Fast Differentially Private Matrix Factorization

Ziqi Liu
MOEKLINNS Lab, Computer Science
Xi'an Jiaotong University
Xi'an, China
ziqilau@gmail.com

Yu-Xiang Wang
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA
yuxiangw@cs.cmu.edu

Alexander J. Smola
CMU Machine Learning
Marianas Labs Inc.
Pittsburgh, PA
alex@smola.org

ABSTRACT

Differentially private collaborative filtering is a challenging task, both in terms of accuracy and speed. We present a simple algorithm that is provably differentially private, while offering good performance, using a novel connection of differential privacy to Bayesian posterior sampling via Stochastic Gradient Langevin Dynamics. Due to its simplicity the algorithm lends itself to efficient implementation. By careful systems design and by exploiting the power law behavior of the data to maximize CPU cache bandwidth we are able to generate 1024 dimensional models at a rate of 8.5 million recommendations per second on a single PC.

Keywords

Differential Privacy; Collaborative Filtering; Scalable Matrix Factorization

1. INTRODUCTION

Privacy protection in recommender systems is a notoriously challenging problem. There are often two competing goals at stake: similar users are likely to prefer similar products, movies, or locations, hence sharing of preferences between users is desirable. Yet, at the same time, this exacerbates the type of privacy sensitive queries, simply since we are now not looking for aggregate properties from a dataset (such as a classifier) but for properties and behavior of other users 'just like' this specific user. Such highly individualized behavioral patterns are shown to facilitate provably effective user de-nonymization [25, 37].

Consider the case of a couple, both using the same location recommendation service. Since both spouses share much of the same location history, it is likely that they will receive similar recommendations, based on other users' preferences similar to theirs. In this context sharing of information is desirable, as it improves overall recommendation quality.

Moreover, since their location history is likely to be very similar, each of them will also receive recommendations to visit the place that their spouse visited (e.g. including places of ill repute), regardless of whether the latter would like to share this information or not. This creates considerable tension in trying to satisfy those two conflicting goals.

RecSys '15 September 16-20, 2015, Vienna, Austria

DOI: http://dx.doi.org/10.1145/2792838.2800191.

Differential privacy offers tools to overcome these problems. Loosely speaking, it offers the participants plausible deniability in terms of the estimate. That is, it provides guarantees that the recommendation would also have been issued with sufficiently high probability if another specific participant had not taken this action before. This is precisely the type of guarantee suitable to allay the concerns in the above situation [8].

Recent work, e.g. by Mcsherry and Mironov [20] has focused on designing *custom built* tools for differential private recommendation. Many of the design decisions in this context are hand engineered, and it is nontrivial to separate the choices made to obtain a differentially private system from those made to obtain a system that works well. Furthermore, none of these systems [20, 36] lead to very fast implementations.

In this paper we show that a large family of recommender systems, namely those using matrix factorization, are well suited to differential privacy. More specifically, we exploit the fact that sampling from the posterior distribution of a Bayesian model, e.g. via Stochastic Gradient Langevin Dynamics (SGLD) [35], can lead to estimates that are sufficiently differentially private [34]. At the same time, their stochastic nature makes them well amenable to efficient implementation. Their generality means that we *need not custom-design a statistical model for differential privacy* but rather that is possible to *retrofit an existing model* to satisfy these constraints. The practical importance of this fact cannot be overstated — it means that no costly re-engineering of deployed statistical models is needed. Instead, one can simply reuse the existing inference algorithm with a trivial modification to obtain a differentially private model.

This leaves the issue to performance. Some of the best reported results are those using GraphChi [15], which show that state-of-the-art recommender systems can be built using just a single PC within a matter of hours, rather than requiring hundreds of computers. In this paper, we show that by efficiently exploiting the power law properties inherent in the data (e.g. most movies are hardly ever reviewed on Netflix), one can obtain models that achieve peak numerical performance for recommendation. More to the point, they are 3 times faster than GraphChi on identical hardware.

In summary, this paper describes the by far the fastest matrix factorization based recommender system and it can be made differentially privately using SGLD without losing performance. Most competing approaches excel at no more than one of those aspects. Specifically,

1. It is efficient at the state of the art relative to other matrix factorization systems.
   - we develop a cache efficient matrix factorization framework for general SGD updates.
   - we develop a fast SGLD sampling algorithm with bookkeeping to avoid adding the Gaussian noise to the whole

parameter space at each updates while still maintaining the correctness of the algorithm.

2. And it is differentially private.

- We provably show that sampling from a scaled posterior distribution for matrix factorization system can guarantee user-level differential privacy.
- We present a personalized differentially private method for calibrating each user's privacy and accuracy.
- We only privately release V to public, and design a local recommender system for each user.

Experiments confirm that the algorithm can be implemented with high efficiency, while offering very favorable privacy-accuracy tradeoff that nearly matches systems without differential privacy at meaningful privacy level.

2. BACKGROUND

We begin with an overview of the relevant ingredients, namely collaborative filtering using matrix factorization, differential privacy and a primer in computer architecture. All three are relevant to the understanding of our approach. In particular, some basic understanding of the cache hierarchy in microprocessors is useful for efficient implementations.

2.1 Collaborative Filtering

In collaborative filtering we assume that we have a set of $\mathcal{U}$ users, rating $\mathcal{V}$ items. We only observe a small number of entries r_{ij} in the rating matrix R. Here r_{ij} means that user i rated item j. A popular tool [14] to deal with inferring entries in $R \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{V}|}$ is to approximate R by a low rank factorization, i.e.

$$R \approx UV^\top \text{ where } U \in \mathbb{R}^{|\mathcal{U}|k} \text{ and } V \in \mathbb{R}^{|\mathcal{V}|k} \tag{1}$$

for some $k \in \mathbb{N}$, which denotes the dimensionality of the feature space corresponding to each item and movie. In other words, (user,item) interactions are modeled via

$$r_{ij} \approx \langle u_i, v_j \rangle + b_i^u + b_j^m + b_0. \tag{2}$$

Here u_i and v_j denote row-vectors of U and V respectively, and b_i^u and b_j^m are scalar offsets responsible for a specific user or movie respectively. Finally, b_0 is a common bias.

A popular interpretation is that for a given item j, the elements of v_j measure the extent to which the item possesses those attributes. For a given user i the elements of u_i measure the extent of interest that the user has in items that score highly in the corresponding factors. Due to the conditions proposed in the Netflix contest, it is common to aim to minimize the mean squared error of deviations between true ratings and estimates. To address overfitting, a norm penalty is commonly imposed on U and V. This yields the following optimization problem

$$\min_{u,v} \sum_{i,j \in R} (r_{ij} - \langle u_i, v_j \rangle - b_i^u - b_j^m - b_0)^2 + \lambda(||U||_2^2 + ||V||_2^2)$$

A large number of extensions have been proposed for this model. For instance, incorporating co-rating information [28], neighborhoods, or temporal dynamics [13] can lead to improved performance. Since we are primarily interested in demonstrating the efficacy of differential privacy and the interaction with efficient systems design, we focus on the simple inner-product model with bias.

Bayesian View. Note that the above optimization problem can be viewed as an instance of a Maximum-a-Posteriori estimation problem. That is, one minimizes

$$\begin{aligned} -\log p(U, V|R, \lambda_r, \Lambda_u, \Lambda_v) = &-\log \mathcal{N}(R|\langle U, V\rangle, \lambda_r^{-1}) \\ &-\log \mathcal{N}(U|0, \Lambda_u^{-1}) - \log \mathcal{N}(V|0, \Lambda_v^{-1}) \end{aligned}$$

where, up to a constant offset

$$-\log p(r_{ij}|u_i, v_j) = \lambda_r(r_{ij} - \langle u_i, v_j \rangle - b_i^u - b_j^m - b_0)^2$$

and $-\log p(U) = U\Lambda_u U^\top$ and likewise for V. In other words, we assume that the ratings are conditionally normal, given the inner product $\langle u_i, v_j \rangle$, and the factors u_i and v_j are drawn from a normal distribution. Moreover, one can also introduce priors for $\lambda_r, \Lambda_u, \Lambda_v$ with a Gamma distribution $\mathcal{G}(\cdot|\alpha, \beta)$.

While this setting is typically just treated as an afterthought of penalized risk minimization, we will explicitly use this when designing differentially private algorithms. The rationale for this is the deep connection between samples from the posterior and differentially private estimates. We will return to this aspect after introducing Stochastic Gradient Langevin Dynamics.

Stochastic Gradient Descent. Minimizing the regularized collaborative filtering objective is typically achieved by one of two strategies: Alternating Least Squares (ALS) and stochastic gradient descent (SGD). The advantage of the former is that the problem is biconvex in U and V respectively, hence minimizing $U|V$ or $V|U$ are convex. On the other hand, SGD is typically faster to converge and it also affords much better cache locality properties. Instead of accessing e.g. all reviews for a given user (or all reviews for a given movie) at once, we only need to read the appropriate tuples. In SGD each time we update a randomly chosen rating record by:

$$\begin{aligned} u_i &\leftarrow (1 - \eta_t\lambda)u_i + \eta_t v_j \left(r_{ij} - \langle u_i, v_j \rangle - b_i^u - b_j^m - b_0\right) \\ v_j &\leftarrow (1 - \eta_t\lambda)v_j + \eta_t u_i \left(r_{ij} - \langle u_i, v_j \rangle - b_i^u - b_j^m - b_0\right) \end{aligned} \tag{3}$$

One problem of SGD is that trivially parallelizing the procedure requires memory locking and synchronization for each rating, which could significantly hamper the performance. [27] shows that a lock-free scheme can achieve nearly optimal solution when the data access is sparse. We build on this *statistical* property to obtain a *fast system* which is suitable for differential privacy.

2.2 Differential Privacy

Differential privacy (DP) [7, 9] aims to provide means to cryptographically protect personal information in the database, while allowing aggregate-level information to be accurately extracted. In our context this means that we protect user-specific sensitive information while using aggregate information to benefit all users.

Assume the actions of a statistical database are modeled via a randomized algorithm $\mathcal{A}$. Let the space of data be $\mathcal{X}$ and data sets $X, Y \in \mathcal{X}^n$. Define $d(X, Y)$ to be the edit distance or Hamming distance between data set X and Y, for instance if X and Y are the same except one data point then we have $d(X, Y) = 1$.

Definition 1 (Differential Privacy)**.** *We call a randomized algorithm $\mathcal{A}$ (ϵ, δ)-differentially private if for all measurable sets $S \subset Range(\mathcal{A})$ and for all $X, X' \in \mathcal{X}^n$ such that the hamming distance $d(X, X') = 1$,*

$$\mathbb{P}(\mathcal{A}(X) \in S) \leq \exp(\epsilon)\mathbb{P}(\mathcal{A}(X') \in S) + \delta$$

If $\delta = 0$ we say that $\mathcal{A}$ is ϵ-differential private.

The definition states that if we arbitrarily replace any individual data point in a database, the output of the algorithm doesn't change much. The parameter ϵ in the definition controls the maximum amount of information gain about an individual person in the database given the output of the algorithm. When ϵ is small, it prevents any forms of linkage attack to individual data record (e.g., linkage of Netflix data

to IMDB data [25]). We refer readers to [8] for detailed interpretations of the differential privacy in statistical testing, Bayesian inference and information theory.

An interesting side-effect of this definition in the context of collaborative filtering is that it also limits the influence of so-called whales, i.e. of users who submit extremely large numbers of reviews. Their influence is also curtailed, at least under the assumption of an equal level of differential privacy per user. In other words, differential privacy confers robustness for collaborative filtering.

Wang et al. [34] show that posterior sampling with bounded log-likelihood is essentially exponential mechanism [21] therefore protecting differential privacy for free (similar observations were made independently in [23, 5]). Wang et al. [34] also suggests a recent line of works [35, 4, 6] that use stochastic gradient descent for Hybrid Monte Carlo sampling essentially preserve differential privacy with the same algorithmic procedure. The consequence for our application is very interesting: if we trust that the MCMC sampler has converged, i.e. if we get a sample that is approximately drawn from the posterior distribution, then we can use one sample as the private release. If not, we can calibrate the MCMC procedure itself to provide differential privacy (typically at the cost of getting a much poorer solution).

2.3 Computer Architecture

A key difference between generic numerical linear algebra, as commonly used e.g. for deep networks or generalized linear models, and the methods used for recommender systems is the fact that the access properties regarding users and items are highly nonuniform. This is a significant advantage, since it allows us to exploit the caching hierarchy of modern CPUs to benefit from higher bandwidth than what disks or main memory access would permit.

A typical computer architecture consists of a hard disk, solid-state drive (SSD), random-access memory (RAM) and CPU cache. A good algorithm design should be pushing the data flow to CPU cache level and *hide the latency* from SSD or even RAM and amplify the available bandwidth.

The key strategy in obtaining high throughput collaborative filtering systems is to obtain peak bandwidth on *each* of the subsystems by efficient caching. That is, if a movie is frequently reused, it is desirable to retain it in the CPU cache. This way, we will neither suffer the high latency (100ns per request) of a random read from memory, nor will we have to pay for the comparably slower bandwidth of RAM relative to the CPU cache.

3. DIFFERENTIALLY PRIVATE MATRIX FACTORIZATION

We start by describing the key ideas and algorithmic framework for differentially private matrix factorization. The method, which involves preprocessing data and then sampling from a scaled posterior distribution, is provably differentially private and has profound statistical implications. Then we will describe a specific Monte Carlo sampling algorithm: Stochastic Gradient Langevin Dynamics (SGLD) and justify its use in our setting. We then come up with a novel way to personalize the privacy protection for individual users. Finally, we discuss how to develop fast cache-efficient solvers to exploit bandwidth-limited hardware such that it can be used for general SGD-style algorithms.

Our differential privacy mechanism relies on a recent observation that posterior sampling preserves differential privacy, provided that the log-likelihood of each user is uniformly bounded [34]. This simple yet remarkable result suggests that sampling from posterior distribution is differentially private for free to some extent. In our context, the claim is that, if[1] $\max_{U,V,R,i} \sum_{j \in R_i} (r_{ij} - \langle u_i, v_j \rangle)^2 \leq B$ then the method that outputs a sample from

$$P(U,V) \propto \exp\left(-\sum_{(i,j) \in R} (r_{ij} - \langle u_i, v_j \rangle)^2 + \lambda(\|U\|_F^2 + \|V\|_F^2) \right)$$

preserves $4B$-differential privacy. Moreover, when we want to set the privacy loss ϵ to another number, we can easily do this by simply rescaling the entire expression by $\epsilon/4B$.

The question now is whether $\max_{U,V,R,i} \sum_{j \in R_i} (r_{ij} - \langle u_i, v_j \rangle)^2$ is bounded. Since the ratings are bounded between $1 \leq r_{ij} \leq 5$ and we can consider a reasonable sublevel set $\{U, V \mid \max_{i,j} |u_i^T v_j| \leq \kappa\}$, we have every summand to be bounded by $(5 + \kappa)^2$. This does not affect the privacy claim as long as κ is chosen independent to the data.

B could still be large, if some particular users rated many movies. This issue is inevitable even if all observed users have few ratings, since differential privacy also protects users not in the database. We propose two theoretically-inspired algorithmic solutions to this problem:

Trimming: We may randomly delete ratings for those who rated a lot of movies so that the maximum number of ratings from a single user τ will not be too much larger than the average number of ratings. This procedure is the underlying gem that allows OptSpace (the very first provable matrix factorization based low-rank matrix completion method) [12] to work.

Reweighting: Alternatively, one can weight each user appropriately so that those who rated many movies will have smaller weight for each rating. Mcsherry and Mironov [20] used this reweighting scheme for controlling privacy loss. A similar approach is considered in the study of non-uniform and power-law matrix completion [22, 30], where the weighted trace norm has the same effect as if we reweight the loss-functions.

In addition, these procedures have their practical benefits for the robustness of the recommendation system, since they prevents any malicious user from injecting too much impact into the system, see e.g., Wang and Xu [33], Mobasher et al. [24]. Another justification of these two procedures is that, if the fully observed matrix is truly in a low-dimensional subspace, neither of these two procedures changes the underlying subspace. Therefore, the solutions should be similar to the non-preprocessed version.

The procedure for differentially private matrix factorization (DPMF) is summarized in Algorithm 1. Note that this is a *conceptual* sketch (we will discuss an efficient variant thereof later). The following theorem guarantees that our procedure is indeed differentially private.

Theorem 1. *Algorithm 1 obeys ϵ-differential privacy if the sample is exact and $(\epsilon, (1 + e^{\epsilon})\delta)$-differential privacy if the sample is from a distribution δ-away from the target distribution in L_1 distance.*

The proof in [17] shows that this procedure uses in fact the exponential mechanism [21] with utility function being the negative MF objective and its sensitivity being $2B$. Note that this can be extended to considerably more complex models. This is the strength of our approach, namely that a large variety of algorithms can be adapted quite easily to differential privacy capable models.

[1]For convenience of notation we will omit the biases from the description below in favor of a slightly more succinct notation.

Algorithm 1 Differentially Private Matrix Factorization

Require: Partially observed rating matrix $R \in \mathbb{R}^{m \times n}$ with observation mask Ω. $m = \#$ of movies, $n = \#$ of users. Privacy parameter ϵ, a predefined positive parameter κ such that $\{U, V \mid u_i^T v_j \in [1-\kappa, 5+\kappa] \ \forall i, j\}$, rating range $[1, 5]$, max allowable number of ratings per-user τ, number of ratings of each user $\{m_1, ..., m_n\}$, weight of each user w, tuning parameter λ.

1: $B \leftarrow \max_{i=1,...,n} \min\{\tau, m_i\} w_i (5 - 1 + \kappa)^2$. ▷ Compute uniform upper bound.
2: Trim all users with ratings $> \tau$.
3: $F(U, V) := \sum_{i \in [i], j \in \Omega_i} w_i (R_{ij} - u_i^T v_j)^2 + \lambda(\|U\|_F^2 + \|V\|_F^2)$.
4: Sample $(U, V) \sim P(U, V) \propto e^{-\frac{\epsilon}{4B} F(U,V)}$
5: **while** $u_i^T v_j \notin [1-\kappa, 5+\kappa]$ for some i, j **do**
6: Sample $(U, V) \sim P(U, V) \propto e^{-\frac{\epsilon}{4B} F(U,V)}$
7: **return** (U, V)

3.1 Personalized Differential Privacy

Another interesting feature of the proposed procedure is that it allows us to calibrate the level of privacy protection for every user independently, via a novel observation that weights assigned to different users are linear in the amount of privacy we can guarantee for that particular user.

We will use the same sampling algorithm, and our guarantees in Theorem 1 still hold. The idea here is that we can customize the system so that we get a lower basic privacy protection for all users, say $\epsilon = 4B$. As we explained earlier this is the level of privacy that we can get more or less "for free". The protection of DP is sufficiently strong as to include even those users that are not in the database.

By adjusting the weight parameter, we can make the privacy protection stronger for particular users according to how much they set they want privacy. This procedure makes intuitive sense because if some user wants perfect privacy, we can set their weight to 0 and they are effectively not in the database anymore. For people who do not care about privacy, their ratings will be assigned default weight. Formally, we define personalized differential privacy as follows:

Definition 2 (Personalized Differential Privacy)**.** *An algorithm $\mathcal{A}$ is (ϵ, δ)-personalized differentially private for User i in database X if for any measureable set S in the range of the algorithm $\mathcal{A}$*

$$\mathbb{P}(\mathcal{A}(X) \in S) \leq e^{\epsilon} \mathbb{P}(\mathcal{A}(X') \in S) + \delta.$$

for any $X \in \mathcal{X}^n$ and X' is either $X \cup \{x_i\}$ or $X \backslash \{x_i\}$.

Note that instead of replacing a user, we are now only adding or removing one specific user, such that the notion of personalized privacy is well-defined. We claim that

Theorem 2. *If we set w_i for User-i such that*

$$B_i := \min\{\tau, m_i\} w_i (4 + \kappa)^2 \leq B,$$

then Algorithm 1 guarantees $\frac{\epsilon B_i}{2B}$-personalized differential privacy for User i.

We show the proof in [17]. Note that if we set $\epsilon = 4B$ (so we are essentially sampling from the posterior distribution), we get $2B_i$-Personalized DP for user i.

In summary, if we simply set $\epsilon = 4B$, the method protects $4B$-differential privacy for everybody at very little cost and by setting the weight vector w, we can provide personalized service for users who demands more stringent DP protection. Also, personalized privacy offers a new perspective in interpreting DP beyond the worst case scenario. For instance, users who are more predictable by the model will be likely to have stronger privacy protection. If a model fits the data well, even a large worst-case privacy loss ϵ could in fact provide meaningful protection to the large majority of users. Lastly, we recently become aware that personalized privacy (the exact same definition) has been independently developed in [10] along with a few basic properties that mirrors the standard DP. The difference is that we focus on the weighted posterior sampling aspect of it.

4. EFFICIENT SAMPLING VIA SGLD

Clearly, sampling from $\exp\left(-\frac{\epsilon}{4B} F(U, V)\right)$ is nontrivial. For a tractable approach we use a recent MCMC method named stochastic gradient Langevin dynamics (SGLD) [35], which is an annealing of stochastic gradient descent and Langevin dynamics that samples from the posterior distribution [26]. The basic update rule is

$$(u_i, v_j) = (u_i, v_j) - \eta_t \widehat{\nabla}_{(u_i, v_j)} F(U, V) + \mathcal{N}(0, \eta_t I) \qquad (4)$$

where $\widehat{\nabla}_{(u_i, v_j)} F(U, V)$ is a stochastic gradient computed using only one or a small number of ratings. In other words, the updates are almost identical to those used in stochastic gradient descent. The key difference is that a small amount of Gaussian noise is added to the updates. This allows us to solve it extremely efficiently. We will describe our efficient implementation of this algorithm in Section 5.4.

The basic idea of SGLD is that when we are far away from the basin of convergence, the gradient of the log-posterior $\widehat{\nabla}_{(u_i, v_j)} F(U, V)$ is much larger than the additional noise so the algorithm behaves like stochastic gradient descent. As we approach the basin of convergence and η_t becomes small, $\sqrt{\eta_t} \gg \eta_t$ so the noise dominates and it behaves like a Brownian motion. Moreover, as η_t gets small, the probability of accepting the proposal in Metropolis-Hastings adjustment converges to 1, so we do not need to do this adjustment at all as the algorithm proceeds, as designed above.

This seemingly heuristic procedure was later shown to be consistent in [29, 31], where asymptotic "in-law" and "almost sure" convergence of SGLD to the correct stationary distribution are established. More recently, Teh et al. [32] further strengthens the convergence guarantee to include any finite iterations. This line of work justifies our approach in that if we run SGLD for a large number of iterations, we will end up sampling from the distribution that provides us (ϵ, δ)-differential privacy. By taking more iterations, we can make δ arbitrarily small.

5. SYSTEM DESIGN

The performance improvement over existing libraries such as GraphChi are due to both cache efficient design, prefetching, pipelining, the fact that we exploit the power law property of the data, and by judicious optimization of random number generation. This leads to a system that comfortably surpasses even moderately optimized GPU codes.

We primarily focus on the Stochastic Gradient Descent solver and subsequently we provide some details on how to extend this to SGLD. Inference requires a very large number of following operations on data:

- Read a rating triple (i, j, r_{ij}), possibly from disk, unless the data is sufficiently tiny to fit into RAM.
- For each given pair (i, j) of users and items fetch the vectors u_i and v_j from memory.
- Compute the inner product $\langle u_i, v_j \rangle$ on the CPU.
- Update u_i, v_j and write their new values to RAM.

To illustrate the impact of these operations consider training a $2,048$ dimensional model on the 10^8 rating triples of Netflix. Per iteration this requires over 3.2TB read/write operations to RAM. At a main memory bandwidth of 20GB/s and a latency of 100ns for each of the 200 million cache misses each pass would take over 6 minutes. Instead, our code accomplishes this task in approximately 10 seconds by using the steps outlined below.

5.1 Processing Pipeline

To deal with the dataflow from disk to CPU, we use a pipelined design, decomposing global and local state akin to [1]. This means that we process users sequentially, thus reducing the retrieval cost per user, since the operations are amortized over all of their ratings. This effectively halves IO. Moreover, since the data cannot be assumed to fit into RAM, we pipeline reads from disk. This hides latency and avoids stalling the CPUs. The writer thread periodically snapshots the model, i.e. U and V to disk.

Algorithm 2 Cache efficient Stochastic Gradient Descent

Require: parameters U, V; ratings R; P threads,
preprocessing Split R into B blocks;
procedure READ ▷ Keep pipeline filled
  while #blocks in flight $\leq P$ **do**
    Read: block b from disk
    Sync: notify UPDATE about b
procedure UPDATE ▷ Update U, V
  while at least one of P processors is available **do**
    Sync: receive a new block b from READ
    for user i in b **do**
      for each rating $r_{ij} \in b$ from user i **do**
        Prefetch next movie factor v_{j+1} from data stream
        $u_i \leftarrow u_i - \eta_t \widehat{\nabla}_{u_i}$
        $v_j \leftarrow v_j - \eta_t \widehat{\nabla}_{v_j}$
        ($\widehat{\nabla}$ is either the exact or private gradient)
procedure WRITE
  if B_t blocks processed **then** save U, V

5.2 Cache Efficiency

The previous reasoning discussed how to keep the data pipeline filled and how to reduce the user-specific cache misses by preaggregating them on disk. Next we need to address cache efficiency with regard to movies. More to the point, we need to exploit cache locality relative to the CPU *core* rather than simply avoiding cache misses. The basic idea is that each CPU core exactly reads a cache line (commonly 64 bytes) from RAM each time, so algorithm designers should not waste it until that piece of cache line is fully utilized.

We exploit the fact that movie ratings follow a power law [11], as is evident e.g. on Netflix in Figure 1. This means that if we succeed at keeping frequently rated movies in the CPU cache, we should see substantial speedups. Note that traditional matrix blocking tricks, as widely used for matrix multiplications operations are not useful, due to the sparsity of the rating matrix R. Instead, we decompose the movies into tiers of popularity. To illustrate, considering a decomposition into three blocks consisting of the Top 500, the Next 4000, and the remaining long tail.

Within each block, we process a batch of users simultaneously. This way we can preserve the associated user vectors u_i in cache and we are likely to cache the movie vectors, too (in particular for the Top 500 block). Also, parallelizing all the updates for multiple users does not require locks. Movie parameters are updated in a Hogwild fashion [27].

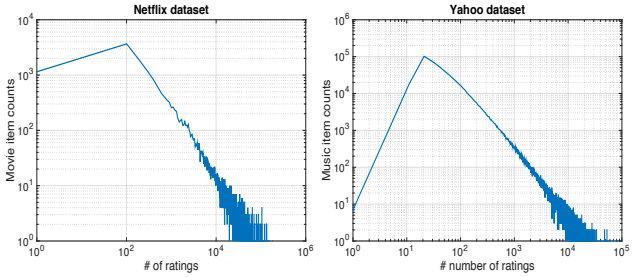

Figure 1: Distribution of items (Movies/Music pieces) as a function of their number of ratings. Many movies have 100 ratings or less, while the majority of ratings focuses on a small number of movies.

This design is particularly efficient for low-dimensional models since the Top 500 block fits into L1 cache (this amounts to 44% of all movie ratings in the Netflix dataset), the Next 4000 fits into L2, and ratings will typically reside in L3. Even in the extreme case of 2048 dimensions we can fit about 55% of all ratings into cache, albeit L3 cache.

5.3 Latency Hiding and Prefetching

To avoid the penalty for random requests we perform latency hiding by prefetching. That is, we actively request v_j in advance before the rating r_{ij} is to be updated. For dimensions less than 256, accurate prefetching leads to a dataflow of v_j into L1 cache. Beyond that, the size of the latent variables could be too big to benefit from the lowest level of caching due to limited size of caches in modern computers. We provide a detailed caching analysis in Section 6 to illustrate the effect of these techniques.

5.4 Optimizations for SGLD

The data flow of SGLD is almost analogous to that in SGD, albeit with a number of complications. First off, note that (4) applies to the whole parameter *matrix* U, V rather than just to a single vector. Following [3] we can derive an unbiased approximation of $\widehat{\nabla}_{u_i}$ in (4) which is nonzero only for (u_i, v_j) as follows:

$$\widehat{\nabla}_{u_i} = -N\lambda_r \left(r_{ij} - \langle u_i, v_j \rangle\right) v_j + \frac{N}{N_i} u_i^\top \Lambda_u u_i$$

where N, N_i denote number of rating data rated by all and rated by user i respectively. The parameters $\lambda_r, \Lambda_u, \Lambda_v$ do not incur any major cost — Λ_u, Λ_v are diagonal matrices with a Gamma distribution over them. We simply perform Gibbs sampling once per round. However, the most time-consuming part is to sample the remaining vectors, i.e. $\mathrm{P}(U^{-i}, V^{-i}|R, \text{rest})$ since it both requires dense updates and moreover, it requires many random numbers, which adds nontrivial cost.

Dense Updates: Note that unless we encounter the triple (i, j, r_{ij}) all other parameters are only updated by adding Gaussian noise. This means that by keeping track of when a parameter was last updated, we can simply aggregate the updates (the Normal distribution is closed under addition). That is, c_i subsequent additions amount to a single draw from $\mathcal{N}(0, c_i\eta)$. The is possible since we only need to know the value of u_i, v_j whenever we encounter a new triple.

Table Lookup: Drawing iid samples from a Gaussian is quite costly, easily dominating all other floating point operations combined. We address this by pre-generating a large table of numbers [19] and then by performing random lookup within the table. More to the point,

a lookup table of r random numbers is statistically indistinguishable from the truth until we draw $O(r^2)$ samples from it (this follows from the slow rate of convergence for two-sample tests), hence a few MB of data suffice. Finally, for cache efficiency, we read contiguous segments with random offset (this adds a small amount of dependence which is easily addressed by using a larger table).

A cautionary note is that the impact of this approach on privacy, namely how it affects the stationary distribution of the SGLD, is unknown. For readers who are interested in the impact of lookup table size vs. accuracy please refer to [17].

6. EXPERIMENTS AND DISCUSSION

We now investigate the efficiency and accuracy of our fast SGD solver and Stochastic Gradient Langevin Dynamics solver, compared with state-of-the-art available recommenders. We explore the differentially private accuracy by using our proposed method while varying different privacy budgets.

6.1 Comparisons

We compare the performance of both the SGD solver and the SGLD solver to other publicly available recommenders and one closed-source solver. In particular, we compare to both CPU and GPU solvers, since the latter tend to excel in massively parallel floating point operations.

GraphChi Most of our experiments focus on a direct comparison to GraphChi [15]. This is primarily due the fact that the code for GraphChi is publicly available as open source and its very good performance.

GraphLab Create is a closed source data analysis platform [18]. It is currently the fastest recommender system available, being slightly faster than GraphChi. We compared our system to GraphLab Create, albeit without fine-grained diagnostics that were possible for GraphChi.

BidMach is a GPU based system [38]. It reports runtimes of 90, 129 and 600 seconds respectively for 100, 200 and 500 dimensions using an Amazon g2.2xlarge instance for the Netflix dataset.[2] This is slower than the runtimes of 48, 63, and 83 seconds for 128, 256, and 512 that we achieve without GPU optimization on a c3.8xlarge instance.

Spark is a distributed system (Spark MLlib) for inferring recommendations and factorization. In recent comparison the argument has been made that it is somewhat slower[3] than GraphLab while being substantially faster than Mahout.

6.2 Data

We use two datasets — the well known Netflix Prize dataset, consisting of a training set of 99M ratings spanning 480k customers and their ratings on almost 18k, each movie being rated at a scale of 1 to 5 stars. Additionally, we use their released validation set which consists of 1.4M ratings for validation purposes.

Secondly, we use the Yahoo music recommender dataset, consisting of almost 263M ratings of 635k music items by 1M users. We also use the released validation set which consists of 6M ratings for validation. We re-scale each rating at a scale of 0 to 5.

[2] http://github.com/BIDData/BIDMach/wiki/Benchmarks

[3] http://stanford.edu/~rezab/sparkworkshop/slides/xiangrui.pdf, Slide 31

K	SC-SGD		GraphChi	
	L1 Cache	L3 Cache	L1 Cache	L3 Cache
16	2.84%	0.43%	12.77%	2.21%
256	2.85%	0.50%	12.89%	2.34%
2048	3.3%	1.7%	15%	9.8%

Table 1: Cache miss rates in C-SGD and GraphChi.

6.3 Runtime

We run all the experiments on an Amazon c3.8xlarge instance running Ubuntu 14.04 with 32 CPUs and 60GB RAM.

For SGD-based methods we grid search the best parameters [17] for initial learning rate η_0, decay rate γ and regularizer λ [17]. For our fast SGLD solver, in addition to previous paramters, practically we multiply learning rate by a temperature parameter ζ [4] in the Gaussian noise $\mathcal{N}(0, \zeta \cdot \eta_t)$ with $\sqrt{\zeta \cdot \eta_t} \gg \eta_t$ to speed up SGLD's burn-in procedure.

Since it is nontrivial to observe the test RMSE error in each epoch when using Graphlab Create, we only report the timing of Graphlab Create and all other methods in Figure 3. Note that we were unable to obtain performance results from BidMach for the Yahoo dataset, since Scala encountered memory management issues. However, we have no reason to believe that the results would be in any way more favorable to BidMach than the findings on the Netflix dataset. For reproducibility the results were carried out on an AWS `g2.8xlarge` instance.

To illustrate the convergence over time. We run all the methods in a fixed number of epochs. That is 15 epochs and 30 epochs respectively because we observe that our SGD solver can reach the convergence at that time. Figure 2 shows our timing results along with convergence while we vary dimensions of the models.

Both of our solvers, i.e. C-SGD and Fast SGLD benefit from our caching algorithm. C-SGD is around 2 to 3 times faster than GraphChi and Graphlab while simultaneously outperforming the accuracy of GraphChi.

Note that the algorithm required for Fast SGLD is rather more complex, since it performs sampling from the Bayesian posterior. Consequently, it is slower than plain SGD. Nonetheless, its speed is comparable to GraphChi in terms of throughput (despite the latter solving a much simpler problem). One problem of SGLD is that the more complex the models are, the worse its convergence becomes (even though generates good samples on small dimensions Figure 4), due to the fact that we are sampling from a large state space. This is possibly due to the slow mixing of SGLD, which is a known problem of SGLD [2]. Improving the mixing rate by considering a more advanced stochastic differential equation based sampler, e.g. [4, 6], while keeping the cache efficiency during the updates will be important future works. To our best knowledge we are the first to report the convergence results of SGLD at this scale.

6.4 Cache-efficient Design

We show the cache efficiency of C-SGD and Graphchi in this section. Our data access pattern can accelerate the hardward cache prefetching. In the meanwhile we also use software prefetching strategies (with prefetching stride set to 2) to prefetch movie factors in advance.

We set the experiments as follows. In each gradient update step given r_{ij}, once the parameters e.g. u_i and v_j in (3) been read they will stay in cache for a while until they be flushed away by new parameters. What we really care about in this section is if the first time each parameter be read by CPU is already staying in cache or not. If it is not in cache then there will be a cache miss and will push CPU to idle. We use Cachegrind [16] as a cache profiler and analyze

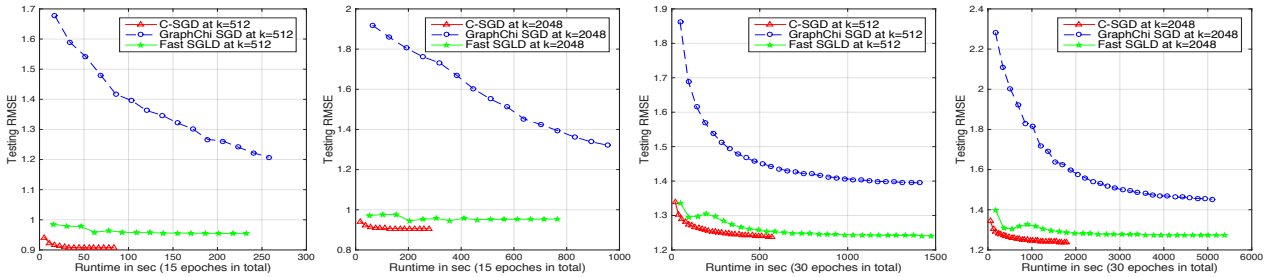

Figure 2: Runtime comparisons of the C-SGD solver, differentially private SGLD solver vs. non-private GraphChi/Graphlab on a identical Amazon AWS c3.8xlarge instance. (Left: Netflix, Right: Yahoo).

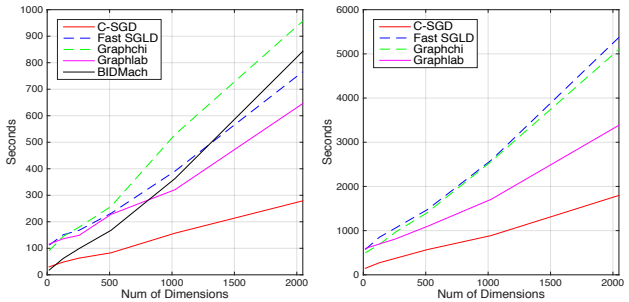

Figure 3: Timing comparisons on Netflix (left, 15 epochs) and Yahoo (right, 30 epochs).

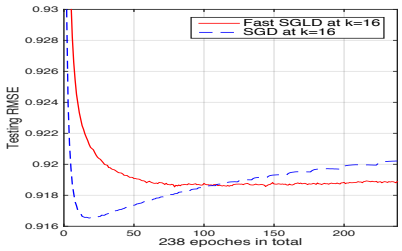

Figure 4: Convergence of SGLD for 16-dimensional models on Netflix.

cache miss (see Table 1 which shows it plays crucial effects on the final performance) for this purpose.

6.5 Privacy and Accuracy

We now investigate the influence of privacy loss on accuracy. As discussed previously, a small rescaling factor B can help us to get a nice bound on the loss function. For private collaborative filtering purposes, we first trim the training data by setting each user's maximum allowable number of ratings $\tau = 100$ and $\tau = 200$ for the Netflix competition dataset and Yahoo Music data respectively. We set $B = \tau(5 - 1 + \kappa)^2$ and weight of each user as $w_i = \min(\rho, \frac{B}{m_i(5-1+\kappa)^2})$ where κ is set to 1. According to different trimming strength we have $B = 2500$ and $B = 5000$ for Netflix data and Yahoo data respectively. As such we get a dataset with 33M ratings for Netflix and 100M ratings for Yahoo Music data. We study the prediction accuracy, i.e. the utility of our private method by varying the differential privacy budget ϵ for fixed model dimensionality $K = 16$. The parameters of the experiment are set as in [17].

While we are sampling (U, V) jointly, we essentially only need to release V. Users can then apply their own data to

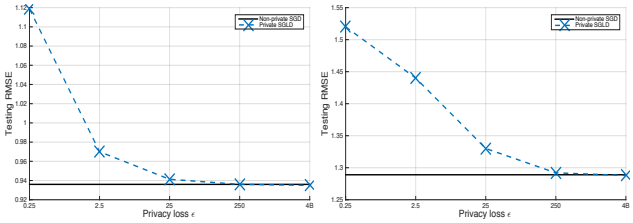

Figure 5: Test RMSE vs. privacy loss ϵ on Netflix (left) and Yahoo (right). A modest decrease in accuracy affords a useful gain in privacy.

get the full model and have a local recommender system:

$$u_i \approx \left(\lambda \mathbf{1} + \sum_{j|(i,j)\in\mathcal{S}} v_j v_j^\top\right)^{-1} \sum_j v_j r_{ij} \qquad (5)$$

The local predictions, i.e. in our context the utility of differentially private matrix factorization method, along with the different privacy loss ϵ are shown in Figure 5.

More specifically, the model (5) is a *two-stage* procedure which first takes the differentially private *item vectors* and then use the latter to obtain locally non-private user parameter estimates. This is perfectly admissible since users have no expectation of privacy with regard to their own ratings.

6.6 Rating privacy, user privacy and average personalized privacy

Interpreting the privacy guarantees can be subtle. A privacy loss of $\epsilon = 250$ as in Figure 5 may seem completely meaningless by Definition 1 and the corresponding results in Mcsherry and Mironov [20] may appear much better.

We first address the comparison to Mcsherry and Mironov [20]. It is important to point out that our privacy loss ϵ is stated in terms of user level privacy while the results in Mcsherry and Mironov [20] are stated in terms of rating level privacy, which offers exponentially weaker protection. ϵ-user differential privacy translates into ϵ/τ-rating differential privacy. Since $\tau = 200$ in our case, our results suggest that we almost lose no accuracy at all while preserving rating differential privacy with $\epsilon < 1$. This matches (and slightly improves) Mcsherry and Mironov [20]'s carefully engineered system.

On the other hand, we note that the plain privacy loss can be a very deceiving measure of its practical level of protection. Definition 1 protects privacy of an arbitrary user, who can be a malicious spammer that rates every movie in a completely opposite fashion as what the learned model would predict. This is a truly paranoid requirement, and arguably not the right one, since we probably should not protect these malicious users to begin with. For an average

user, the personalized privacy (Definition 2) guarantee can be much stronger, as the posterior distribution concentrates around models that predict reasonably well for such users. As a result, the log-likelihood associated with these users will be bounded by a much smaller number with high probability. In the example shown in Figure 5, a typical user's personal privacy loss is about $\epsilon/25$, which helps to reduce the essential privacy loss to a meaningful range.

7. CONCLUSION

In this paper we described an algorithm for efficient collaborative filtering that is compatible with differential privacy. In particular, we showed that it is possible to accomplish all three goals: accuracy, speed and privacy without any significant sacrifice on either end.

Moreover, we introduced the notion of *personalized* differential privacy. That is, we defined (and proved) the notion of obtaining estimates that respect different degrees of privacy, as required by individual users. We believe that this notion is highly relevant in today's information economy where the expectation of privacy may be tempered by, e.g. the cost of the service, the quality of the hardware (cheap netbooks deployed with Windows 8.1 with Bing), and the extent to which we want to incorporate the opinions of users.

Our implementation takes advantage of the caching properties of modern microprocessors. By careful latency hiding we are able to obtain near peak performance. In particular, our implementation is approximately 3 times as fast as GraphChi, the next-fastest recommender system. In sum, this is a strong endorsement of Stochastic Gradient Langevin Dynamics to obtain differentially private estimates in recommender systems while still preserving good utility.

Acknowledgments: Parts of this work were supported by a grant of Adobe Research. Z. Liu was supported by Sience Fund for Creative Research Groups (61221063); Ministry of Education Innovation Research Team (IRT13035); NSF of China (91118005, 91218301, 61428206). Y.-X. Wang was supported by NSF Award BCS-0941518 to CMU Statistics and Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

References

[1] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. Smola. Scalable Inference in Latent Variable Models. In *WSDM*, 2012.

[2] S. Ahn, A. Korattikara, and M. Welling. Bayesian posterior sampling via stochastic gradient fisher scoring. In *ICML'12*, pages 1591–1598, 2012.

[3] S. Ahn, A. Korattikara, N. Liu, S. Rajan, and M. Welling. Large scale distributed bayesian matrix factorization using stochastic gradient MCMC. 2015.

[4] T. Chen, E. B. Fox, and C. Guestrin. Stochastic Gradient Hamiltonian Monte Carlo. 32, 2014.

[5] C. Dimitrakakis, B. Nelson, A. Mitrokotsa, and B. I. Rubinstein. Robust and private bayesian inference. In *Algorithmic Learning Theory*, pages 291–305. Springer, 2014.

[6] N. Ding, C. Chen, R. D. Skeel, and R. Babbush. Bayesian Sampling Using Stochastic Gradient Thermostats. In *NIPS*, pages 1–14, 2014.

[7] C. Dwork. Differential privacy. In *Automata, Languages and Programming*, pages 1–12. Springer, 2006.

[8] C. Dwork and A. Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2013.

[9] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography*, pages 265–284. Springer, 2006.

[10] H. Ebadi, D. Sands, and G. Schneider. Differential privacy: Now it's getting personal. In *ACM Symposium on Principles of Programming Languages*, pages 69–81. ACM, 2015.

[11] A. Hartstein, V. Srinivasan, T. Puzak, and P. Emma. On the nature of cache miss behavior: Is it $\sqrt{2}$? *The Journal of Instruction-Level Parallelism*, 10:1–22, 2008.

[12] R. Keshavan, A. Montanari, and S. Oh. Matrix completion from noisy entries. In *NIPS'09*, pages 952–960, 2009.

[13] Y. Koren. Collaborative Filtering with Temporal Dynamics. In *KDD*, number 4, 2009.

[14] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer Society*, pages 42–49, 2009.

[15] A. Kyrola, G. Blelloch, and C. Guestrin. GraphChi : Large-Scale Graph Computation on Just a PC Disk-based Graph Computation. In *OSDI*, 2012.

[16] N. P. Laptev. Analysis of cache architectures. *Department of Computer Science–University of California Santa Barbara*.

[17] Z. Liu, Y.-X. Wang, and A. J. Smola. Fast differentially private matrix factorization. *arXiv:1505.01419*, 2015.

[18] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein. Graphlab: A new framework for parallel machine learning. *arXiv:1408.2041*, 2014.

[19] G. Marsaglia, W. W. Tsang, and J. Wang. Fast Generation of Discrete Random Variables. *Journal of Statistical Software*, 11, 2004.

[20] F. Mcsherry and I. Mironov. Differentially Private Recommender Systems : Building Privacy into the Netflix Prize Contenders. In *KDD*, 2009. ISBN 9781605584959.

[21] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 94–103. IEEE, 2007.

[22] R. Meka, P. Jain, and I. S. Dhillon. Matrix completion from power-law distributed samples. In *NIPS*, 2009.

[23] D. J. Mir. *Differential privacy: an exploration of the privacy-utility landscape*. PhD thesis, Rutgers University-Graduate School-New Brunswick, 2013.

[24] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology (TOIT)*, 7(4):23, 2007.

[25] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE, 2008.

[26] R. M. Neal. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2, 2011.

[27] F. Niu, B. Recht, R. Christopher, and S. J. Wright. Hogwild ! : A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *NIPS*, pages 1–22, 2011.

[28] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann Machines for Collaborative Filtering. In *ICML*, 2007.

[29] I. Sato and H. Nakagawa. Approximation analysis of stochastic gradient langevin dynamics by using fokker-planck equation and ito process. In *ICML'14*, pages 982–990, 2014.

[30] N. Srebro and R. R. Salakhutdinov. Collaborative filtering in a non-uniform world: Learning with the weighted trace norm. In *NIPS'10*, pages 2056–2064, 2010.

[31] Y. W. Teh, A. Thiéry, and S. Vollmer. Consistency and fluctuations for stochastic gradient langevin dynamics. *arXiv preprint:1409.0578*, 2014.

[32] Y. W. Teh, S. J. Vollmer, and K. C. Zygalakis. (Non-) asymptotic properties of Stochastic Gradient Langevin Dynamics. *arXiv preprint:1501.00438*, 2015.

[33] Y.-X. Wang and H. Xu. Stability of matrix factorization for collaborative filtering. In *ICML'12*, pages 417–424, 2012.

[34] Y.-X. Wang, S. E. Fienberg, and A. Smola. Privacy for free: Posterior sampling and stochastic gradient monte carlo. In *ICML*, 2015.

[35] M. Welling and Y. W. Teh. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *ICML*, 2011.

[36] Y. Xin and T. Jaakkola. Controlling privacy in recommender systems. In *NIPS*, 2014.

[37] A. Zhang, N. Fawaz, S. Ioannidis, and A. Montanari. Guess who rated this movie: Identifying users through subspace clustering. *arXiv preprint arXiv:1208.1544*, 2012.

[38] H. Zhao and J. F. Canny. *High Performance Machine Learning through Codesign and Rooflining*. PhD thesis, EECS Department, University of California, Berkeley, Sep 2014.