry(b).i}^{t+1} - \bar{m}_{entry(b).i}^{t}\right] \quad (8)$$

$$U^{t+1} = U^t + \frac{\alpha^t}{M}\left(\bar{m}_U^{t+1}\right) \quad (9)$$

$$\bar{m}_{entry(b).j}^{t+1} = \frac{\mathrm{d}f(\bar{u}_{entry(b).i}, \bar{v}_{entry(b).j})}{\mathrm{d}\bar{v}_{entry(b).j}} \quad (10)$$

$$\bar{m}_V^{t+1} = \bar{m}_V^t + \sum_{b=1}^{B}\left[\bar{m}_{entry(b).j}^{t+1} - \bar{m}_{entry(b).j}^{t}\right] \quad (11)$$

$$V^{t+1} = V^t + \frac{\alpha^t}{M}\left(\bar{m}_V^{t+1}\right) \quad (12)$$

$\bar{m}_{entry(b).i}^t$ and $\bar{m}_{entry(b).j}^t$ are the fine-grain gradients of individual matrix entries that were previously sampled.

$\bar{m}_{entry(b).i}^t$ is a 1-by-$nRows$ row vector; $\bar{m}_{entry(b).j}^t$ is a $nCols$-by-1 column vector.

$\bar{m}_U^t$ is a $nRows$-by-$nDims$ matrix, because $\bar{m}_U^t$ aggregates the gradients of all rows in the *user* matrix U. Similarly, $\bar{m}_V^t$ is a $nDims$-by-$nCols$ matrix.

We apply SAG into matrix factorization for two reasons. First, SAG has iteration cost as low as stochastic gradient (SG). Second, SAG's convergence rate is faster than SG, and sometimes as fast as full deterministic gradient (FG).

Convergence rate, Iteration cost, and Prototyping recommender systems. At a high level, the ideal combination of a fast convergence rate and a low iteration cost implies a better optimization in a shorter amount of time when data-scientists prototype model-based recommender systems. An intuition behind gradient methods is that, at least for objective functions that are convex, the gradients guide the updates of U^t and V^t towards the direction of optimization. Convergence rate measures how many iterations a gradient method is expected to take towards reach a quality of optimization similar to FG. Iteration cost measures how many entries we sample per iteration.

Full deterministic gradient has the best possible convergence rate because each iteration of FG samples all N entries in the dataset. However, while FG is guaranteed to take a less number of iterations than SG to reach optimization, sampling all N entries per iteration slows down FG overall because the optimization process would still take many iterations. Depending on the mathematical properties of the objective function, stochastic gradient often has much slower convergence rates than FG because SG samples only one or a few random entries per iteration. Therefore, while SG has the lowest possible $\theta(1)$ iteration cost, overall SG is still slow because SG would take many more iterations to reach optimization.

SAG speeds-up the convergence rate by reusing the gradients of past samples. Reusing past gradients enables SAG to sample $\theta(1)$ entries per iteration and to achieve the lowest possible iteration cost. Our evaluation illustrates that SAG gives a better optimization with less time than FG and SG. In this paper, we minimize the drawbacks or costs of using SAG in matrix factorization while preserving SAG's benefits.

3. CHALLENGE

As equations 8 and 11 illustrate, updating $\bar{m}_U^{t+1}$ and $\bar{m}_V^{t+1}$ requires $\bar{m}_{entry(b).i}^t$ and $\bar{m}_{entry(b).j}^t$. $\bar{m}_{entry(b).i}^t$ and $\bar{m}_{entry(b).j}^t$ are the fine-grained gradients of an individual entry $entry(b)$ from the last time (or the most recent time) that $entry(b)$ was sampled.

When applying SAG into matrix factorization, a major challenge is to make these fine-grain gradients available: $\bar{m}_{entry(b).i}^t$ from equation 8, and $\bar{m}_{entry(b).j}^t$ from equation 11

A naïve approach is to store all these fine-grain graidents. As we shall prove, the naïve approach is undesirable because storing all these gradients would take up a lot of space.

Theorem 1. The total asymptotic space complexity is $\theta(nDims * (min(M, N) + nRows + nCols))$ for the naïve approach of storing the fine-grain gradients of all entries that we had previously sampled.

PROOF. For each individual entry, the amount of space required is $2 * nDims$: the gradient with respect to row $\bar{u}_i$ ($\bar{m}_{entry(b).i}^t$) is a *1*-by-*nDims* row vector; the gradient with respect to column $\bar{v}_j$ ($\bar{m}_{entry(b).j}^t$) is a *nDims*-by-*1* column vector.

When we store the fine-grain gradients of all previously-sampled entries, the amount of space required becomes $M * 2 * nDims$. Recalling from the background section, M is the number of distinct entries that we previously sampled.

As shown in equations 8 and 11, SAG requires only the most recent gradient of each previously-sampled entry. Thus for each entry, we store a max of only one set of gradients ($\bar{m}_{entry(b).i}^t$ and $\bar{m}_{entry(b).j}^t$). The total amount of space required becomes $min(M, N) * 2 * nDims$.

Now, according to equations 8 and 11, we must also store the aggregated gradients: $\bar{m}_U^t$ and $\bar{m}_V^t$. $\bar{m}_U^t$ takes $nRows * nDims$ space; $\bar{m}_V^t$ takes $nDims * nCols$ space. Thus the total amount of space that we use to store the aggregated gradients is $(nRows*nDims)+(nDims*nCols)$, which is equivalent to $nDims*(nRows+nCols)$ after simplification.

Adding the fine-grain gradients and the aggregated gradients together, the asymptotic space complexity becomes $\theta(nDims*(min(M,N)+nRows+nCols))$ after ignoring the constants. $\square$

No guarantee that $min(M,N)$ is small. If we can guarantee that $min(M,N)$ is small, or that $min(M,N)$ is asymptotically not larger than *nRows* or *nCols*, then the effective asymptotic space-complexity becomes $\theta(nDims*(nRows+nCols))$, which is the most compact anyone can possibly get. Unfortunatley, we shall prove that there is no such guarantee.

First, we explore what the best possible asymptotical space-complexity can be in matrix factorization.

Theorem 2. $\Omega(N+nDims*(nRows+nCols))$ is the lower-bound asymptotic space-complexity in matrix factorization.

PROOF. Matrix factorization is to approximate a matrix A (e.g. the *user-item* matrix) through the dot product of two matrices U (e.g. the *user* matrix) and V (e.g. the *item* matrix). A has N non-zero entries. U is a *nRows*-by*nDims* matrix; V is a *nDims*-by-*nCols* matrix. In each iteration of convex optimization, we must update U and V, and use an objective function to compare our approximation to the ground-truth matrix A. Therefore, any matrix factorization algorithm would have an asymptotic space-complexity of at least $\Omega(N+nDims*(nRows+nCols))$. $\square$

If we can guarantee that $min(M,N)$ is asymptotically not larger than *nRows* or *nCols*, then we can prove that the naïve approach has already achieved the best possible asymptotic space-complexity, and that our challenge is irrelevant. However, we shall prove that such guarantee does not exist.

Theorem 3. There is no guarantee that $min(M,N)$ is asymptotically not larger than *nRows* or *nCols*.

PROOF. N is the number of non-zero entries in the matrix A. Unless there is, or unless we are restricted to an upper-bound of matrix density, then N must have an uppder-bound of $O(nRows*nCols)$ space.

M is the number of *distinct* entries that we previously sampled. According to equation 4, M depends on the batch size at each iteration B_l, and the number of iterations previously done $t-1$. Usually, the batch size is a constant B. Thus the lower bound of M most likely depends on the lower bound of t. However, the lower bound of t depends on the convergence rate, and the tolerance of error ϵ. For example, if the convergence rate is exponential (e.g. $O(p^t)$), then the lower bound of t is $\Omega(log(\frac{1}{\epsilon}))$. Therefore, the lower bound of M does not depend on N, *nRows* or *nCols*. Given a dataset, the only way to enforce $M \leq N$ is to either tolerate a high error, or to find a combination of objective function and gradient method that yields the fastest convergence rate possible. The asymptotic space-complexity of SAG-RE is compact enough so that SAG-RE does not enforce data-scientists to tolerate a high error. Given any objective function, the convergence rate of *SAG* [8, 7] is always faster than stochastic gradient and is sometimes as fast as the fastest full deterministic gradient. SAG-RE preserves the convergence rate of *SAG*. $\square$

Using chain rule worsens space-complexity in matrix factorization. In supervised machine-learning, we can use the chain-rule in differential-calculus to reduce space-complexity. Unfortunately, applying the chain-rule in matrix-factorization would result in a space-complexity larger than the naïve approach.

In supervised machine-learing, the goal is to compute the best-fit column-vector $\bar{\omega}$ that optimizes an objective function, which can be written as

$$\arg\min_{\bar{\omega}}(\text{or} \arg\max)\left[F(\hat{y}=X*\bar{\omega})=\sum_{i=1}^{N}f_i(\hat{y}_i=\bar{x}_i*\bar{\omega})\right] \tag{13}$$

X is a N-by-d matrix: N is the number of samples, and d is the number of features. $\bar{x}_i$ is the 1-by-d row vector reprsenting i-th sample. $\bar{\omega}$ is the d-by-1 column vector of features that we are trying to learn from X. We can use the chain-rule and re-write the gradient of $\bar{\omega}$ with respect to f_i:

$$\frac{\mathrm{d}f_i}{\mathrm{d}\bar{\omega}}=\left(\frac{\mathrm{d}f_i}{\mathrm{d}\hat{y}_i}\right)\frac{\mathrm{d}\hat{y}_i}{\mathrm{d}\bar{\omega}}=(\bar{x}_i)'\left(\frac{\mathrm{d}f_i}{\mathrm{d}\hat{y}_i}\right) \tag{14}$$

Originally, using the naïve approach of *SAG* results in $\theta(min(M,N)*d+d)$ space. The reason is that $\frac{\mathrm{d}f_i}{\mathrm{d}\bar{\omega}}$ is a d-by-1 column vector; and the naïve approach stores $min(M,N)$ copies of them. The memory gradient $\bar{m}_{\bar{\omega}}$ is a d-by-1 column vector and thus takes $\theta(d)$ space.

The dot-product $\hat{y}_i=(\bar{x}_i*\bar{\omega})$ is a 1-by-1 scalar. Consequently, $\frac{\mathrm{d}f_i}{\mathrm{d}\hat{y}_i}$ is also a 1-by-1 scalar. From equation 13, $\frac{\mathrm{d}\hat{y}_i}{\mathrm{d}\bar{\omega}}$ $=(\bar{x}_i)'$. Therefore, we can apply the chain rule and reduce space-complexity to $\theta(min(M,N)+d)$, because we can use the vector $\bar{x}_i$ to re-compute $\frac{\mathrm{d}f_i}{\mathrm{d}\bar{\omega}}$ from the scalar $\frac{\mathrm{d}f_i}{\mathrm{d}\hat{y}_i}$.

Theorem 4. Applying the chain-rule for using *SAG* in matrix facotrization would result in $\theta(min(M,N)+nDims*(min(M,N)+nRows+nCols))$ space.

PROOF. In matrix factorization, $\hat{a}_{ij}=(\bar{u}_i*\bar{v}_j)$ is a 1-by-1 scalar. Therefore, we can rewrite the gradients as

$$\frac{\mathrm{d}f}{\mathrm{d}\bar{u}_i}=\left(\frac{\mathrm{d}f}{\mathrm{d}\hat{a}_{ij}}\right)\frac{\mathrm{d}\hat{a}_{ij}}{\mathrm{d}\bar{u}_i}=(\bar{v}_j)'\left(\frac{\mathrm{d}f}{\mathrm{d}\hat{a}_{ij}}\right) \tag{15}$$

$$\frac{\mathrm{d}f}{\mathrm{d}\bar{v}_j}=\left(\frac{\mathrm{d}f}{\mathrm{d}\hat{a}_{ij}}\right)\frac{\mathrm{d}\hat{a}_{ij}}{\mathrm{d}\bar{v}_j}=(\bar{u}_i)'\left(\frac{\mathrm{d}f}{\mathrm{d}\hat{a}_{ij}}\right) \tag{16}$$

$\left(\frac{\mathrm{d}f}{\mathrm{d}\hat{a}_{ij}}\right)$ is a 1-by-1 scalar, and the chain-rule approach stores $min(M,N)$ copies, occupying $\theta(min(M,N))$ space.

Unfortunately both U and V change over time in matrix factorization. When we apply the chain-rule, we cannot just use the current versions of $\bar{u}_i$ and $\bar{v}_j$. We must use and thus must store the past versions of $\bar{u}_i^l$ and $\bar{v}_j^l$ at the last time l that the entry a_{ij} (in matrix A) was sampled. Both $\bar{u}_i^l$ and $\bar{v}_j^l$ are vectors of length *nDims*. Therefore, using the chain rule induces an additional $(min(M,N)*2*nDims)$ space. When we include the memory of aggregated gradients $\bar{m}_U$ and $\bar{m}_V$, the total space-complexity becomes larger than the naïve approach with $\theta(min(M,N)+nDims*(min(M,N)+nRows+nCols))$ space. The chain-rule approach yields space savings in supervised machine-learning because $\bar{x}_i$ does not change over time; so there is no need to store past versions of $\bar{x}_i$. $\square$

4. APPROACH

Similar to the chain-rule approach, SAG-RE does not store and re-computes $\bar{m}^t_{entry(b).i}$ in equation 8 and $\bar{m}^t_{entry(b).j}$ in equation 11:

$$\bar{m}^{t}_{entry(b).i} = recomputed\ \frac{\mathrm{d}f(\bar{u}^{s}_{entry(b).i}, \bar{v}^{s}_{entry(b).j})}{\mathrm{d}\bar{u}^{s}_{entry(b).i}} \quad (17)$$

$$\bar{m}^{t}_{entry(b).j} = recomputed\ \frac{\mathrm{d}f(\bar{u}^{s}_{entry(b).i}, \bar{v}^{s}_{entry(b).j})}{\mathrm{d}\bar{u}^{s}_{entry(b).j}} \quad (18)$$

The chain-rule approach is undesirable because it must store $min(M, N)$ different copies of past versions of $\bar{m}^{t}_{entry(b).i}$ and $\bar{m}^{t}_{entry(b).j}$. There are two problems. First, each entry can come from a different iteration; or different entries can come from different iterations. Second, the same entry may get sampled more than once at two or more different iterations.

To save space, we must store as few copies of $\bar{m}^{t}_{entry(b).i}$ and $\bar{m}^{t}_{entry(b).j}$ as possible. SAG-RE resolves the two problems above with two steps. First, SAG-RE predicts ahead the entires that we are going to sample. Second, SAG-RE performs a full deterministic gradient FG just before SAG-RE re-samples the same entry.

At the iteration that SAG-RE performs a full deterministic gradient, we call it iteration s, SAG-RE stores 4 matrices:

1. the actual *user* matrix U at iteration s: U^s
2. the actual *item* matrix V at iteration s: V^s
3. aggregated memory gradient for *user* matrix U: $\bar{m}^s_U$
4. aggregated memory gradient for *item* matrix V: $\bar{m}^s_V$

We should distinguish that U^s and V^s are stored *just before* SAG-RE performs a full deterministic gradient at iteration s. The significance is that we will use U^s and V^s to re-compute the fine-grain memory gradients at future iterations $t > s$.

$\bar{m}^s_U$ and $\bar{m}^s_V$ are the direct outcome results of the full deterministic gradient. The reason is that FG samples all N entries and thus resets every possible fine-grain gradient in memory. Thus we store $\bar{m}^s_U$ and $\bar{m}^s_V$ *after* SAG-RE performs an iteration of FG.

At the iterations t in between SAG-RE performs two FG's, e.g. $s < t < s'$, SAG-RE performs iterations of ordinary SAG. When SAG-RE performs ordinary SAG, SAG-RE computes but does **not store** the latest version of fine-grain gradients of individual entries:

$$\bar{m}^{t+1}_{entry(b).i} = \frac{\mathrm{d}f(\bar{u}_{entry(b).i}, \bar{v}_{entry(b).j})}{\mathrm{d}\bar{u}_{entry(b).i}}\ in\ equation\ 7$$

$$\bar{m}^{t+1}_{entry(b).j} = \frac{\mathrm{d}f(\bar{u}_{entry(b).i}, \bar{v}_{entry(b).j})}{\mathrm{d}\bar{v}_{entry(b).j}}\ in\ equation\ 10$$

SAG-RE simply updates $\bar{m}^s_U$ and $\bar{m}^s_V$ with the newly computed fine-grain gradients, as equation 8 and equation 11 show.

After we perform an iteration of FG, we predict upcoming entries ahead of time. Therefore, at future iterations $t > s$ after a FG, we ensure that the different entries that we are going to sample are **distinct** before we perform another iteration of full deterministic gradient. The significance of having *distinct* entries is that, at future iterations $t > s$, we will not overwrite any fine-grain gradient of individual entries: e.g. $\bar{m}^{t}_{entry(b).i}$ in equation 8 and $\bar{m}^{t}_{entry(b).j}$ in equation 11. Therefore, we can *re-compute* all possible fine-grain gradients of individual entries from a single copy of the *user* matrix U^s and the *item* matrix V^s, that SAG-RE stored at the same iteration s.

Before we perform another iteration of FG, we do not store any fine-grain gradient $\bar{m}^{t+1}_{entry(b).i}$ or any $\bar{m}^{t+1}_{entry(b).j}$. The reason is that we do not ever need them: SAG-RE ensures that we will perform an iteration of FG before we re-sample any identical entry. The purpose of an iteration of FG at iteration $s' > t$ is to reset all fine-grain gradients of individual entries at the same iteration s'. This way we will not need any of the fine-grain gradients at iterations $t < s'$ because we will not visit the same entries again until after we do a full reset. Not storing the newly-computed fine-grain gradients saves $\theta(min(M, N) * nDims)$ space.

SAG-RE re-computes the indivdiaul fine-grain gradients from the raw U^s and V^s matrices; doing so preserves generality. We do not use the chain-rule: not all objective functions is compatible with it. Both [10, 5] do not work with the chain-rule because computing the fine-grain gradient of an entry requires not just $(\hat{a}_{ij} = \bar{u}_i * \bar{v}_j)$, but also $(\hat{a}_{ik} = \bar{u}_i * \bar{v}_k)$ for all $k \neq j$.

Next we prove SAG-RE preserves the theoretical advantages of SAG, and SAG-RE is compact in space.

Theorem 5. SAG-RE has convergence rate at least as fast as SAG.

PROOF. The proofs of SAG's convergence rates [8, 7] do not restrict where the starting points are for optimization. In matrix factorization, the meaning is that we can start SAG with any (random) matrices U and V (e.g. U^S and V^S) and still experience the convergence rates of SAG. Therefore, at iterations that SAG-RE performs SAG, SAG-RE has convergence rate equal to SAG. Similarly, the convergences rates of full determinsitic gradient (FG) allows any U and V as the starting matrices. Therefore, when SAG-RE performs FG, SAG-RE inherits the convergence rates of FG. FG has the fastest convergence rates. Therefore, at any iteration, SAG-RE has convergence rates at least as fast as SAG. □

Theorem 6. SAG-RE has $\theta(1)$ time-complexity and is asymptotically as efficient as both SAG and stochastic gradient.

PROOF. At iteartions that SAG-RE performs SAG, we totally re-compute the past versions of the fine-grain gradients for the same batch of samples. The re-computing done by SAG-RE essentially doubles the amount of computation compared to SAG and stochastic gradient. Doubling the amount of computation multiplies time-complexity by only a constant; thus SAG-RE preserves the low iteration cost of SAG.

The interesting case is when SAG-RE performs an iteration of FG, because an iteration of FG samples all N entries. After an iteration of FG, N is also the maximum number of *distinct* entries that SAG-RE can predict ahead. Spread over $\theta(N)$ iterations, the overhead associated with an iteration of FG armortizes to $\theta(1)$ over time. In average, the re-computation and armortization together triple SAG-RE's expected iteration cost over time. Tripling also multiplies overall time-complexity by only a constant. Without loss of generality, SAG-RE posseses $\theta(1)$ iteration cost even when SAG-RE performs iterations of FG (up to) a constant number of times for every $\theta(N)$ iteraions.

Our evaluation will illustrate that, despite tripling the itearation cost, SAG-RE still returns the best optimizations within the shortest time. □

Theorem 7. SAG-RE has $\theta(N + min(M, N) + nDims * (nRows + nCols))$ space-complexity and is asymptotically as compact as any memory-less gradient method.

PROOF. U^s and $\bar{m}_U^s$ are $nRows$-by-$nDims$ matrices. V^s and $\bar{m}_V^s$ are $nDims$-by-$nCols$ matrices.

SAG-RE also stores the indices of entries that SAG-RE is going to sample in the future; these indices take $\theta(min(E[M], N))$ space. When $M > N$, $min(M, N)$ returns N; and space-complexity becomes $\theta(N + N + nDims * (nRows + nCols))$. When $N > M$, space complexity becomes $O(N + N + nDims * (nRows + nCols))$.

The extra matrices and indices that SAG-RE stores does not asymptotically increase the most-compact possible space-complexity (*Theorem 2*). Both full deterministic gradient FG and stochastic gradient SG are memory-less methods and thus they also achieve the most-compact possible space-complexity in *Theorem 2*. Indeed, SAG-RE is as compact as any memory-less method because the space-complexity does not become any more compact than what is proved in *Theorem 2*. Our evaluation will illustrate that the actual memory usage are similar among SAG-RE, FG and SG. □

5. IMPLEMENTATION

Matlab and Mental Model. We implement SAG-RE in Matlab because Matlab is a widely popular tool for prototyping algorithms in machine learning and data mining.

Matlab has an advantage that the system model of the source-code closely matches the mental model of the data-scientist. In the eyes of a data-scientist, the close match between the system model and the mental model makes the programming-language highly usable.

For example, $\hat{A} = U * V$ is a mathematical representation for matrix multiplication. In Matlab, the code to multiply two matrices is exactly identical to the mathematical representation above. Thus data-scientists can exert the least amount of mental effort and seamlessly translate their thoughts into code.

In C, C++ or Java, data-scientists must deal with additional mental overhead that distracts them from concentrating on their primary goal of formulating an algorithm: e.g. memory allocation; pointers and references; variable type; the specific function name to use; namespaces; and the precise number, order and type of input arguments.

Architecture. We architect our implementation so that we implement SAG-RE only once in only one self-contained file. Given an objective function, switching between gradient methods requires changing only one line of code. That line of code is easily identifiable, locatable, modifiable and self-contained. Changing that line of code also does not have any side-effects and does not require changing other lines of code.

Batching and Parallelism. Matlab vectorizes computations and parallelizes matrix operations by default. Matlab's parallel computing toolkit allows Matlab-users to run the same parallel version of code on a diversity of hardware from multi-core CPUs and CUDA-GPUs to multi-machine clusters. SAG-RE is implemented so that at each iteration, we can calculate in parallel the fine-grain gradients of individual entries sampled in the batch.

Re-computation can also run in parallel to the computation of new gradients, because the re-computation of a past gradient is entirely independent from the computation of a new gradient. To closely examine the true additional cost of re-computing, our evaluation does *not* parallelize re-computing.

6. EVALUATION

6.1 Research Questions

Quality vs. Time. Our first set of research questions focuses on optimization quality vs. time. Given more time, any gradient method yields a better optimization. Our focus here is to identify which gradient method is the most suitable for data-scientists prototyping recommender systems. In terms of suitability, we mean the gradient method that yields the best quality optimization within the shortest amount of time. Here, we consider the general SAG approach *as is*. The next set of research questions studies the specific *space vs. time* trade-off between SAG-RE and the naïve approach to SAG.

Between SAG, full deterministic gradient and stochastic gradient,

1. Which gradient method yields a better optimization given the same amount of time?
2. Which gradient method uses the shortest amount of time to reach a similar quality of optimization?
3. Can SAG and specifically SAG-RE work well with different objective functions in recommender systems?
4. Can SAG and specifically SAG-RE work well with different matrix datasets?

Space vs. Time. Our second set of research questions investigates whether re-computing is worth the additional time. Here, we investigate the actual space vs. time trade-off between SAG-RE vs. the naïve approach to SAG:

Compared to the naïve approach to SAG, in practice

5. How much slower is SAG-RE due to re-computing?
6. How much memory does SAG-RE save?

6.2 Experimental Setup

Objective Functions. The objective functions we choose already uses full deterministic gradient (FG) or stochastic gradient (SG). In general, any function that is differentiable, and specifically any function that uses (FG) or (SG) can use SAG and SAG-RE. If a function is convex, then gradient methods guarantee a global optimum over time.

1. Least-squares: L2 and its variants [11, 3, 4] are popular objective functions when building recommender systems.
2. CLiMF [10]: Collaborative-Less-is-More-Filtering uses ordinal logistic regression to smooth the mean reciprocal rank function and to learn how a user ranks different items; CLiMF performs gradient ascent because the optimization goal is to maximize an objective function.
3. BPR-MF [5]: Bayseian Personalized Learning has an objective function that minimizes the difference between any two *item* ratings (column entries) of the same user (same row). BPR-MF performs gradient descent.

Sub-datasets.

Hyper Parameters. For the purpose of comparison, we standardize all hyper-parameters across all objective functions, all datasets, and all gradient methods. The only exception is that we run full deterministic gradient FG) for only 500 iterations vs. 5,000 for stochastic gradient (*SG*) and *SAG*.

Convergence theory guarantees that given the same number of iterations, *FG* yields a much better quality optimization than *SG*. However, our goal is to identify the gradient method that yields the best quality optimization within the shortest amount of time. Therefore, we want to see whether *FG* would take longer to yield a similar quality of optimization as *SG*, and how much longer. Through experience with our objective functions and datasets, we observed that 500 iterations *FG* yields a similar quality of optimization as *SG*. As a result, we run *FG* to 500 iterations, and compare how much longer 500 iterations of *FG* would take than 5000 iterations of *SG*.

- Step size or learning rate: 0.0001
- Regularization λ: 0.001; λ is identical for regularizing both *user* matrix U and *item* matrix V
- Iterations: 5000 for *SG* and *SAG*, which is roughly 10% of the number of non-zero entries in each sub-dataset.
- Latent dimensions (*nDims*): 5

For gradient descent, step size is $\alpha < 0$ and $\lambda > 0$; for ascent, step size is $\alpha > 0$ and $\lambda < 0$.

Hardware and OS. A MacBook Pro run all experiments that study optimization *Quality vs. Time.* Our MacBook Pro is the Late 2013 15-inch version [1]; it has OS-X Yosemite, 2.3Ghz Intel i7 quad-core CPU, 16GB RAM, and a Nvidia 750M GPU.

When studying *Space vs. Time*, we measure memory usage after the first iteration. Initially we plan to run all experiments on the MacBook Pro. However, the memory-profiing feature of Matlab works only on Windows.

For the sub-datasets, we run the *memory* experiment on a Dell XPS 12 [2] laptop. The Dell XPS 12 has Windows 8.1, 1.6Ghz Intel i5 dual-core CPU, 4GB RAM, and integrated graphics.

For the full datasets, we run the *memory* experiment on a remote server that has more RAM. Our remote server has Windows Server 2008R2, 2.50Ghz Intel Xeon 2x quad-core CPUs (total 8 CPU cores) and 16GB RAM.

Both MacBook Pro and remote server have Matlab R2014a; Dell XPS 12 has Matlab R2012a. All 3 computers have the Matlab parallel computing toolkit.

6.3 Quality vs. Time

For the purpose of comparison, we fix the seed for generating random numbers so that Measure optimization quaity in each iteration. report the best optimixatiom

7. FUTURE WORK & CONCLUSION

This paper is the first in the series of our study on data scientists prototyping model-based recommender systems. We explored the convex-optimization perspective of the problem: we propose Stochastic Average Gradient as a viable alternative to Full Deterministic gradient and Stochastic gradient. By taking advantage of *SAG*'s fast convergence rate and low iteration cost, we aim to enable data-scientists run more experiments and produce high quality results with less time. In theory, we proved that our extension and adaptation of *SAG* preserves the fast convergence rate as the original *SAG*. Furthermore, SAG-RE has asymptotic time complexity as efficient as gradient methods with the lowest itreation cost, and asymptotic space complexity as compact as any memory-less gradient methods. In practice, through extensive evaluation we demonstrated that, even without any fine-tuning or optimization of the implementation, SAG-RE still outperforms both full deterministic gradient and stochastic gradient in terms of reaching the best quality optimization within the same amount of time. Following up, we provided evidence that full deterministic gradient and stochastic gradient would take much longer to reach a quality of optimization similar to SAG-RE.

Currently we are extending SAG-RE in two directions. Both directions relate to running an iteration of full deterministic gradient in SAG-RE. First, we are investigating if it is beneficial to run an iteration of full deterministic gradient more often. In our experiments, we observed that both *SG* and *SAG* may converge early; the optimization may get stuck at a local sub-optimum for a long number of iterations. Thus we are exploring if an iteration of full deterministic gradient would get the optimization back on track in case SAG-RE gets stuck. Secondly, we aim to investigate how well SAG-RE would perform in the production environment, and in distributed systems potentially running in parallel, because running full deterministic gradient even once can be prohibitive for full-scale datasets with millions to billions of non-zero entries.

In the future, we also aim to complete our ongoing work on the metrics perspective and on the software engineering perspective. Given a dataset, the quality of a recommender system is often evaluated in various metrics: e.g. precision, recall, area under curve, reciprocal rank, NDCG, and variants of the above such as top-K precision and top-K hit rate. Many papers in the literature claim their objective function is better by illustrating that their objective function performs in some of these metrics better than other objective functions. Therefore, in the metrics perspective, we are exploring and investigating which factors are more relevant and important towards scoring high in the various metrics: is it the objective function, the method for convex-optimization such as *SAG*, other fine-tuning mechanisms such as bootstrapping, or the hyper-parameters that we use in convex-optimization. All of these factors can be dataset-specific. Indeed, our inherent assumption in this paper is that a better quality optimization yields better recommender systems. In the future, we would like to explore if there are other factors that are more worthwhile than a fast convergence rate or a low iteration cost towards better recommender systems.

In the software engineering perspective, we study how to increase the productivity of data scientists. At this point, we are designing and developing a *mix-n-match* or *plug-n-play* framework that enables data scientists in a least effort way, to very rapidly prototype and experiment many different combinations of objective functions, datasets, gradient methods, hyper parameters and evaluation metrics.

8. REFERENCES

[1] Apple. MacBook Pro (Retina, 15-inch, Late 2013) - Technical Specifications. `http://support.apple.com/kb/SP690`.

[2] Dell. XPS 12 2-in-1 Ultrabook™ with Touch Screen. `http://www.dell.com/us/p/xps-12-9q33/pd`.

[3] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. IEEE, 2008.

[4] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 502–511. IEEE, 2008.

[5] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461. AUAI Press, 2009.

[6] J. D. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM, 2005.

[7] N. L. Roux, M. Schmidt, and F. R. Bach. A stochastic gradient method with an exponential convergence _rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671, 2012.

[8] M. Schmidt, N. L. Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *arXiv preprint arXiv:1309.2388*, 2013.

[9] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, and A. Hanjalic. Gapfm: Optimal top-n recommendations for graded relevance domains. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2261–2266. ACM, 2013.

[10] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 139–146. ACM, 2012.

[11] H. Steck. Training and testing of recommender systems on data missing not at random. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 713–722. ACM, 2010.