The Impact of Training Epoch Size on the Accuracy of Collaborative Filtering Models in GraphChi Utilizing a Multi-Cyclic Training Regimen

by

James W. Curnalia

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Computing and Information Systems

in the

Computer Science and Information Systems

Program

YOUNGSTOWN STATE UNIVERSITY

May, 2013

The Impact of Training Epoch Size on the Accuracy of Collaborative Filtering Models in GraphChi Utilizing a Multi-Cyclic Training Regimen

James W. Curnalia

I hereby release this thesis to the public. I understand that this thesis will be made available from the OhioLINK ETD Center and the Maag Library Circulation Desk for public access. I also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

Signature:

James W. Curnalia, Student Date

Approvals:

Alina Lazar, Thesis Advisor Date

Bonita Sharif, Committee Member Date

John Sullins, Committee Member Date

Bryan DePoy, Interim Dean of School of Graduate Studies and Research Date

Abstract

More and more outlets are utilizing collaborative filtering techniques to make sense of the sea of data generated by our hyper-connected world. How a collaborative filtering model is generated can be the difference between accurate or flawed predictions. This study is to determine the impact of a cyclical training regimen on the algorithms presented in the Collaborative Filtering Toolkit for GraphChi. Initial testing shows that some of the algorithms benefited from a multi-cyclic approach, a result that is reinforced by repeating the experiment on a separate dataset. Additional testing focuses on the effectiveness of dynamic versus fixed training cycle sizes. However, there is no additional benefit to adopting this more complex training scheme. While the results are far from universal, half of the algorithms saw a significant increase in accuracy when subjected to a multi-cyclic training regimen.

Table of Contents

Chapter 1: Introduction to Collaborative Filtering

With the growth of on-line services and e-commerce the amount of data that users need to make sense of has increased exponentially. The difficult task facing these service providers is to sift through a sea of options and find what a user wants before their competitors. One of the most powerful tools they have at their disposal is collaborative filtering.

One of the central assumptions about collaborative filtering is that people with similar tastes, or shopping histories, are better predictors of a user's future behavior than a random person. Collaborative filtering techniques search through large datasets to identify patterns and similarities between these users, and then make recommendations. These processes are not limited to the retail sphere, collaborative filtering has also been applied to financial, geological, and other endeavors. An examination of a small example of collaborative filtering, to produce song recommendations for customers, will help illustrate some of these ideas.

Table 1: Example User Song History and Ratings

User	song1	song2	song3	song4	song5	song6	song7
A	3	**X**	**X**	4	1	5	2
B	1	2	**X**	2	4	**X**	4
C	5	**X**	4	**X**	1	4	**X**
D	2	**X**	1	**X**	5	**X**	5

The example data covers four different customers (A-D) and their listening history over a catalog of seven songs. In addition to the fact that a user has listened to a particular song, there is an explicit rating (1-5, 5 being the highest). If a user has not listened to a song

this fact is highlighted with an 'X'. A predicted rating needs to be generated for these unknown songs, so that they can be suggested to the appropriate users.

The first step in the process is to identify which users have similar tastes. User A and User C have three points of similarity. They both responded very positively to song6 (ratings of A:5 and C:4) and very negatively to song5 (both had ratings of 1). The third point of similarity, song1, is less significant. User C responded very favorably to the song, while User A had a neutral reaction. A similar process can generate a taste profile for Users B and D (song1 B:1/D:2, song5 B:4/D:5, and song7 B:4/D:5). Using these pairings we can begin to identify likely reactions to some of the users' unknown songs.

Table 2: Suggestions Based on Similarities

User	**song1**	**song2**	**song3**	**song4**	**song5**	**song6**	**song7**
A	3	**X**	+	4	1	5	2
B	1	2	-	2	4	**X**	4
C	5	**X**	4	+	1	4	-
D	2	-	1	-	5	**X**	5

Six of the unknown songs have been replaced with a suggested reaction (+ for positive or – for negative). An exact score would require further analysis, and most likely a great deal more information, but a more simplistic measure of attitude can be inferred.

This still leaves four unknown songs in the table. A quick look over the table shows that the two pairs of users have completely opposite tastes in music. It would be reasonable to assume that what one pair likes the other will dislike, and vice versa. In this way we can utilize not only the similarity between users to determine attitudes, but also the reactions of diametrically opposed Users.

In Table 3 all of the unknown songs have been assigned a suggested user reactions, and a

number of recommendations have been identified for User A and User C. These songs will be suggested the next time the users visit the site. Now all that remains is to scale the entire process to handle millions of users and products, while maintaining accuracy and minimizing costs (both financial and temporal).

Table 3: Suggestions Based on Inverse Taste Profiles

User	song1	song2	song3	song4	song5	song6	song7
A	3	+	+	4	1	5	2
B	1	2	-	2	4	-	4
C	5	+	4	+	1	4	-
D	2	-	1	-	5	-	5

Chapter 2: Related Work

Collaborative Filtering has traditionally included methods such as Bayesian Networks and clustering. A Bayesian Network uses probability and causal relationships to classify new observations (Pearl 1994). Clustering algorithms attempt to represent observations as "points" in a multi-dimensional space. The closer together that two points are, the more similar the underlying observations (Witten, Frank and Hall 2011). These and other traditional methods are, and will continue to be, powerful and valid collaborative filtering methods.

However, in the wake of events like the Netflix Prize a new series of algorithms were developed to deal with a new phenomenon "Big Data". These new algorithms sought to incorporate the concepts of the traditional methods into new frameworks capable of dealing with data that was increasingly large, complex, and often extremely sparse.

This study will concern itself with thirteen of these modern Collaborative Filtering algorithms:

1. Alternating Least Squares (ALS), (Zhou et al. 2008)
2. Stochastic Gradient Descent (SGD), (Koren 2009)
3. Bias Stochastic Gradient Descent (BSGD), (Koren 2008)
4. Koren's Singular Value Decomposition (SVD++), (Koren 2008)
5. Weighted ALS (WALS), (Hu, Koren, and Volinsky 2008)
6. Non-Negative Matrix Factorization (NMF), (Lee and Seung 2001)
7. Singular Value Decomposition (SVD), (Hernandez, Roman, and Tomas 2007)
8. One-Sided SVD, (Hernandez, Roman, and Tomas 2007)

9. Tensor ALS (TALS), (Comon, Luciani, and de Almeida 2009)

10. Restricted Bolzman Machines (RBM), (Hinton 2010)

11. Time-SVD++(TSVD++), (Koren 2009)

12. libFM, (Rendle 2010)

13. Probabilistic Matrix Factorization (PMF), (Salakhutdinov and Mnih 2008)

Each of the modern Collaborative Filtering algorithms seeks to either reduce or capitalize on the complexity of large datasets. Algorithms utilizing decomposition, factorization, and SGD seek to reduce the dimensionality of data in order to expose underlying relationships. Least squares methods treat recommendations as linear equations, and attempt to find the best estimation of the parameters necessary to calculate an accurate rating. TALS and TSVD++ try to leverage additional information, in this case time, in order to more accurately model behaviors.

This study will focus on graph-based implementations of these algorithms given their recent popularity and the ability to be executed on smaller machines. Recently graph-based algorithms have been adopted by many large, commercial websites including Amazon and YouTube (Walia 2008). While it is important to note the adoption of techniques like this by powerful and influential corporations, it is admittedly the latter that was the driving force in adopting a graph-based approach.

The GraphLab Project was developed in order to facilitate distributed, parallel, graph-based algorithms in an efficient and reliable manner (Low et al. 2010). GraphChi is an offshoot of the GraphLab Project that seeks to leverage the power graph-based algorithms on a single machine, while maintaining high performance standards (Kyrola, Blelloch,

and Guestrin 2012). Bickson, one of the original developers of GraphLab, has ported a number of collaborative filtering algorithms from GraphLab to GraphChi in the form of the Collaborative Filtering Toolkit (CFT). Thirteen algorithms were supported at the time this study was run, December 2012, but two additional algorithms had already been added as this paper was being written, January 2013.

In addition to developing the toolkit, Bickson has written a blog entry which serves as an introduction to the CFT and its underlying algorithms (Bickson 2012). In the tutorial, Bickson identifies a number of algorithms which have an element of fault tolerance. These algorithms allow the user to save the model to disk and then resume training from that exact state.

Experimentation with the fault-tolerant algorithms seemed to yield an additional benefit, in that the accuracy of the model would often jump between executions of the training epochs (this paper will use the terms “epoch” and “cycle” interchangeably to refer to a group of training iterations). This would seem to suggest that there is an advantage to using multiple cycles beyond simple fault tolerance. The cumulative value of this inter-cycle bump in accuracy could be quite significant.

As these algorithms train a model, they attempt to reduce the sample space in an attempt to converge on an optimal answer. Given that these algorithms become more restrictive and focused the longer that they run, it is reasonable to assume that restarting an algorithm would have a significant positive impact on the final model. By loosening the bounds placed on the algorithm it is possible to identify the possibility that the current parameters have focused on a local rather than global minimum.

Chapter 3: Proposal

The observed increase in inter-epoch accuracy of models being trained with fault tolerant algorithms suggests that there is significant value to be gained from such a training regimen. This thesis will explore the possibilities of exploiting this behavior in three progressively more focused phases.

Phase 1:

First it will be necessary to establish whether there is in fact a boost in model accuracy between training cycles. Next, I will identify the cycle size that results in the best final model. A fault-tolerant algorithm, utilizing a epoch size of no more than 20, should result in a model that is significantly more accurate than those trained using a single-epoch algorithm.

Phase 2:

Having confirmed the existence of an inter-cycle boost in accuracy, it will be necessary to confirm the results on a separate dataset. The results of this experiment should mirror those of the first part. There will be slight variations in accuracy and running time, but the behaviors observed in the initial testing should present themselves.

Phase 3:

The use of a fixed number of iterations in an epoch is simple, but limits the effectiveness of the training regimen because large portions of later cycles are spent trapped in a local minima or overfitting. A dynamic epoch size should allow the user to reap the maximum benefit of a multi-cyclic training approach while eliminating the shortcomings of a fixed-cycle regimen.

Chapter 4: Common Terms and Measures of Accuracy

Local Minima:

A global minimum refers to the lowest value for the entire domain of a function. However it is possible to have a value that is the lowest for a particular neighborhood of a function. This value is referred to as local minima. Figure 1 is a graph of a function, both of the highlighted points represent minima for a particular neighborhood of values.

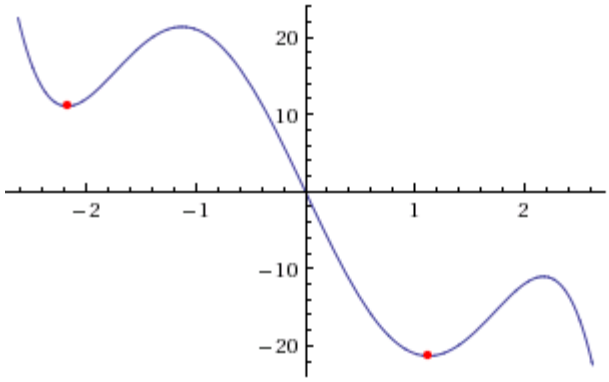


Figure 1: Local Minima (image generated on www.wolframalpha.com)

Matrix Factorization:

The process of reducing a matrix into its component factors, which are also matrices. When multiplied together these factors result in the original matrix. This decomposition often reveals relationships hidden in the original data.

Overfitting:

Generally there are two error scores generated while training a model, training and validation. Training error is a measure of how well the model performs using the exact data that was used to generate it. Validation error is how well the model performs on a subset of the data that was withheld from the training process. Overfitting occurs when

the training error continues to improve while the validation error stagnates or increases. The root problem in overfitting is often that the model has latched onto some quirk of the training set that doesn't represent an actual relationship in the data.

Root Mean Squared Error (RMSE):

In order to estimate the error of a model overall, a single number is generated. This number represents the average variance between a predicted value and the actual value. RMSE is calculated by squaring the difference between the predicted and real values, adding them all together, and dividing by the number of instances in the set. As a final step, the square root is taken to put RMSE into the same measure as the original values. This measure is very similar to standard deviation, and can be utilized in many of the same ways.

$$RMSE = \sqrt{\frac{\sum \left(Z_{real} - Z_{computed}\right)^2}{n}}$$

Image modified from www.gisdevelopment.net

Figure 2: RMSE Equation

Chapter 5: The Algorithms

At the time of this study there were a total of thirteen algorithms in the CFT. Of these thirteen algorithms, seven are capable of fault tolerance (ALS, WALS, TALS, NMF, SGD, Bias-SGD and SVD++).

ALS:

In ALS the entire dataset is reduced to two matrices (U:Users & M:Movies). Matrix M contains the average rating for each movie. Matrix U contains each users deviation from the average score. The first pass though the algorithm, the values in M are static while values in U are solved linearly to decrease the overall error. The second pass switches matrices, locking in U and solving for M. The product of the matrices is then taken and compared to the training set. Passes through the algorithm continue until there is convergence.

SGD:

SGD is an implementation of the FunkSVD algorithm (Funk 2006). The algorithm seeks to isolate the effect of a number of unknown features on user's ratings. The identity of these features does not need to be known or identified prior to running the algorithm. The general process is that for each of the features to be identified a generic value is assigned to the user and the movie. A rating is generated by taking the product of the current feature values for both the user and the movie, and all previously defined features. The error between the actual rating and the predicted rating, combined with a constant, is used to refine the feature values.

BSGD:

The BSGD algorithm integrates implicit feedback into the SGD algorithm. Implicit feedback seeks to identify the features that led a user to select a given item regardless of the explicit rating. Additionally, it is possible to trace the impact of individual items to identify items which led to the recommendation.

SVD++:

SVD++ is simply a refinement of the BSGD model, but utilizes the same basic concepts.

WALS:

WALS uses implicit feedback, represented by a binary rating, to generate matrices U and M. The impact of a given movie is then weighted based on its similarity to other movies the user has consumed.

NMF:

This algorithm performs matrix factorization with the additional requirement that all factors be positive.

SVD & OSVD:

Both algorithms decompose the initial user/movie matrix into two separate matrices, one for users and the other for movies. After the decomposition, a triplet is generated (u_i, σ_i, v_i). u_i and v_i are the singular vectors, while σ_i is the singular value. The product of this triplet is an estimation of the original rating. OSVD attempts to reduce the computational cost of the algorithm while maintaining accuracy.

TALS:

TALS uses tensor decomposition to reduce the chance that ALS will become stuck in a local minima and reduce the number of training iterations.

RBM:

The RBM is a neural network consisting of two layers. The first, visible layer corresponds to the features being used to train the network. The second, hidden layer is trained by the features in the visible layer. The "restricted" in RBM refers to the requirement that all connections must be between visible-hidden node pairs. A gradient descent algorithm is used to compute the weights between the nodes.

TSVD++:

TSVD++ adds a temporal element to the SVD++ algorithm, tracking the changes in a user/item popularity over time. The inclusion of time allows for the modeling of a user's changing attitudes over time. This model helps to identify how a particular user's attitudes change, and apply the new attitude to future recommendations.

libFM:

The libFM library is a particular implementation of matrix factorization that includes SGD and ALS with Markov Chain Monte Carlo (MCMC). MCMC is a sampling method useful when dealing with mutli-dimensional integrals, utilizing random walks.

PMF:

PMF is a matrix factorization model based on Bayesian probabilities, which is trained using MCMC methods and assumes that users and movies are dependent.

Chapter 6: The Data

The type of data used in GraphChi, or any collaborative filtering method, is immaterial. It is up to the researcher to provide the appropriate framework to make the data meaningful to the algorithms. This study will utilize the Netflix and MovieLens datasets based on their identical framework and common use in the field of collaborative filtering.

Section 6.1: Netflix

The first phase of this study uses the same dataset featured in Bickson's blog, which is a synthetic Netflix dataset created using an anonymized sample of the original. Although the dataset from the Netflix challenge is unavailable because of copyright, the general characteristics of the data are well established by the competition's creators (Bennett and Lanning 2007). The GraphLab Netflix sample was done to preserve these characteristics (i.e. sparsity of data, user to movie ratio, user to rating ratio, etc) while ensuring the anonymity of the users.

The Netflix subset has the following general characteristics: 95,526 unique users, 3,561 movies, and 3,298,163 ratings (non-zeroes). This is a very sparse dataset with less than 0.97% of the resulting matrix having ratings.

Section 6.2: MovieLens

The second phase of this study uses a similar dataset, the Million Rating MovieLens (Lam and Herlocker 2000), to identify if the results of Phase I are valid, or if they are the artifact of the Netflix dataset. The GraphLab team has converted this dataset into a training/testing pair appropriate for use with the CFT. The MovieLens dataset has the following general characteristics: 6,040 unique users, 3,952 movies, and 1,000,209

ratings (non-zeroes). This dataset has more than 23.55% of the resulting matrix having ratings, a far more dense sample.

Section 6.3: Training and Testing Sets

Both the Netflix and MovieLens datasets were divided, formatted, and hosted by the GraphLab team. These particular datasets were chosen because of their frequent use in collaborative filtering research. Their use will allow the results of this study to compared or applied to existing methodologies.

The methods used to divide the data into training and testing sets facilitated a few simple goals. The training sets were designed to maintain the same statistical properties of the original dataset, in order to maintain the integrity of the predictions. The testing sets contain at least a single instance of every user and movie in order to test the model's full range of predictive possibilities.

Section 6.4: Historic Benchmarks and Baseline Predictors

In order to provide some context in which to view the results of this study the following results from the Netflix Prize (Netflix 2009) were retrieved:

- Cinematch (2006) : RMSE 0.9525
- 2007 Progress Prize : "KorBell" : RMSE 0.8723
- 2008 Progress Prize : "BellKor in BigChaos" : RMSE 0.8627
- Winners : "BellKor's Pragmatic Chaos" (2009) : RMSE 0.8567

Additionally, KorBell reported that the best result they could get from a single method was an increase of 6.57% over Cinematch, RMSE ~0.8882 (Bell and Koren 2007).

Three baseline methods are also included in the Bickson tutorial. All of these methods

seek to reduce the CF process to a simple average that can quickly be applied to a set of unknown instances. The first simply assigns the user's mean rating to all unrated movies, regardless of any other considerations. The second method assigns the mean rating of the movie, if it hasn't been rated. The final method determines the average of every rating in the dataset, and uses that as the default rating.

The error rates for each of the baseline methods were:

Netflix:	**MovieLens:**
User's Mean Rating : RMSE 0.9728	User's Mean Rating : RMSE 1.02809
Movie's Mean Rating : RMSE 1.0005	Movie's Mean Rating : RMSE 1.11734
Global Mean Rating : RMSE 1.0781	Global Mean Rating : RMSE 0.97439

Section 6.5: Control Groups

One final piece of information is necessary to determine the effectiveness of a multi-cyclic training regimen, a control group. The control group was trained using a single-epoch consisting of a large number of iterations. The control groups were generated after Phase I in an attempt to limit the number of unnecessary training cycles, since large epochs require 8+ hours to run.

Waiting until after the completion of Phase I had the additional benefit of identifying a number of algorithms which did not need to included in the control group. All of the algorithms utilizing ALS already had runs which demonstrated that even moderately sized single iterations were outperformed by multi-cyclic regimens (as was noted in the Methodology section). This meant that it was only necessary to run control groups on SGD, BSGD, NMF, and SVD++.

Each of the control groups is trained using the same parameters utilized in Phase I, only

the progression of epoch size is altered. A starting size of 200 iterations was selected since it is within the bounds of each of the selected algorithms total number of iterations run from Phase I. Each succeeding epoch will be doubled in size up to 1,600 iterations. The next epoch will be increased to 2,000 and then incremented by 1,000 every epoch after that. Training will continue until the RMSE no longer decreases between runs, and actually begins to increase.

Table 4: Netflix Control Group Results

Algorithm	**# of Iterations**	**RMSE**	**Running Time (sec)**
SGD	200	1.123820	274.217
BSGD	400	1.117690	727.475
SVD++	400	0.982024	1025.760
NMF	2000	2.370640	5549.790

Table 5: MovieLens Control Group Results

Algorithm	**# of Iterations**	**RMSE**	**Running Time (sec)**
SGD	200	1.111930	59.2760
BSGD	200	1.388990	58.0992
SVD++	200	0.933052	138.1750

Chapter 7: The Output

A. B. C. D. E.

```
0.247825) Iteration:   0 Training RMSE:    1.55184  Validation RMSE:     1.4803 ratings_per_sec:          0
0.551404) Iteration:   1 Training RMSE:    1.43175  Validation RMSE:    1.38094 ratings_per_sec: 1.45679e+06
0.843033) Iteration:   2 Training RMSE:    1.34697  Validation RMSE:    1.31508 ratings_per_sec: 1.97772e+06
  1.1353) Iteration:   3 Training RMSE:    1.29099  Validation RMSE:    1.27127 ratings_per_sec: 2.24475e+06
 1.42998) Iteration:   4 Training RMSE:     1.2528  Validation RMSE:    1.24062 ratings_per_sec: 2.40533e+06
 1.72584) Iteration:   5 Training RMSE:     1.2254  Validation RMSE:    1.21809 ratings_per_sec: 2.51095e+06
  2.0265) Iteration:   6 Training RMSE:    1.20487  Validation RMSE:    1.20091 ratings_per_sec: 2.58013e+06
 2.32415) Iteration:   7 Training RMSE:    1.18894  Validation RMSE:    1.18741 ratings_per_sec: 2.63517e+06
 2.61634) Iteration:   8 Training RMSE:    1.17624  Validation RMSE:    1.17654 ratings_per_sec: 2.68354e+06
                                                                                .
                                                                                .
                                                                                .
 21.0371) Iteration:  72 Training RMSE:    1.10307  Validation RMSE:    1.11021 ratings_per_sec: 3.07028e+06
 21.3199) Iteration:  73 Training RMSE:    1.10306  Validation RMSE:    1.11021 ratings_per_sec: 3.07214e+06
 21.5915) Iteration:  74 Training RMSE:    1.10306  Validation RMSE:     1.1102 ratings_per_sec: 3.07491e+06
 21.8678) Iteration:  75 Training RMSE:    1.10305  Validation RMSE:     1.1102 ratings_per_sec: 3.07751e+06
 22.1428) Iteration:  76 Training RMSE:    1.10305  Validation RMSE:    1.11019 ratings_per_sec: 3.07977e+06
  22.422) Iteration:  77 Training RMSE:    1.10305  Validation RMSE:    1.11019 ratings_per_sec: 3.08168e+06
  22.701) Iteration:  78 Training RMSE:    1.10304  Validation RMSE:    1.11019 ratings_per_sec: 3.0832e+06
 22.9786) Iteration:  79 Training RMSE:    1.10304  Validation RMSE:    1.11018 ratings_per_sec: 3.08536e+06
```

Figure 3: Sample Output from the CFT

Above is a selection of output from one of the training cycles. It is useful as an illustration of what training a model in CFT looks like, and to talk about some of the common behaviors demonstrated by many of the algorithms. In addition to output similar to this selection, each training cycle also outputs to the screen a list of all the environmental variables and arguments used to train the model.

There are six pieces of information displayed for each iteration of a training epoch, labeled A-E. A is a simple timer that shows how many seconds have elapsed between the beginning of the cycle and the end of the current iteration. B is the number of the iteration

in the sequence, starting with 0. C is the training RMSE, which is the ability to correctly predict the ratings of the movies which were used to train the model. D is the validation RMSE, this is a measure of how accurately the model predicts the rating of movies outside of the training set. Finally, E is the number of ratings generated per second by the current iteration.

Figure 3 is a good example of how most of the algorithms perform for any given training cycle. At the start of the cycle both training and validation RMSE improve dramatically every iteration. As the training continues the rate of improvement slowly wanes until even the earliest iterations see little or no improvement. As the cycle draws to a close, training and validation RMSE only improve every couple iterations. At this point, the parameters under which the algorithm is operating under have become so restrictive that large improvements are no longer possible. Later epochs will experience this type of stagnation at earlier iterations as the algorithm reaches the limits of its effectiveness.

Most of the algorithms perform and display in the manner described above, but there are exceptions. SVD and OSVD report the error for each of the factors identified, but no intermediary measures. NMF does not make use of a validation set, and so only reports the training RMSE. NMF also has a much more consistent rate of improvement, even for early iterations. It would seem that NMF is the one fault tolerant algorithm that has no benefit to restarting the training cycle.

The behavior identified in the sample output, the initial surge in accuracy at the beginning of an epoch, is the factor that should lead to the multi-cyclic training regimen delivering a more effective model.

Chapter 8: Phase I

Section 8.1: Methodology

Since this study is seeking to understand the impact of epoch size on the accuracy of the resulting model, there will be no attempt to tune any of the algorithms. The default settings were used in order to limit the impact of user proficiency on the resulting models (Bickson 2012).

The CFT reports the accuracy of the model generated using root mean squared error (RMSE). This statistic is generated for every complete pass over the training set, and is reported in terms of both training and validation. The training RMSE will not be used in this study as it is only a reflection of how well the model performs on the training data. Instead the validation RMSE will be the only measure reported since it demonstrates how well the model handled the test data. Additionally, the validation RMSE will identify problems with the model such as overfitting which are ignored in the training algorithm.

Fault Tolerance Algorithms (ALS, WALS, TALS, NMF, SGD, BSGD and SVD++):

The initial epoch size will be set to the maximum iterations specified in the tutorial, usually 6 iterations. After the first training cycle, the argument `--load_factors_from_file=1` will be added to the algorithm to resume training with the current state. Training cycles will continue until one of the following three conditions are met:

1. The validation RMSE no longer improves with subsequent training cycles (a minima is reached).
2. The validation RMSE increases with additional training cycles (overfitting).

3. Multiple training cycles result in an improvement of the validation RMSE of less than .00005 / 10 iterations (diminishing returns).

Upon reaching one of the above criteria, the current state of the model will be saved and the validation RMSE and total number of training cycles recorded.

Next the epoch size will be increased and the entire process will repeat for the new model. The training epoch will be increased on the following schedule: 6, 20, 40, 80, 100, 120, 140, 180, 200. After 200 iterations the size of the epoch will be incremented by 50 for every subsequent increase. The training of new models on this schedule will continue until the resulting model has a higher validation RMSE than the previously generated model.

After the complete training of an algorithm is completed, an average starting RMSE is selected and recorded as well. The starting RMSE is defined as the validation RMSE after a single iteration of the algorithm. Since there is a degree of variation inherent in all of these algorithms it is necessary to choose a representative initial state. The starting RMSE will allow for a model's training regimen to be judged both by its final accuracy and the degree of accuracy that is a result of training.

Remaining Algorithms (SVD, One-Sided SVD, RBM, TSVD++, libFM, and PMF):

Even though these algorithms do not support a multi-cyclic training regimen, they will be trained on the same epoch schedule to provide additional context.

Additional Note on Alternating Least Squares (ALS) Algorithms:

Overfitting was a significant problem with the ALS algorithms, and as such some modifications to the methods were made for those algorithms. Instead of stopping the training schedule with the first model resulting in a higher RMSE, all schedules were run

to at least a training epoch of 80 iterations in size. This was an attempt to see if overfitting could be overcome with additional training. Overfitting also led to the inclusion of two smaller epoch sizes on the training schedule, two and ten, in order to identify if the optimal size was located at this smaller scale.

Section 8.2: Results

Hypothesis:

The initial impressions of the CFT suggested that the apparent boost in accuracy between training epochs would favor a training regimen consisting of a large number of very small cycles (no more than 20 iterations per epoch). This type of training would lead to results superior, to those generated without it.

*Table 6: Algorithms Ordered by Final RMSE (Multi-Cyclic Algorithms **Bolded**)*

Algorithm	**Initial RMSE**	**Final RMSE**
PMF	2.498400	0.914566
RBM	0.979169	0.926279
SVD++	**1.124420**	**0.931921**
BSGD	**1.363540**	**0.952970**
SGD	**1.240700**	**0.959890**
TSVD++	1.041220	0.995435
LibFM	1.090030	1.025770
TALS	**1.244250**	**1.147030**
ALS	**1.251550**	**1.159920**

NMF	**1.580120**	**2.375430**
WALS	**5.522080**	**5.325280**

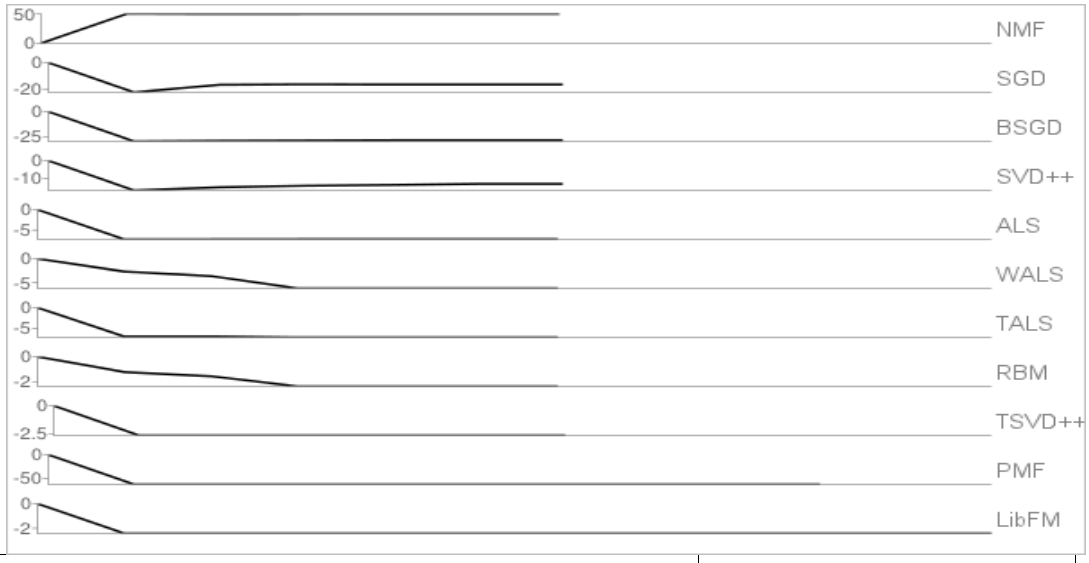
The first thing of note about Table 6 is that only eleven algorithms are included. Both SVD and One-Sided SVD have been left off of the table intentionally. No modification to the number of training iterations yielded any variation in how these algorithms performed. Every run of these methods resulted in both an identical process and model. Additionally, these algorithms utilize a different error metric than the rest of the CFT by reporting an error estimate for each of the features generated. For these reasons these algorithms were left out of the rest of the discussion of this study's results, but an example of the output of each has been included at the end of this paper as Appendix 1.

Looking at the results purely in terms of accuracy suggest that the fault-tolerant algorithms are generally of inferior quality. However, this view of the results is misleading. There was no effort made to tune these algorithms, or to even check if there current settings were conducive to producing good models, before these results were generated. So while it is interesting which algorithms handled the data best, it doesn't really show how well these models developed over the course of training.

Table 7: Results Ordered by Percent Improved Over Initial RMSE

Algorithm	**% Improvement**
PMF	63.39%
BSGD	**30.11%**

SGD	**22.63%**
SVD++	**17.12%**
TALS	**7.81%**
ALS	**7.32%**



LibFM	5.90%
RBM	5.40%
TSVD++	4.40%
WALS	**3.56%**
NMF	**-50.33%**

By ordering the results by how much a model's RMSE was improved over the course of training reveals a far different picture of the fault-tolerant algorithms. The improvement in final RMSE would suggest that these algorithms are more effective at training, but it is

unclear whether this is a product of the algorithm itself or the training regimen. A closer look at all of the results, as well as the effect of the restart boost, should provide a clearer picture of the factors at work.

F*igure 4: Trendlines of Algorithm Performance*

Figure 4 shows the percentage difference between all of the training regimens for a given algorithm and its starting RMSE. The trendlines clearly show that while the effectiveness of the algorithms may vary, their training behaviors are very similar. From this limited sample it would seem that there is no evidence that the restart boost creates a more effective training regimen. But this is not evidence that it has no effect.

*Table 8: Epoch Size and Number of Cycles Trained for Each Algorithm (Multi-Cyclic Algorithms **Bolded**)*

Algorithm	Epoch Size (Iterations)	Training Cycles	Total Running Time (sec)
PMF	180	-	2,501.9900
RBM	80	-	692.3140
SVD++	**40**	**7**	**691.6511**
BSGD	**40**	**92**	**6,860.8908**
SGD	**40**	**59**	**3127.5310**
TSVD++	80	-	251.9750

LibFM	200	-	908.0320
TALS	**10**	**1**	**74.6055**
ALS	**2**	**4**	**47.4264**
NMF	**40**	**56**	**9,045.0080**
WALS	**10**	**1**	**66.1071**

The idea that a cyclical training regimen is superior to a single epoch algorithm has been thoroughly disproved, with the exceptional performance of both PMF and RBM. But what about the ideal size for these epochs? Table 8 shows, rather conclusively, that the ideal number of iterations is larger than the 20 iteration ceiling that had been theorized. The problem of overfitting was again to blame. Each cycle needs to be large enough to take advantage of as many positive iterations (those that reduce the validation RMSE), while minimizing the number of overfitted iterations (a common occurrence in later training cycles).

While the cyclical training regimen fails to outperform the single epoch algorithms, it is clearly not without its benefits. Three of the cyclically generated models have significantly higher accuracy than similar models learned over the course of a lone epoch. All of the algorithms in the CFT suffer from the same design flaw, they fail to take into account validation RMSE. This results in either overfitting or the algorithm becoming trapped in a local minima, as the parameters of the algorithms become more and more restrictive. Restarting the algorithm loosens the bounds on the program allowing it to move beyond erroneous assumptions about the data. So while the hypothesized accuracy

failed to materialize, there are definitely significant advantages to this style of training with SGD, BSGD, and SVD++.

Table 9: Final RMSE of Control and Test Models (Phase I)

Algorithm	**Control RMSE**	**Test RMSE**	**Improvement**
SGD	1.123820	0.959890	14.59%
BSGD	1.117690	0.952970	14.74%
SVD++	0.982024	0.931921	5.10%
ALS	1.159920	1.161300	0.11%
TALS	1.147030	1.147030	0.00%
WALS	5.325280	5.325280	0.00%
NMF	2.370640	2.375430	-0.20%

If the results seen in this initial experiment are not a fluke, there should be some gain in removing the hindrance of a fixed cycle size. So the first step is to recreate this experiment with another dataset. If the results of this second run confirm the behaviors observed in the initial data, it should be possible to generate a more accurate model by loosening the strict epoch size requirements.

Chapter 9: Phase II

Section 9.1: Methodology

This phase is meant to verify the results of Phase I with a different dataset, the 1M MovieLens. Given this emphasis, the number of training epochs will be severely truncated. Each algorithm will be run using the regimen identified in Phase I. The second model will be trained with the the final step from Phase I.

At this point the training will continue in one of two directions. If the RMSE decreases for the second run, epoch size will continue to increase on the schedule outlined in Phase I. If however, the RMSE increases it will be necessary to run at least one of the earlier training regimens to ensure that the optimal model isn't generated earlier on the training schedule.

Some of the algorithms in the CFT rely on modified datasets. LibFM, TALS, TSVD++, and WALS all require additional time stamp data that is not included in this version of the MovieLens dataset. Additionally, NMF requires a version of the dataset where positions of the user and movie data are switched. Since this phase designed only to verify the results found in Phase 1, there will be no attempt to engineer these datasets. Instead these algorithms will be left out of this phase. A more complete investigation of this dataset would be interesting, but is beyond the scope of this study.

Section 9.2: Results

Hypothesis:

The results of generating models on a different dataset will mirror the results found in Phase I.

*Table 10: Algorithms Ordered by Final RMSE (Multi-Cyclic Algorithms **Bolded**)*

Algorithm	Initial RMSE	Final RMSE	Epoch Size	Cycles	% Improved	Running Time
PMF	2.456660	0.848362	250	-	65.47%	543.3890
SVD++	**1.130650**	**0.881066**	**40**	**9**	**22.07%**	**220.6638**
RBM	0.951297	0.892780	100	-	6.15%	174.1320
SGD	**1.480210**	**0.896868**	**40**	**118**	**39.41%**	**1367.3722**
BSGD	**1.800190**	**0.900937**	**100**	**115**	**49.95%**	**3306.0545**
ALS	**1.018160**	**0.965357**	**10**	**1**	**5.19%**	**13.0477**

For the most part, the results of Phase 2 are similar to the results found in Phase 1. SVD++ performs slightly better with the MovieLens data, managing to eek out a slightly better result the RBM. In the same way SGD and BSGD have switched places. However, aside from some reordering of how the individual algorithms performed, there was little difference between how the training process itself behaved.

The best way to see the similarities between Phases 1 and 2 is to focus on how effectively the algorithms trained their final models. Each of the algorithms had a percentage of improvement that was in keeping with the results seen in Phase 1. Most saw a modest increase in improvement, but this is in keeping with the fact that the MovieLens dataset is far more dense than Netflix and should therefore provide more concrete relationships.

Table 11 verifies that the models trained using a multi-cyclic approach are far more accurate than single-epoch models developed by the same algorithms. SGD and SVD++ show very similar results to those generated, in the first phase. BSGD seems to have

been far more effective, but this may be an artifact of its unusually high starting RMSE. ALS has once again shown that least squares algorithms benefit very little from a multi-cyclic approach. Table 10 already showed that the overall effectiveness of the training regimen was similar to the previous experiment, and this is more a testament to how limited a single-epoch training regimen is for these algorithms.

Table 11: Final RMSE of Control and Test Models (Phase I)

Algorithm	**Control RMSE**	**Test RMSE**	**Improvement**
SGD	1.111930	0.896868	19.34%
BSGD	1.388990	0.900937	35.14%
SVD++	0.933052	0.881066	5.57%
ALS	0.965357	0.965357	0.00%

Phase 2 has confirmed that the behaviors identified in Phase 1 are inherent to the algorithms themselves, and not the product some quirk in the Netflix dataset. By instituting a more intelligent and adaptive training regimen it should be possible to exploit these behaviors to develop more accurate models.

Chapter 10: Phase III

Section 10.1: Methodology

Based on the previous testing, using a fixed epoch size should limit the effectiveness of a model due to the inherent flaws of the CFT. If the number of iterations is set too low, early training cycles fail to take maximum advantage of unrestrictive parameters in order to more effectively position the model. If the epoch size is too large, later training cycles fritter away gains with cycles that overfit the model for up to 75% of the iterations.

What is needed is a training regimen that leverages the benefits of large epochs in early cycles, but doesn't suffer from rampant overfitting in later cycles. To this end epochs will be decremented every training cycle that results in either overfitting or becoming stuck in a local minima. The specific starting size and decremental schedule will vary for each algorithm in order to fit the training regimen to the individual needs of the model. The initial epoch size will be derived from the results of the control group training, since it represents a wide view of how the algorithms perform over large cycles, as shown in Table 12.

Table 12: Starting Epoch Size for Decremental Training

Algorithm	Initial Epoch
SGD	100
BSGD	100
SVD++	120
NMF	2000/100

ALS	8
TALS	10
WALS	6

The initial epoch sizes are drawn from the results of the control group generation. The number of iterations was selected to maximize the number of positive training runs, minimize initial training stagnation, and prevent any overfitting in the first cycle. This should maximize the effectiveness of early cycles, and prevent the need for radical iteration reductions.

After the initial training cycle, epoch size will be decremented as needed for each algorithm. The signal to reduce the number of iterations will be one of two conditions, overfitting or stagnation. The triggering event will be identified by the iteration it occurs in, referred to as n. Any cycle resulting in overfitting will immediately have the next epoch reduced to a maximum of n-1 iterations. Stagnation will be a bit more flexible in its definition.

There will be two different triggers based on the model becoming trapped in a local minima, each represented by a separate training regimen. The first is based on the assumption that little harm is done to the model if the validation RMSE is caught in a local minima while the training RMSE continues to decline. However if both training and validation become stagnant, the next epoch will be reduced to a maximum of n-1 iterations. The second is based on the idea that any cycles involving the model becoming trapped in a minima are detrimental to the development of an effective model. This

regimen will reduce epoch size as soon as iterations fail to result in improved RMSE. In keeping with this regimen's focus on reducing/eliminating stagnant training cycles, a smaller initial cycle size was selected for NMF.

Training in either fashion shall continue until the epoch size is zero or the training cycles no longer result in significant improvement. A significant gain in accuracy was previously defined as an reduction of RMSE of at least of $5.0e^{-5}$/10 iterations.

Section 10.2: Results

Hypothesis:

The adoption of a more flexible, adaptive training regimen should result in a final model that is significantly more accurate than its fixed-cycle counterpart. A significant improvement of accuracy will be defined as at least a 5% decrease in RMSE when comparing final model accuracy to initial RMSE values. Of the two training styles, the regimen that seeks to completely eliminate stagnant training iterations should result in the superior model.

Despite what seemed to be evidence to the contrary, there is no value to adopting a more complex multi-cyclic training regimen. Table 13 clearly shows that all of the models, regardless of training style, had a nearly identical final RMSE. Differences of less than 1% can easily be attributed to the type of variation that occurs when using any machine learning algorithm. This level of difference is more the product small advantages or missteps encountered during the training process, rather than an impact of how the training was organized.

It would seem that any reasonably-sized, multi-cyclic training regimen will result in a

model which approaches the upper bounds of accuracy possible with these algorithms. That being the case, there is no logical justification for the additional time and effort necessary to execute a decremented training regime.

Table 13: Comparison of All Netflix Training Regimen

Algorithm	**Initial RMSE**	**Final RMSE**	**% Improved**
SVD++	**1.12442**		
Phase 1		0.931921	17.12%
Phase 3 (Minima)		0.932509	17.07%
Phase 3 (No Minima)		0.931851	17.13%
BSGD	**1.36354**		
Phase 1		0.952970	30.11%
Phase 3 (Minima)		0.953158	30.10%
Phase 3 (No Minima)		0.954605	29.99%
SGD	**1.24070**		
Phase 1		0.959890	22.63%
Phase 3 (Minima)		0.958879	22.71%
Phase 3 (No Minima)		0.957286	22.84%
NMF	**1.58012**		
Phase 1		2.37543	-50.33%
Phase 3 (Minima)		2.37544	-50.33%
Phase 3 (No Minima)		2.38669	-51.04%
ALS	**1.25155**		

Phase 1		1.159920	7.32%
Phase 3 (Minima)		1.159500	7.35%
Phase 3 (No Minima)		1.159950	7.32%
TALS	**1.24425**		
Phase 1		1.147030	7.81%
Phase 3 (Minima)		1.150900	7.50%
Phase 3 (No Minima)		1.152530	7.37%
WALS	**5.52280**		
Phase 1		5.325280	3.56%
Phase 3 (Minima)		5.345600	3.21%
Phase 3 (No Minima)		5.356200	3.02%

Chapter 11: Conclusions and Future Work

Initial examination of the Collaborative Filtering Toolkit suggested that some of the behaviors observed in the fault-tolerant algorithms would result in models superior to those generated by single-epoch algorithms. It also appeared that limiting the size of individual training cycles to a small number of iterations would enhance these behaviors. Both assumptions turned out to be false. As with any other set of tools, gimmicks were trumped by matching the best tool to the job at hand. However this study was not without its lessons.

Although a multi-cyclic approach failed to outperform other algorithms, it yielded superior results to a single-epoch approach for the same algorithm. Additional training cycles allowed the some of the algorithms (BSGD, SGD, and SVD++ in particular) to work past limitations that left them susceptible to becoming trapped in local minima and overfitting. Exploration of a more complicated training regimen, incorporating a decremented epoch size, failed to produce further enhancements. This failure is actually a boon to those who would use the CFT. Automation of a multi-cyclic training regimen can focus on the relatively trivial task of a fixed cycle size, without sacrificing the resulting model's accuracy.

The Collaborative Filtering Toolkit represents a powerful tool which allows users to easily leverage the power data analysis. Continued research into this constantly evolving resource is necessary in order to understand and effectively utilize the ever growing ensemble of algorithms the CFT represents.

References

Bell, R. M.; Koren, Y. (2007). “Lessons from the Netflix Prize Challenge.” SIGKDD Explorations Vol. 9 Issue 2, 75-79. ACM New York, NY.

Bennett, J.; Lanning, S. (2007). “The Netflix Prize”. In Proceedings of KDD Cup and Workshop 2007, 3-6. ACM New York, NY.

Bickson, D. (2012). “Collaborative filtering with GraphChi”. Large Scale Machine Learning and Other Animals. Accessed on December 5, 2012. http://bickson.blogspot.com/2012/08/collaborative-filtering-with-demographical

Comon, P.; Luciani, X.; de Almeida, A. L. F. (2009). “Tensor Decompositions, Alternating Least Squares and other Tales.” Special issue, Journal of Chemometrics. In memory of R. Harshman.

Curnalia, J.; Lazar, A. (2013). “An Initial Investigation of Multi-Cyclic Training Regimen for Collaborative Filtering Models in GraphChi.” In Proceedings of Midwest Artificial Intelligence and Cognitive Science Conference 2013, 23-30. MAICS New Albany, IN.

Hernandez, V.; Roman, J. E.; Tomas, A. (2007). “Restarted Lanczos Bidiagonalization for the SVD in SLEPc.” SLEPc Technical Report STR-8.

Hinton, G. (2010).” A Practical Guide to Training Restricted Boltzmann Machines.” University of Toronto Tech Report UTML TR 2010-003.

Hu, Y.; Koren, Y.; Volinsky, C (2008). “Collaborative Filtering for Implicit Feedback Datasets”. IEEE International Conference on Data Mining (ICDM 2008), IEEE Washington, DC.

Koren, Y. (2009). “Collaborative filtering with temporal dynamics.” In Proceedings of the

15th ACM SIGKDD, 447-456. ACM, New York, NY.

Koren, Y (2008). “Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model”. ACM SIGKDD. ACM, New York, NY.

Koren, Y.; Bell, R.; Volinsky, C (2009). “Matrix Factorization Techniques for Recommender Systems.” In IEEE Computer, Vol. 42, No. 8. (07 August 2009), pp. 30-37. IEEE Washington, DC.

Kyrola, A.; Blelloch, G.; Guestrin, C. (2012). “GraphChi: Large-Scale Graph Computation on Just a PC.” In Proceedings of the Tenth USENIX Symposium on Operating Systems Design and Implementation, 31-46. OSDI Press Hollywood, CA.

Lee, D..D.; Seung, H.S. (2001). “Algorithms for Non-negative Matrix Factorization”, Advances in Neural Information Processing Systems 13, 556-562.

Low, Y.; Gonzalez, J.; Kyrola, A.; Bickson, D.; Guestrin, C.; Hellerstein,J. M. (2010). "GraphLab: A New Parallel Framework for Machine Learning." In Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence. AAAI Press Palo Alto, CA.

Netflix (2009). “Leaderboard”. NetflixPrize.com. Accessed on January 5, 2013. http://www.netflixprize.com//leaderboard

Pearl, J. (1994). "A Probabilistic Calculus of Actions". In UAI'94 Proceedings of the Tenth international conference on Uncertainty in artificial intelligence. Morgan Kaufman San Mateo CA. pp. 454–462.

Rendle, S. (2010). “Factorization Machines.” in Proceedings of the 10th IEEE International Conference on Data Mining (ICDM 2010), Sydney, Australia. IEEE Washington, DC.

Salakhutdinov, R.; Mnih, A. (2008). “Bayesian Probabilistic Matrix Factorization using Markov Chain Monte Carlo.” In the Proceedings of the International Conference on Machine Learning.

Walia, R.R. (2008). Collaborative Filtering: A comparison of graph-based semi-supervised learning methods and memory-based methods. Strengthening the Role of ICT in Development, 70-82.

Witten, I.H.; Frank, E.; Hall, M.A. (2011). Data Mining: Practical Machine Learning Tools and Techniques 3rd ed. Morgan Kaufman San Mateo CA.

Appendix: SVD and One-Sided SVD Output

```
./toolkits/collaborative_filtering/svd --training=smallnetflix_mm --nsv=3 --nv=10 --max_iter=5 --quiet=1 --tol=1e-1
WARNING:  common.hpp(print_copyright:104): GraphChi Collaborative filtering library is written by Danny Bickson (c). Send any  comments or bug reports to danny.bickson@gmail.com
[training] => [smallnetflix_mm]
[nsv] => [3]
[nv] => [10]
[max_iter] => [5]
[quiet] => [1]
[tol] => [1e-1]
Load matrix smallnetflix_mm
Starting iteration: 1 at time: 2.71863
Starting step: 1 at time: 3.42417
Starting step: 2 at time: 4.6307
Starting step: 3 at time: 5.88049
Starting step: 4 at time: 7.11613
Starting step: 5 at time: 8.36737
Starting step: 6 at time: 9.63118
Starting step: 7 at time: 10.9138
Starting step: 8 at time: 12.2115
Starting step: 9 at time: 13.5747
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
Number of computed signular values 5
Singular value 0      3276.69 Error estimate: 0.000305186
Singular value 1      1064.07 Error estimate: 1.18507e-13
Singular value 2      956.541 Error estimate: 0.00162432
Singular value 3      889.028 Error estimate: 0.00841469
Singular value 4       710.42 Error estimate: 0.0551811
Going to save output vectors U and V
Lanczos finished 22.5142
```

```
./toolkits/collaborative_filtering/svd_onesided --training=smallnetflix_mm --nsv=3 --nv=10 --max_iter=5 --quiet=1 --tol=1e-1
WARNING:  common.hpp(print_copyright:104): GraphChi Collaborative filtering library is written by Danny Bickson (c). Send any  comments or bug reports to danny.bickson@gmail.com
[training] => [smallnetflix_mm]
[nsv] => [3]
[nv] => [10]
[max_iter] => [5]
[quiet] => [1]
[tol] => [1e-1]
Load matrix smallnetflix_mm
Starting iteration: 1 at time: 0.560262
Starting step: 1 at time: 1.22546
Starting step: 2 at time: 4.76345
Starting step: 3 at time: 8.32286
Starting step: 4 at time: 11.8127
Starting step: 5 at time: 15.4576
Starting step: 6 at time: 19.1342
Starting step: 7 at time: 22.8375
Starting step: 8 at time: 26.5084
Starting step: 9 at time: 30.1965
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
set status to tol
Number of computed signular values 5
Singular value 0      3276.69 Error estimate: 3.8991e-14
Singular value 1      1064.07 Error estimate: 0.00146017
Singular value 2      956.541 Error estimate: 0.00782078
Singular value 3      889.028 Error estimate: 0.0440951
Singular value 4       710.42 Error estimate: 0.0864268
Lanczos finished in 47.1228
```