FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Martin Spišák

Sparse Approximate Inverse for Enhanced Scalability in Recommender Systems

Department of Software Engineering

Supervisor of the master thesis: Mgr. Ladislav Peška, Ph.D.
Study programme: Mathematics for Information Technologies
Study branch: Mathematics for Information Technologies

Prague 2023

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In .... date ....

Author's signature

I want to express my heartfelt appreciation to the following people who have played a significant role in the completion of my thesis:

First and foremost, I am profoundly grateful to my thesis supervisor, *Mgr. Ladislav Peška, Ph.D.*, for introducing me to the fascinating field of recommender systems and his invaluable support, which helped shape and enrich this thesis. I extend my sincere gratitude to *prof. Ing. Miroslav Tůma, CSc.*, for igniting my interest in matrix computations and for his readiness to serve as a consultant for this thesis. I am also immensely thankful to *Radek Bartyzal* and *Antonín Hoskovec* for allowing me to conduct this research as part of my job at GLAMI and for their patience. Moreover, I thank GLAMI for providing the infrastructure to conduct the experiments.

I would like to jointly thank all four for their willingness to co-author a research paper summarizing the results of this thesis. Their expertise was invaluable in shaping our submission, and I am forever thankful for the lessons I learned.

Lastly, I want to thank from all my heart *my parents, family*, and *Michaela* for their unwavering support throughout my academic journey. Their love, encouragement, and understanding have been a constant source of motivation and inspiration for me. I dedicate my thesis to them.

Title: Sparse Approximate Inverse for Enhanced Scalability in Recommender Systems

Author: Bc. Martin Spišák

Department: Department of Software Engineering

Supervisor: Mgr. Ladislav Peška, Ph.D., Department of Software Engineering

Abstract: In theory, the linear autoencoder EASE is one of the most capable collaborative filtering recommenders for large item domains with sparse user-item feedback. However, the model's weights are determined by the inverse of a matrix of dimension equal to the item set size. This inverse matrix is generally dense, and for large item sets, the computed weight matrix might be too large to store in memory during inference. Consequently, scaling the model beyond tens of thousands of items quickly becomes very expensive.

We propose a modification of EASE called SANSA to alleviate the issue. SANSA approximates the weights of EASE with prescribed density via an end-to-end sparse training procedure. To find a method capable of computing the sparse approximation efficiently, we investigate approaches for constructing sparse approximate inverse preconditioners. We select a method fitting for very large SPD problems with general sparsity patterns. The training procedure is robust and finds a good approximation of EASE even on datasets with dense item relations. Moreover, as the number of items in datasets grows, SANSA achieves unparalleled efficiency, even compared to EASE's previous state-of-the-art modification focused on scalability. Consequently, SANSA effortlessly scales the concept of EASE to millions of items.

Keywords: EASE sparse approximate inverse recommender systems

Contents

Introduction **3**

1 Personalized recommendation via collaborative filtering **5**
1.1 Overview of recommender systems . . . 5
1.1.1 Objectives of recommendation . . . 6
1.1.2 Applications and challenges . . . 7
1.2 Collaborative filtering . . . 8
1.2.1 User-item interaction data . . . 8
1.2.2 Relation to graphs . . . 9
1.3 Algorithms for collaborative filtering . . . 11
1.3.1 Neighborhood-based approaches . . . 12
1.3.2 Model-based approaches . . . 13

2 Sparse matrices **17**
2.1 Definitions . . . 17
2.2 Storage formats . . . 18
2.3 Operations with sparse matrices . . . 19
2.3.1 Efficient multiplication . . . 20

3 Embarrassingly Shallow Autoencoder **21**
3.1 Model definition . . . 21
3.1.1 Closed-form solution . . . 21
3.1.2 Properties of weights . . . 24
3.2 Model training . . . 25
3.3 Interpretation and advantages . . . 26
3.3.1 Similarity through user chains . . . 26
3.4 Expensive scaling . . . 30
3.4.1 Improvements . . . 32

4 Sparse approximate inverse **33**
4.1 Motivation . . . 33
4.2 Frobenius norm minimization methods . . . 34
4.2.1 When sparsity pattern is known . . . 35
4.2.2 Adaptive strategies . . . 35
4.3 Factorized sparse approximate inverse . . . 37
4.3.1 FSAI method . . . 38
4.3.2 Incomplete biconjugation . . . 39
4.3.3 Bordering approach . . . 40
4.4 Inverse incomplete factorization techniques . . . 41
4.5 Comparison of approaches . . . 42
4.5.1 Frobenius norm minimization methods . . . 42
4.5.2 Factorized sparse approximate inverse . . . 43
4.5.3 Inverse incomplete factorization techniques . . . 44

5 Enhancing scalability of Embarrassingly Shallow Autoencoder 45
5.1 Method selection . . . 45
5.1.1 Properties of the problem . . . 45
5.1.2 Selected approach . . . 47
5.2 Model definition . . . 48
5.2.1 Optimization objective . . . 48
5.2.2 Architecture . . . 49
5.3 Model training . . . 50
5.3.1 Sparse (and approximate) Cholesky factorization . . . 50
5.3.2 Choice of initial guess . . . 53
5.3.3 Uniform Minimal Residual algorithm . . . 54
5.3.4 Training procedure . . . 55
5.4 Implementation . . . 57

6 Experiments 59
6.1 Datasets . . . 59
6.1.1 Splits . . . 60
6.2 Metrics . . . 62
6.3 Baselines . . . 64
6.4 Results . . . 64
6.4.1 Robustness on small, dense datasets . . . 64
6.4.2 Robustness and efficiency on medium-sized, dense dataset 66
6.4.3 Trading accuracy for shorter training . . . 68
6.4.4 Extreme scalability . . . 69

Conclusion 72
Future work . . . 72

Bibliography 74

List of Figures 82

List of Tables 83

List of Abbreviations 84

A Appendix 85
A.1 Elimination tree of sparse Cholesky factorization . . . 85
A.1.1 Elimination tree . . . 85
A.1.2 Order of elimination and tree parallelism . . . 85
A.1.3 Column replication principle . . . 85
A.1.4 Equivalent condition for existence of a fill-in entry . . . 86
A.2 Supporting arguments for the choice of initial guess . . . 87
A.2.1 When the factor to-be-inverted is sparse . . . 87
A.2.2 Column elimination matrices and elimination tree . . . 87
A.3 Repository . . . 90
A.3.1 File organization . . . 90
A.3.2 Setup . . . 91
A.3.3 Reproducing the results . . . 92

Introduction

The information overload, exponentially magnified with the rise of the internet, has prompted the need for personalized recommendations. They simplify search, enable exploration, and offer suggestions that would otherwise go unnoticed. However, with their global reach and the power to decide what information to show to whom, online recommender systems shape the views and decisions of people worldwide. Understanding the implications, short and long-term effects, and potential dangers of omnipresent personalized recommendation is, therefore, one of the most critical tasks for artificial intelligence research and legislature for the foreseeable future. For that, it is crucial to understand the inner mechanisms of personalized recommendation.

Collaborative filtering has emerged as one of the predominant paradigms. The underlying idea is to leverage past user feedback to gain insights into their preferences by identifying similar users or items, then use the collective opinion to predict sentiments toward other items. However, the sparsity of the observed user-item interactions makes learning accurate models difficult in practice. The problem with sparsity becomes arduous in domains where users are numerous and have rich and diverse preferences, and the served items are abundant and often niche. Here, user satisfaction requires diversity only possible by viewing the interaction data from a broader perspective, which led to the popularity of deep learning approaches in recent years. Especially popular became graph neural networks (see, e.g., Wang et al. [2019], He et al. [2020], Mao et al. [2021]) and autoencoders (e.g., Ning and Karypis [2011], Liang et al. [2018], Steck [2019a]). Unfortunately, expanding the scope of view introduces a trade-off. Training state-of-the-art models on extensive datasets is often expensive, and the resulting models tend to be large and resource-intensive during inference. Consequently, high operational costs often limit their use in industry applications, and the increased complexity raises the entry threshold for researchers and limits the scope of experiments they can conduct.

Despite its simplicity, the linear autoencoder EASE^R proposed by Steck [2019a] is one of the most capable methods for collaborative filtering on large datasets with sparse user-item feedback, thanks to its ability to find long distance relational information about items. Another advantage of the model is its closed-form solution, which provides some level of interpretability and allows for faster training compared to approaches based on deep neural networks. As such, EASE^R is highly desirable for large production scenarios and academics alike. However, the weights of EASE^R are determined by the inverse of a matrix of dimension equal to the item set size. This inverse matrix is generally dense, and for large item sets, the computed weight matrix might be too large to store in memory during inference. Consequently, scaling the EASE^R model beyond tens of thousands of items quickly becomes very expensive.

This thesis aims to address these challenges and strike a balance between accuracy and efficiency in collaborative filtering. In particular, we search for an efficient and robust approach for finding an accurate sparse approximation of the EASE^R model. We begin by introducing recommender systems in Chapter 1, first generally, and then shift our focus to collaborative filtering. Since the gathered user feedback is often sparse in practical applications, we define sparse matrices and discuss basic storage and manipulation techniques in Chapter 2. In Chapter 3, we describe the EASE^{R} model. We derive its closed-form solution and explain its concept and advantages, but also its limited ability to scale as the number of items increases. Finally, we discuss prior attempts at solving this issue.

Like one other proposal, our idea is to modify the concept of EASE^{R} by finding a sparse, full-rank approximation of its weight matrix. Hence, we research possible ways to efficiently compute an accurate sparse approximate inverse of a very large matrix. For this, we turn to numerical mathematics, where efficient sparse approximate inverse techniques facilitate the construction of robust preconditioners for large sparse linear systems. Motivated by numerous challenging applications, the field of numerical mathematics has developed a comprehensive understanding of sparse matrix computations over the past 60 years or so (the books by Duff et al. [2017] and Scott and Tůma [2023] provide detailed summaries of contemporary knowledge of the topic). In Chapter 4, we discuss various approaches for constructing sparse approximate inverses (the primary reference here is an encompassing survey paper by Benzi and Tůma [1999]). We discuss the specifics of our problem in Chapter 5 and use the insights to select a method capable of efficiently computing the desired sparse approximate inverse with minimal unnecessary overhead. We then use the selected method to propose a scalable modification of EASE^{R}, resulting in a smaller model and reduced inference-time memory requirements. The proposed model, named Scalable Approximate NonSymmetric Autoencoder (SANSA), approximates the weights of EASE^{R} with prescribed density via an end-to-end sparse training procedure. We propose two variants of SANSA which suit different scenarios, depending on whether *training* memory limitations play a decisive role.

Our experiments in Chapter 6 demonstrate that the training procedure of SANSA is robust and finds a good approximation of EASE^{R} even on datasets with dense item relations. Moreover, as datasets grow beyond tens of thousands of items, SANSA achieves unparalleled efficiency compared to any other collaborative filtering model while matching or even outperforming them in accuracy. Consequently, SANSA effortlessly scales the concept of EASE^{R} to millions of items.

Our paper (Spišák et al. [2023]), which presents a summary of the findings documented in this thesis, has been accepted for the 17th ACM Conference on Recommender Systems (ACM RecSys 2023).

1. Personalized recommendation via collaborative filtering

We begin the thesis by introducing the domain of recommender systems. The domain itself spans many topics and is subject to intense active research. Therefore, we offer a concise overview of crucial aspects before narrowing our focus to the relevant part of the field. We refer interested readers to Falk [2019] and Ricci et al. [2022] for a detailed overview of the entire field of recommender systems.

1.1 Overview of recommender systems

Around the turn of the 21st century, recommendations became essential parts of the internet and our lives. Their importance grew fast with the rise of e-commerce, streaming platforms, and social media, which began to serve millions of users and accumulate giant pools of data, products, or content. Due to the large size of item catalogs, it may only be possible for users to find relevant products with assistance. Essentially, recommender system (RS) (frequently referred to as *the algorithms*) are designed to alleviate this problem.

While not exclusive to the internet, recommender systems are predominantly used online due to the ease of data collection and the abundance of valuable use cases. Hence, we formulate the basic terminology in the setting of a hypothetical website. The website has its *content* organized in data sets. The individual pieces of content are called *items*. A party visiting the website is called a *user*.

Definition 1.1 (Recommender system). *A* recommender system *is a service that suggests a potentially relevant selection of items to a user. The returned selections are called* recommendations*.*

Modern recommender systems are tailored to the specific use case and attempt to maximize one or more target metrics corresponding to business goals such as user satisfaction, customer retention, and profitability. Recommenders utilize many forms of data, including but not limited to item features and characteristics. Additionally, they often use user demographics and *preferences*[1] to create *personalized* recommendations. Note that personalization is not a necessity. We can divide recommender systems into three groups based on the level of personalization used:

1. **Non-personalized**: Examples of non-personalized recommendations include curated lists, most popular or newest items. Every user gets the same recommendations.

2. **Semi/Segment-personalized**: Semi-personalized recommendations differ for members of different user groups or segments. The groups are constructed using statistics about users' demography and various patterns. Information about the current session context may also be used for user segmentation.

[1] Interpretability and accurate extraction of user preferences are fundamental topics of recommender system research.

3. **Personalized**: "Recommended for you." The recommendations are user-specific, based on their preference profiles or past feedback.

Personalized recommendations are better tailored to individual user needs. A quote by Falk [2019] explains a key observation behind this:

> People aren't only interested in the popular items, but also in items that aren't sold the most or items that are in *the long tail*[2].

For this reason, personalized recommendations are preferable in many real-world applications. However, creating personalized recommendations is more expensive in terms of computation, and it can be challenging in some situations. A typical problem for personalized recommender systems is data sparsity, i.e., insufficient information to create good recommendations for a user. We will elaborate on this issue in Section 1.2 because the contribution of this thesis is closely related to the data sparsity problem. From now on, this thesis focuses solely on personalized recommendations.

1.1.1 Objectives of recommendation

Objectives of modern recommender systems can be complex. Historically, recommender systems primarily served to improve user experience. Their purpose has since shifted to optimize gains for *multiple stakeholders*. For illustration, we discuss different objectives of a European e-commerce platform GLAMI[3].

GLAMI is a fashion aggregation platform. Online retailers of fashion products partner with GLAMI to display their products in GLAMI's vast catalog of clothing articles. The catalog is well organized, and users can easily browse, compare products and find things they like. As a result, shoppers arriving from GLAMI have a high conversion rate[4], which is why the retailers are willing to pay GLAMI for their services. It is apparent that GLAMI has several objectives for which they need to optimize jointly:

1. *User satisfaction*, which is different for new users and returning users:
   (a) *New user satisfaction*: Increase the quality of *zero-shot*, *one-shot* and *few-shot* recommendation (i.e., recommendation with no or few inputs from the user) to incentivize new users to return to the platform.
   (b) *Returning user satisfaction*: Increase the recommendation quality for users with multiple interactions (e.g., by learning their preferences) to entice them to return to the website next time.
2. *Vendor satisfaction*: Increase conversion rates of users to keep retailers on the platform and lure in new ones.
3. *Shareholder satisfaction*: Minimize costs, maximize revenue, etc.

[2] The long tail refers to the shape of item popularity/interactions distribution when the majority of interactions belong to a small number of most popular items. The rest of the items fall in the long tail of this distribution. Recommending items from the long tail increases diversity but is often difficult. See Falk [2019], Section 1.1.2 for more details.

[3] `www.glami.cz`

[4] Conversion rate is the number of target actions (e.g., orders) per page visit.

Moreover, even "increasing the quality of recommendation" can mean many things. Traditionally, recommender systems predicted an item ordering according to some objective, for instance, "which item does the user like the most". In this context, better recommendation quality can mean, e.g., higher *accuracy*. However, depending on the domain and context, it may be beneficial to consider other aspects as well, such as *novelty* of items, *diversity*, *serendipity*, *coverage* of the catalog, or *synergy* between the items. Considering these aspects can significantly improve user satisfaction and help the recommendation quality individually and in aggregate. Further details are beyond the scope of this thesis, and we refer the interested reader to Falk [2019].

1.1.2 Applications and challenges

Recommender systems find applications in various areas. Users of social media platforms like Facebook, Instagram, TikTok, or Twitter will reliably find engaging content on their feeds. News platforms recommend personalized articles for their readers, with accurate suggestions on what to read next. Multimedia streaming services like Netflix, Spotify, and YouTube recommend movies, music, and videos to hundreds of millions of daily users. Large e-commerce sites like Amazon, Alibaba, or GLAMI can accurately recommend diverse but complementing sets of products from their enormous catalogs, making sure shoppers always find something to their liking. Services like these have succeeded in no small part thanks to their high-quality recommender systems. They can effectively recommend a diverse panel of videos for a user who does not know what to watch, create long personalized playlists that feel varied, and even aggregate the preferences of a group of users. These are but a few examples that demonstrate the progress made by recommender systems in the last couple of years.

On the other hand, recommender systems face significant challenges that must be acknowledged and addressed. Firstly, a recommender system may be *biased*, disproportionately favoring certain items or excluding others. Second, ensuring *fairness* in recommendations is crucial to avoid discrimination or exclusion based on protected attributes like race, gender, or age. Bias and unfairness become dangerous when the results of recommendations directly affect people, for example, when selecting job applicants or approving mortgage candidates. Lastly, reinforcing users' existing preferences and limiting exposure to diverse content leads to *filter bubbles* and *echo chambers*. Overcoming these challenges requires robust algorithms and ethical considerations to promote unbiased, fair, and diverse recommendations.

Consequently, production recommender systems have evolved into complex architectures, often consisting of multiple sub-systems that integrate diverse data and aggregate recommendations. This complexity is particularly notable in large-scale systems that cater to numerous users and offer recommendations from extensive item collections. Large-scale systems often employ a multi-stage approach to address the challenge of generating accurate recommendations swiftly from such vast item sets. A multi-stage recommender system organizes simpler recommenders into a pipeline. The initial stage, known as *item retrieval* or *candidate selection*, is designed to identify a broader selection of potential items using a simple, fast algorithm. The selected candidates are passed to subsequent layers, where more sophisticated algorithms can repeatedly filter, score, and arrange the items, potentially incorporating additional data. The increase in computational complexity of layers is counterbalanced by a reduced number of items to sort at each layer.

1.2 Collaborative filtering

The basic concept of personalized recommender systems is to select relevant item candidates based on a model of user sentiment toward the items. There exist many different ways to estimate user sentiment. Based on the type of data used in the recommendation, we may divide the approaches into two main groups:

1. **Content-based filtering**: Content-based methods build a profile of a user's interests or preferences. By comparing profile information with the attributes and metadata of items in the catalog, these methods select items that correspond well with the user's interests.

2. **Collaborative filtering**: The idea of collaborative filtering (CF) can be summarized as follows. When users have shown similar sentiments toward items in the past, it is reasonable to expect that they will agree on their preference for unseen items, too. Therefore, by comparing information about a user's sentiment with other users with similar past feedback, collaborative filtering methods recommend items the user has yet to see based on whether the segment of similar users liked them.

Content-based methods work best when enough information about items and, more importantly, users' taste profiles are available. Typically, this is the case in domains that serve content of the same type (e.g., movies), this content has identifiable qualities or categories to which the user can have a preference (e.g., genre), and the platform focuses on returning users so that the system can build their preference profiles. However, these methods can be more expensive. On the other hand, collaborative filtering allows us to identify users with similar tastes without thinking about their shared preferences - they must only like similar things. It then uses wisdom of the crowd of similar users to suggest items with mutual agreement. Collaborative filtering may be more accurate in few-shot scenarios and is typically cheaper to scale with a growing user base because it does not need to compute representations of user preferences. However, to create accurate recommendations, the systems may need a lot of user feedback, which may be difficult to obtain.

Apart from the two main classes, there exist *knowledge-based* methods (see, e.g., Jannach et al. [2010]) or *hybrid* methods which combine algorithms of different classes. The focus of this thesis is collaborative filtering.

1.2.1 User-item interaction data

In order to compute personalized recommendations, most CF-based recommender systems work with gathered user-item interaction data, also called *feedback*. It is customary to divide feedback into two main types.

Explicit feedback (also called *rating*) is direct input by a user through ratings (for example, on a 0-5 star scale), reviews, or explicit actions like liking items. Explicit feedback provides direct information about a preference from a user. However, it is scarce in real-world scenarios, and its reliability and accuracy are affected by numerous subjective factors such as the choice of rating scale or temporal changes in users' moods. A popular strategy for eliminating subjectivity and increasing the amount of gathered explicit feedback is to give users few possible actions. In recent years, many online platforms shifted from explicit feedback on a scale to *binary* feedback, where users have only one possible action for feedback - to like something.

Implicit feedback is derived from user behavior, such as click-through rates, browsing history, and purchase activity. It infers user preferences based on actions rather than direct input. For example, information about whether a user has seen individual items is a case of binary implicit feedback. Implicit feedback is often abundant and inserts less subjectivity but may lack interpretability.

Finally, hybrid approaches combine both types of feedback to reap the benefits of both types: strong information about preference from explicit feedback and an abundance of implicit feedback.

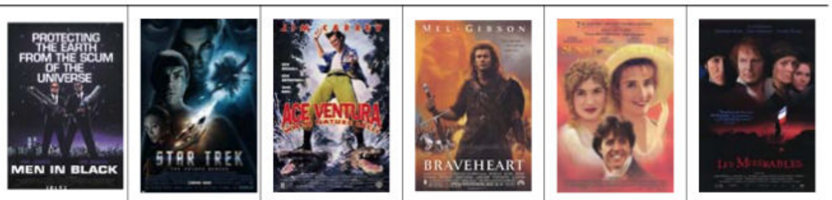
Interaction data is typically represented by a *user-item matrix* $X \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{I}|}$, where $\mathcal{U}$ is the set of users, $\mathcal{I}$ is the item set and $X_{i,j}$ is the collected feedback of the i-th user for the j-th item, see Figure 1.1. A fundamental observation is that in practice, X is typically (very) **sparse**[5], especially in domains with extensive item sets. Most users interact with small portions of the item set in such situations. The result - *data sparsity* - is a problem for CF-based recommender systems (and personalized recommender systems in general), as limited or incomplete feedback from users hampers the system's ability to understand user preferences. Another issue related to data sparsity is the so-called *cold start* when the system struggles to provide recommendations for new users or items. While cold start can be a temporary problem (i.e., until we gather initial data), data sparsity can limit the recommendation quality in the long term and is a common problem, especially for collaborative filtering methods. Luckily, recently developed state-of-the-art methods by Steck [2019a] can extract the most out of limited available data and, as our main contribution, we show how the *inherent* data sparsity can be exploited to scale this state-of-the-art algorithm to domains with extremely large item sets.

1.2.2 Relation to graphs

The user-item matrix X represents a bipartite graph $\mathcal{G}_X = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{V}_\mathcal{U} \bigcup \mathcal{V}_\mathcal{I}$ is the union of vertices representing the set of users ($\mathcal{V}_\mathcal{U}$) and vertices representing the set of items ($\mathcal{V}_\mathcal{I}$), and the edges between users and items $\mathcal{E} \subseteq \mathcal{V}_\mathcal{U} \times \mathcal{V}_\mathcal{I}$ represent extracted user sentiment towards items; see Figure 1.2. Additionally, in case of explicit feedback, the edges are assigned weights; we may assign all edges weight 1 in case of implicit feedback. An important observation is that when the user-item matrix X is sparse, the vertices of $\mathcal{G}_X$ are sparsely connected.

One possible formulation of CF is that during inference, the algorithm receives a list of user feedback or interactions, and its task is to predict the most relevant

[5] A sparse matrix has relatively few nonzero entries. We will provide a more formal definition in the following chapter.



	Comedy	Action	Comedy	Action	Drama	Drama
Sara	5	3		2	2	2
Jesper	4	3	4		3	3
Therese	5	2	5	2	1	1
Helle	3	5	3		1	1
Pietro	3	3	3	2	4	5
Ekaterina	2	3	2	3	5	5

Figure 1.1: Example of a user-item rating/interaction matrix. (Falk [2019])

items (which are often required to be previously unseen). To achieve this, the algorithm uses a model of *item-item* relations, which can be represented by graph $\mathcal{G}_\mathcal{I} = (\mathcal{V}_\mathcal{I}, \mathcal{E}_\mathcal{I})$ with weighted edges $\mathcal{E}_\mathcal{I} \subseteq \binom{\mathcal{V}_\mathcal{I}}{2}$, illustrated in Figure 1.3. For edge $e = \{v_{i_1}, v_{i_2}\} \in \mathcal{E}_\mathcal{I}$ representing the relation between items $i_1, i_2 \in \mathcal{I}$, its weight $w(e)$ is computed by aggregating information from the user-item matrix X (equivalently graph $\mathcal{G}_X$). While the particular method of aggregation depends on the model used, a common choice is to use the *cosine similarity* between columns $\overrightarrow{x_{i_1}}$ and $\overrightarrow{x_{i_2}}$ of X,

$$\texttt{cosine\_sim}(\overrightarrow{x_{i_1}}, \overrightarrow{x_{i_2}}) := \frac{\overrightarrow{x_{i_1}}^T \overrightarrow{x_{i_2}}}{\|\overrightarrow{x_{i_1}}\| \|\overrightarrow{x_{i_2}}\|} = \frac{\sum_{u \in \{1,\dots,|\mathcal{U}|\}} X_{u,i_1} \cdot X_{u,i_2}}{\sqrt{\sum_{u \in \{1,\dots,|\mathcal{U}|\}} X_{u,i_1}^2} \sqrt{\sum_{u \in \{1,\dots,|\mathcal{U}|\}} X_{u,i_2}^2}}.$$

Cosine similarity measures the similarity between the two items using the angle between the $|\mathcal{U}|$-dimensional vector representations of items i_1 and i_2, which we obtain from the user interaction data.

The edges of $\mathcal{G}_\mathcal{I}$ and their weights represent modeled relations between pairs of items (the larger the weight, the more similar the items). Specifically, a positive weight represents a positive relationship or similarity, and a negative weight represents dissimilarity. Moreover, when v_{i_1} and v_{i_2} are not connected by an edge in $\mathcal{G}_\mathcal{I}$, the model did not learn a relation between items i_1 and i_2. The direct item relations in $\mathcal{G}_\mathcal{I}$ can be used to predict new, interesting items for a user by computing scores for all items with a known relation to at least one of the items with prior user interaction from this user.

To formalize this idea, consider a user $u \in \mathcal{U}$ with interacted items $\mathcal{I}_u \subset \mathcal{I}$. Let $f(u, i_j)$ denote the feedback from user u for item $i_j \in \mathcal{I}_u$. Every item $i_j \in \mathcal{I}_u$ represents a vertex v_{i_j} in the graph $\mathcal{G}_\mathcal{I}$ from which we start the search. To compute a score for some other item $i_k \notin \mathcal{I}_u$, we verify whether it is connected by an edge to some item $i_j \in \mathcal{I}_u$. If there exists such i_j, denote $e_{j,k} = (v_{i_j}, v_{i_k})$ the edge in $\mathcal{G}_\mathcal{I}$. The discussion now splits into cases that share the following idea. Suppose the user's sentiment towards item i_j is positive ($f(u, i_j) > 0$). In that case, it is reasonable to assume that they will also like items similar to i_j - that is, i_k s.t. v_{i_k} is connected to v_{i_j} by an edge with positive weight ($w(e_{j,k}) > 0$).

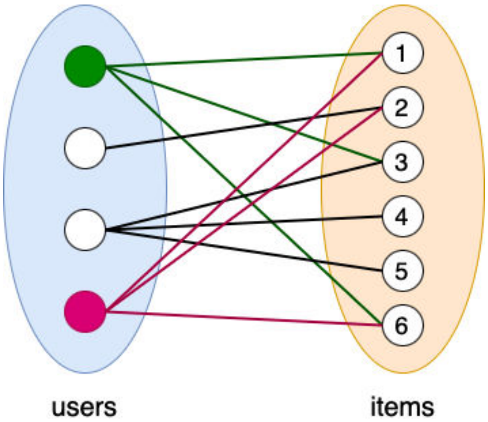


Figure 1.2: An example bipartite graph $\mathcal{G}_X$ of user-item (binary) feedback. Users *Green Gabe* and *Magenta Mike* appear to have similar tastes since both liked `item 1` and `item 6`. *Green Gabe* also likes item 3, which *Magenta Mike* has yet to see. Hence, our CF model recommends `item 3` to *Magenta Mike*.

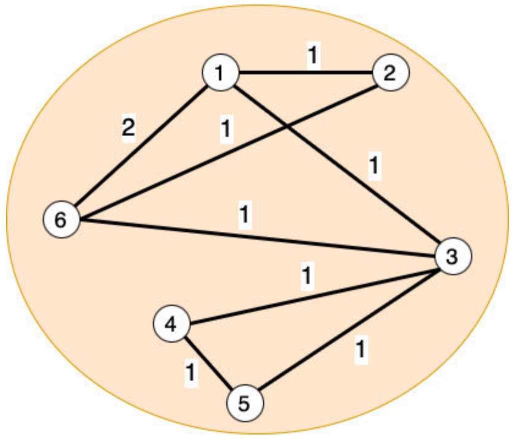


Figure 1.3: Item-item relation graph $\mathcal{G}_{\mathcal{I}}$ obtained from $\mathcal{G}_X$ (from Fig. 1.2) by a specific choice of aggregation: the count of *paths* of length 2. This is an example of a neighborhood-based approach. Based on the (binary) input interactions of *Magenta Mike*, `Item 3` has the highest `score` (see Eq. 1.1).

If both $f(u, i_j) < 0$ and $w(e_{j,k}) < 0$ (i.e., the user dislikes the item i_j, but the candidate item i_k is dissimilar to i_j), it is also not unreasonable to assume that the user would like the item i_k. Finally, if either $f(u, i_j) < 0$ and $w(e_{j,k}) > 0$, or $f(u, i_j) > 0$ and $w(e_{j,k}) < 0$, it makes sense to assume that the user u would not like the item i_k. A common way to define a score function that agrees with the above reasoning is the following:

$$\texttt{score}(i_k) = \sum_{i_j \in \mathcal{I}_u} w(e_{j,k}) f(u, i_j). \tag{1.1}$$

Finally, items with the highest calculated scores are recommended to the user.

1.3 Algorithms for collaborative filtering

The traditional and most common formulation of collaborative filtering assumes we are given a (partially filled) user-item matrix and a user whose feedback is stored in a row of this matrix (so-called *known user*). The system's task is to predict the most suitable items for this user from the set of unseen items. CF methods internally learn to predict ratings of user-item pairs, i.e., to fill in the empty entries of the user-item matrix. During prediction for this user, the system typically predicts the user's ratings for all or some of the items and recommends the highest-rated ones.

This formulation does not work when a *new user* - whose interactions are not in the user-item matrix - visits a website and requests a recommendation. Similarly, the method cannot recommend *new items* because the user-item matrix does not have a corresponding column. Both situations are common and need to be addressed in practice. For a new item, the only possibility is gathering some interaction data by showing it to users and then retraining the model; we will not discuss this situation further. On the other hand, being able to recommend to a new user[6] is often very important. Gathering some interaction data first is necessary, but then there are ways to recommend to a user based only on their interactions, i.e., agnostic to who they are. This approach is prevalent in contemporary CF methods.

The final thing we would like to mention briefly is the user-item matrix preprocessing. Explicit or implicit feedback may come from different scales, which may or may not include zero. However, some (if not most) of the entries in the user-item matrix are not filled, meaning there is an implicit zero value at that position in the user-item matrix. Importantly, this is not the same as the user giving the item an explicit rating of zero; it is merely a missing value. Therefore, it is a good idea to rescale the matrix so that the implicit zeros fall somewhere in the middle of the preference scale. There are many different ways to achieve this, some more suitable in the given situation and for the given CF method, but this is beyond the scope of this thesis. The important message is that CF methods typically expect the user-item matrix to be processed with this in mind.

Based on the method used to estimate the user-item feedback, CF methods are often split into two groups:

1.3.1 Neighborhood-based approaches

Rows of the interaction matrix X represent vectors in the $|\mathcal{I}|$-dimensional space of user preference towards items in the dataset. Similarly, columns of X represent vectors in the $|\mathcal{U}|$-dimensional space of item preference from all users in the dataset. The similarity of users or items may be computed using, for instance, cosine similarity. The methods then select the *most* similar users (or items) based on, e.g., tolerance or the absolute count, and aggregate the preferences, often using weighted averaging. This is the mechanics behind neighborhood-based approaches - *user-based* and *item-based*.

The most common example of a neighborhood-based method is the user-oriented k-nearest neighbors (USERKNN) method, first proposed by Resnick et al. [1994]. Figure 1.4 shows how the computation of the predicted score for a user u_1 and an unseen item i_3 in more detail. First, the algorithm computes the similarity of the user in question with all other users in the matrix X (or, perhaps, only with a selected subset of users to save computation time). Note that it makes sense to consider only a subset of columns of X - those that correspond to items with known preference from the user u_1 (in this example, columns 1,2,4,5,6). Excluding the column corresponding to the item in question is also reasonable. The final score is computed by aggregating the known preferences toward item i_3 of users in a selected neighborhood of most similar users. Analogously, an item-based method (e.g., ITEMKNN proposed by Sarwar et al. [2001]; see also future work in Deshpande and Karypis [2004]) first computes the similarity between the column corresponding to the item i_3 and other columns (here, it makes sense only to consider columns with feedback from user u_1) while excluding the row corresponding to u_1. It then selects a neighborhood of the most similar items and aggregate the preferences from the user u_1 for these items.

[6] More precisely, to a user whose interaction data were not used for model training.

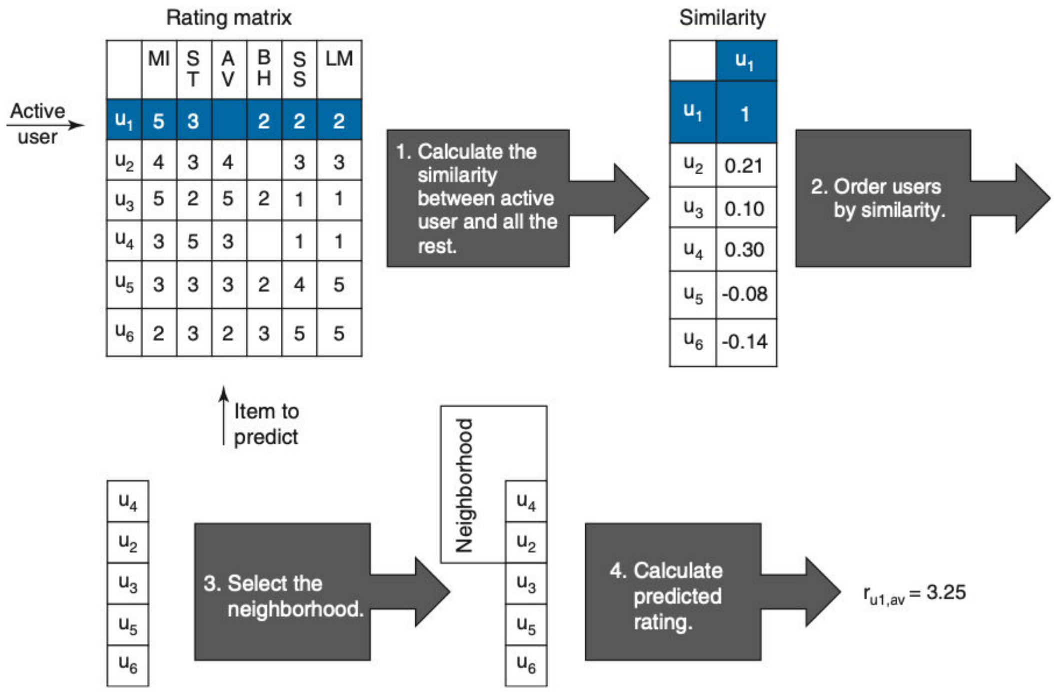


Figure 1.4: User-based neighborhood-based filtering. (Falk [2019])

Both user-based and item-based neighborhood-based methods have their advantages and disadvantages. User-based collaborative filtering works well when the user-to-item ratio is high, while item-based collaborative filtering is suitable when the item-to-user ratio is high. Both, however, struggle when the user-item matrix X is very sparse. In such cases, the similarity calculation may lack accuracy, resulting in poor recommendation quality. Moreover, if, for example, similar users only interacted with items previously seen by the user, a user-based cannot recommend novel items.

From a "global" perspective, neighborhood-based methods are based on constructing a user-user or item-item similarity matrix. An important distinction from other methods is that neighborhood-based approaches construct this matrix without optimizing some objective function.

1.3.2 Model-based approaches

Model-based approaches are based on building a parametric function that takes as input a user (or their feedback) and outputs the calculated user preference score for a subset of items. The items with the highest predicted scores are returned as recommendations. The function's parameters are learned by optimizing a selected objective function based on the interactions in the user-item matrix. This is an example of *machine learning*. A subset of interactions in the user-item matrix is used as the *training data* for the CF *model*. In training, the model's parameters are adjusted in such a way as to minimize a *loss function*. After training, unused interaction data can be used to *evaluate* the model's performance. The model should learn to *generalize*, i.e., to create good predictions even for inputs not seen during training. For more details on machine learning and various methods and practices, refer to, e.g., the textbook by Bishop [2006].

Matrix factorization methods

Matrix factorization methods aim to learn hidden (*latent*) factors that influence the observed user-item interactions. See, e.g., the article by Koren et al. [2009] for an in-depth overview. Users and items are mapped to a shared latent space. In the latent space, user vectors $\vec{q_u} \in \mathbb{R}^k$ represent the agreement of users with individual factors (analogously for item vectors $\vec{p_i} \in \mathbb{R}^k$), and the inner product of user and item representations models the observed feedback: the rating of user u for item i is estimated as $r_{u,i} = \vec{q_u}^T \vec{p_i}$. Note that since $\vec{q}^T \vec{p} = \cos(\angle(\vec{q}, \vec{p})) \|\vec{q}\| \|\vec{p}\|$, the ratings are modeled as the *cosine similarity* of user and item representations *multiplied by their magnitudes.*

An appropriate latent space is found by approximately decomposing the original user-item matrix into two or more *low-rank* matrices. A traditional approach is to use the **singular value decomposition (SVD)** (SVD), a fundamental technique from linear algebra. SVD provides a valuable instrument but suffers from several limitations. Firstly, computing a full SVD can be very expensive. This can be easily solved by computing a truncated singular value decomposition (TSVD) (i.e., only the part corresponding to several largest singular values) instead. TSVD is fast, requires much less storage, acts as denoising, and mathematically provides the optimal low-rank approximation of the user-item matrix. However, SVD does not work for matrices with missing entries; *missing entries are assumed to be zeros*, and this assumption is implicitly used in the approximation. This is **conceptually incorrect**: we have no information about the missing entries. The goal is to estimate their hidden values, not the imputed zero values.

Recent approaches suggest optimizing the regularized squared reconstruction loss using only the seen interactions. These methods learn representations $\vec{q_u}$ and $\vec{p_i}$ of every user and item so that the loss function

$$L = \frac{1}{|\mathcal{U}| \cdot |\mathcal{I}|} \sum_{\substack{u \in \mathcal{U},\ i \in \mathcal{I} \\ r_{u,i} \text{ is known}}} (r_{u,i} - \vec{q_u}^T \vec{p_i})^2 + \lambda(\|\vec{q_u}\|^2 + \|\vec{p_i}\|^2) \tag{1.2}$$

is minimized for all *known* entries $r_{u,i}$ of the user-item interaction matrix X. The optimization problem resulting from Equation (1.2) is not convex, and two main approaches for its optimization were proposed. **Funk-SVD** (Funk [2006])[7], uses stochastic gradient descent to optimize the above problem. The **Alternating Least Squares (ALS)** method (Hastie et al. [2014]) uses the fact that fixing one set of the variables in Equation (1.2) (either user factors $\vec{q_u}$ or item factors $\vec{p_i}$) results in a convex optimization problem. ALS then iteratively minimizes the reconstruction error between the original user-item matrix and its approximation by alternating updates to the factors, with one factor fixed while the other is optimized.

The final well-known matrix factorization method we mention is the **Non-negative Matrix Factorization (NMF)** (e.g., Cichocki and Phan [2009] or

[7] Funk's successful entry to the 2006 Netflix Prize competition (Bennett and Lanning [2006]).

Févotte and Idier [2011]). NMF aims to find non-negative representations of users and items by including a non-negativity constraint in the above optimization problem. The user-item matrix is decomposed into two non-negative matrices. The approach is practical when dealing with non-negative data such as ratings or counts.

Deep learning

Recent years have witnessed the popularity of *deep learning* (see, e.g., the textbook by Chollet [2021] for introduction to the topic) for CF tasks. One popular approach is to use **graph neural networks**, e.g., ULTRAGCN (Mao et al. [2021]), LIGHTGCN (He et al. [2020]) or NGCF (Wang et al. [2019]), Graph neural networks used in CF are often *convolutional* networks which extract information from weighted unoriented graphs representing the user-item feedback data. Another very recent idea for CF is **diffusion models** like BSPM (Choi et al. [2022]).

Autoencoders

Some of the most popular deep learning approaches for CF belong to the class of **autoencoders**. An autoencoder (initially proposed by LeCun [1987]) is a model trained to reconstruct the input and learn meaningful representations of input data. The learned representations can then be used for various applications, such as clustering or (more recently) generative tasks (refer to, e.g., Bank et al. [2021] or Zhai et al. [2018]).

Formally, in the most basic setting, the goal of the training is to learn an *encoder* function $\widehat{E} : \mathbb{R}^n \to \mathbb{R}^k$ and a *decoder* function $\widehat{D} : \mathbb{R}^k \to \mathbb{R}^n$ that satisfy

$$\widehat{E}, \widehat{D} = \mathrm{argmin}_{E,D}\mathbb{E}[\Delta(\vec{x}, D(E(\vec{x})))],$$

where $\mathbb{E}$ is the expected value over the distribution of $\vec{x}$ in the training dataset and Δ is the selected *reconstruction loss* function (often $\Delta(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\|_2^2$). The reconstruction loss measures the distance between the input and the decoded output. Figure 1.5 shows a high-level illustration of autoencoder architecture. Neural networks typically represent the encoder and decoder.

The encoder's task is to create a representation of the n-dimensional input in a k-dimensional *latent space*. This representation is then decoded back to the original feature space of the input by the decoder. Very often, k is selected to be (much) smaller than n. The resulting latent representation is *compressed*. In some cases, subspaces of the latent space can be attributed to identifiable features of the input data (see, e.g., the section on variational autoencoders in Chollet [2021]). Note, however, that it is not easy to force the model to learn "good" latent representations, and various sophisticated modifications (using different regularization and compression techniques) were developed for this reason.

As the most notable example, older techniques often created nonsensical predictions for data not seen during training. Instead of learning to map input data points $\vec{x} \in \mathbb{R}^n$ to individual points $\vec{z} \in \mathbb{R}^k$ and back to $\vec{x}$, **variational autoencoder (VAE)** (introduced by Kingma and Welling [2022]) view $\vec{x}$ and $\vec{z}$ as realizations of random variables $\vec{\mathrm{X}}$ and $\vec{\mathrm{Z}}$ respectively and learn *probabilistic* encoders and decoders, i.e., where the points $\vec{x}$ are likely to map in the latent

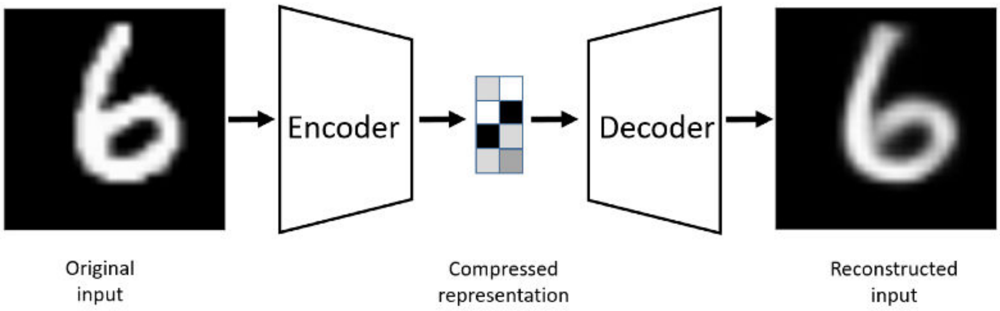


Figure 1.5: Autoencoder architecture (Bank et al. [2021]). The input data is encoded to a latent representation, which is then decoded.

space and vice versa, assuming that $P(\vec{z})$ is fixed and independent of $\vec{x}$. Internally, the decoder in VAE approximates the conditional probability distribution $P(\vec{x}|\vec{z})$ (the probability of $\vec{x}$ being decoded from the latent representation $\vec{z}$) as a parametric function $P_\theta(\vec{x}|\vec{z})$. For autoencoder training, the posterior $P_\theta(\vec{z}|\vec{x})$ (of the encoder) is needed, too. In VAE, this is approximated by the conditional distribution $Q_\phi(\vec{z}|\vec{x})$, which for $\vec{x} \sim P(\vec{x})$ is parametrized as a multivariate normal distribution $\mathcal{N}(\vec{\mu}, \vec{\sigma}^2 I)$. The encoder predicts the parameters $\vec{\mu}$ and $\vec{\sigma}$ for a given $\vec{x}$. The latent representation $\vec{z}$ is sampled from this distribution and subsequently decoded by the decoder. The encoder weights ϕ and the decoder weights θ are optimized by a combination of reconstruction loss and *latent loss*, which measures the distance between the prior distribution $P(\vec{z}) = \mathcal{N}(0, I)$ (independent of $\vec{x}$) and $Q_\phi(\vec{z}|\vec{x})$ over all $\vec{x}$ using Kullback-Leibler divergence[8].

Autoencoders provide the model architecture that fits the collaborative filtering task and enables recommendations based solely on user interactions. A (sparse) vector of user interactions (i.e., a row in the user-item matrix X) represents a point in a $|\mathcal{I}|$-dimensional space of user preferences. An autoencoder trained using the rows of X learns to represent input preference vectors in a latent space, and this representation is used to generate similar preference vectors in the original user preference space. If the predicted vector is close to the original preference vector, items with positive feedback (or score) in the prediction are likely good recommendations for the user. Over the past few years, multiple autoencoder models achieved state-of-the-art recommendation accuracy in popular benchmarks. Some of the well-regarded examples include the denoising autoencoder CDAE (Wu et al. [2016]), variational autoencoders like MULT-VAE$^{\text{PR}}$ (Liang et al. [2018]) and RECVAE (Shenbin et al. [2020]), and linear autoencoders SLIM (Ning and Karypis [2011]), EASE$^{\text{R}}$ (Steck [2019a]) and ELSA (Vančura et al. [2022]).

[8]Kullback-Leibler (KL) divergence (Kullback and Leibler [1951]) is an information-based measure of disparity among probability distributions. It is commonly used as a loss function in machine learning due to its close relation to *maximum likelihood estimation*. See Joyce [2011] for details.

2. Sparse matrices

In practical applications of collaborative filtering, the gathered feedback is often sparse in the sense that a user typically interacts with only a small portion of the item set. Motivated by this, we define a sparse matrix and explain frequently used storage schemes for sparse vectors and matrices and basic operations with them. The primary reference for this chapter is the textbook by Scott and Tůma [2023].

2.1 Definitions

A **sparse matrix** contains many zero entries that can be exploited for computation efficiency gains. Numerous practical applications require solving linear systems $A\vec{x} = \vec{b}$, where A is large and sparse. Exploiting the sparsity of A by avoiding operations with zero entries benefits efficiency and enables very large systems to be solved. Many problems cannot be solved without using sparsity to reduce memory requirements and the number of required operations.

Definition 2.1 (Sparse matrix)**.** *A matrix $A \in \mathbb{R}^{m \times n}$ is* sparse *if it is advantageous to exploit its zero entries (by avoiding them during computation). Otherwise, A is* dense*.*

Similarly, we may define a **sparse vector** as a vector with many zero entries or a vector whose zero entries can be exploited. For example, when performing a linear combination of n vectors with coefficients from a vector $\vec{a} \in \mathbb{R}^n$, if $\vec{a}$ has many zero entries, the linear combination needs to combine fewer vectors, hence saving floating point operations.

We refer to the entries of a sparse matrix $A \in \mathbb{R}^{m \times n}$ using the notation

$$A = (A_{i,j}),\ 1 \leq i \leq m,\ 1 \leq j \leq n.$$

An entry whose value is not zero (or is treated as not being equal to zero) is called a **nonzero**. As an example, let us consider the nonzeros of a diagonal matrix. In any diagonal matrix, every entry outside the main diagonal must be zero. Therefore, the nonzeros of a diagonal matrix are located on the main diagonal.

Definition 2.2 (Sparsity pattern)**.** *For a matrix $A \in \mathbb{R}^{m \times n}$, its* sparsity pattern *$\mathcal{S}(A)$ is the set of nonzeros $\mathcal{S}(A) = \{(i, j) \mid A_{i,j} \neq 0,\ 1 \leq i \leq m,\ 1 \leq j \leq n\}$.*

The sparsity pattern *$\mathcal{S}(\vec{v})$ of a vector $\vec{v} \in \mathbb{R}^n$ is the set of nonzero entries $\mathcal{S}(\vec{v}) = \{i \mid \vec{v}_i \neq 0,\ 1 \leq i \leq n\}$.*

A square matrix $A \in \mathbb{R}^{n \times n}$ is *structurally (or symbolically) singular* if A is singular for any combination of values assigned to the entries on the positions given by $\mathcal{S}(A)$. In this situation, A is singular due to its sparsity pattern (e.g., when $\exists i \in \{1, \ldots, n\}\ \forall j \in \{1, \ldots, n\} : (i, j) \notin \mathcal{S}(A)$). If $\mathcal{S}(A)$ is symmetric, we say that A is *structurally symmetric*.

We denote $\texttt{nnz}(A)$ **the number of nonzeros in** A. Analogously, we denote $\texttt{nnz}(\vec{v})$ the number of nonzeros in $\texttt{nnz}(\vec{v})$. Apparently, it holds that $\texttt{nnz}(A) = |\mathcal{S}(A)|$ and $\texttt{nnz}(\vec{v}) = |\mathcal{S}(\vec{v})|$. The number of nonzero entries in A can be used to define the density and sparsity of a matrix formally:

Definition 2.3 (Density and sparsity)**.** *The* density *of a matrix A is the share of nonzero entries among all entries:* $\texttt{density}(A) = \frac{\texttt{nnz}(A)}{m \cdot n}$*. We define the* sparsity *of A as* $1 - \texttt{density}(A)$*.*

2.2 Storage formats

The density (or sparsity) of a matrix determines the potential compression we can achieve by only storing nonzero entries. We describe several commonly used formats for storing sparse matrices.

Coordinate format The most straightforward way to store a sparse matrix is using the **COOrdinate format (COO)**. This format represents A using triplets $(i, j, A_{i,j})$, where i, j are the row and column indices of $A_{i,j}$. The coordinate format is typically used only for creating sparse matrices. Operations with matrices in the coordinate format are slow because they often need to find required entries first. Moreover, storing a matrix A in COO format requires memory of size $3 \times \texttt{nnz}(A)$, which can be improved.

Compressed sparse formats Probably the most widely used sparse matrix storage formats are the **Compressed Sparse Row format (CSR)** and the **Compressed Sparse Column format (CSC)** proposed by Jennings [1966].

CSR is based on the idea of compressed vector storage. Let $\vec{v}$ be the vector

$$\vec{v} = (3, 0, -2, 0, 0, 56, 0, 0, 0, 9, 0)^T \in \mathbb{R}^{11}.$$

This vector of length 11 has only four nonzero entries. Its sparsity allows us to save space when storing this vector in a computer. Instead of storing a single contiguous array with 11 floating point numbers, we may represent $\vec{v}$ by two contiguous arrays - one for the nonzero values, the other with integers representing the positions of nonzeros in the original vector - each with $\texttt{nnz}(\vec{v}) = 4$ elements:

Subscripts	1	2	3	4
`indices`	1	3	6	10
`data`	3	-2	56	9

Denoting $\texttt{len}(\texttt{arr})$ the length of array `arr`, it is clear that $\texttt{len}(\texttt{indices}) = \texttt{len}(\texttt{data}) = \texttt{nnz}(\vec{v})$. We see that to store a sparse vector, we only need a memory of size $2 \times \texttt{nnz}(\vec{v})$. Such compression may save significant space when the vector (or matrix) is very sparse. Next, we show how to extend this idea to matrices.

The compressed sparse *row* format compresses each row vector of the matrix using the method above to store a sparse matrix efficiently. It then concatenates the `indices` arrays and the `data` arrays obtained from each row, forming two long `indices` and `data` arrays. The only thing remaining is to save the starting position of `indices` and `data` corresponding to each row (in order) in a `indptr` array. Finally, we include $\texttt{len}(\texttt{indices})+1$ (equivalently $\texttt{len}(\texttt{data})+1$) as the final element of `indptr`. This way, we know that the i-th row of the matrix is has its indices stored in $\texttt{indices}[\texttt{indptr}[i] : \texttt{indptr}[i+1]]$ and its data stored in

$\texttt{data}[\texttt{indptr}[i] : \texttt{indptr}[i+1]]$. Here, $\texttt{arr}[a : b]$ are the data stored in the array arr between positions a (including) and b (excluding), and, notably, $\texttt{arr}[a : a]$ is an empty array.

$$\begin{bmatrix} 1 & 2 & 0 & 0 \\ 5 & 0 & 7 & 8 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 15 & 0 \\ 17 & 18 & 19 & 0 \end{bmatrix}$$

Subscripts	1	2	3	4	5	6	7	8	9
indptr	1	3	6	6	7	10			
indices	1	2	1	3	4	3	1	2	3
data	1	2	5	7	8	15	17	18	19

For illustration, in the above example, the nonzero values in the second row of the matrix are stored in $\texttt{data}[\texttt{indptr}[2] : \texttt{indptr}[3]] = \texttt{data}[3 : 6]$ and the indices of nonzeros in the third row is stored in $\texttt{indices}[\texttt{indptr}[3] : \texttt{indptr}[4]] = \texttt{indices}[6 : 6]$, which is an empty array.

Storing a matrix in the CSR format can provide great compression. Specifically, for $m \times n$ matrix A, we need to store only $2 \times \texttt{nnz}(A) + m + 1$ values. In practice, the arrays indptr, indices, data may store different data types: indptr and indices store (non-negative) integers, while data most commonly store floating point numbers in a selected precision.

The compressed sparse column format is analogous to CSR, except it stores the matrix using compressed column vectors. CSR allows fast access to rows of the matrix (as shown above), while CSC allows fast access to the columns. Depending on the intended use case, using one or the other may be beneficial. Also note that the conversion CSR $\rightarrow$ CSC and vice versa are have linear complexity, specifically $O(\texttt{nnz}(A) + \min(m, n))$ for a $m \times n$ matrix; see, e.g., the source code[1] of SciPy (Virtanen et al. [2020]).

The CSR and CSC formats are **static** data structures. While reading A is straightforward, making modifications is complicated. For instance, adding a new entry at a specified location or removing an entry is challenging because it requires a) finding the position where the modification takes place and b) shifting parts of indptr, indices and data, which may require memory reallocation. When deleting an entry, the value of the entry could be, alternatively, set to zero, which is relatively fast, but doing so many times results in many so-called explicit zeros stored in the sparse structure designed to avoid them. This is inefficient, as the operations on A are then performed on zeros, creating unnecessary compute and memory overhead.

Many additional storage formats exist, which we do not discuss here. These formats have specific use cases, such as block formats tailored to situations with block sparse matrices or dynamic formats (often based on linked lists), which sacrifice some compression for easy access to particular entries. We refer to a technical report by Saad [1990] or Scott and Tůma [2023] and references therein for a detailed discussion of possible approaches.

2.3 Operations with sparse matrices

The coordinate format provides the simplest way to insert or modify entries by including a new input triplet. The new triplet stores the update if an entry al-

[1] https://github.com/scipy/scipy/blob/main/scipy/sparse/sparsetools/csr.h

ready exists at the modified position. Therefore, the COO format is preferable for constructing new sparse matrices. However, accessing an entry at the specified position is difficult because it requires finding all triplets with specified coordinate positions in a list of triplets. To increase efficiency, we can sort the list of triplets and merge consecutive updates of the same entries. This step, sometimes referred to as pruning, is typically performed after the matrix construction has been completed. However, even after pruning, accessing entries remains slow compared to other storage formats.

By comparison, compressed sparse formats provide fast access to not only individual nonzero entries but entire rows (or columns, or blocks, ...), which is particularly useful in a number of tasks. However, inserting new nonzeros is more complicated since it requires shifting large contiguous arrays in memory, with possible (expensive) reallocation required. Consequently, adding two compressed sparse matrices with *different* sparsity patterns is not particularly efficient.

2.3.1 Efficient multiplication

Thanks to convenient access to sparsity structures of rows (or columns), compressed sparse formats enable significant speedups in basic linear algebra operations like matrix-vector or matrix-matrix multiplication. To illustrate the possible speedup, let us consider the task of multiplying two square sparse matrices $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times n}$.

If A is stored in the CSR format and B is stored in the CSC format, we can quickly access rows of A and columns of B and perform the multiplication $C = AB$ using the standard scheme, where $C_{i,j}$ is inner the product of the row vector of $A_{i,:}$ and the column vector $B_{:,j}$. Whether the result $C_{i,j}$ is nonzero is not a priori known, and hence we must perform n^2 inner products. The overall complexity is $O(n^3)$[2].

For comparison, consider the situation when both A and B are stored in the CSC format, allowing convenient access to the sparsity structures of their columns. Then, the matrix-matrix multiplication can be performed as n independent linear combinations of columns of A. Specifically, to get the j-th column of C, we compute a linear combination of columns of A with the coefficients in $B_{:,j}$. The speedup follows from the fact that if B is sparse, then $\mathtt{nnz}(B_{:,j})$ is on average small, and we know the exact positions of the few nonzero entries in $B_{:,j}$. The linear combination can then be performed by summing $\mathtt{nnz}(B_{:,j})$ dense vectors of length n, with complexity $O(n \cdot \mathtt{nnz}(B_{:,j}))$. The total complexity of the sparse matrix-matrix multiplication is then $O(n \cdot \sum_j \mathtt{nnz}(B_{:,j})) = O(n \cdot \mathtt{nnz}(B))$.

Analogously, the complexity of sparse matrix-matrix multiplication with both A and B in the CSR format is $O(n \cdot \mathtt{nnz}(A))$.

[2] Sparse inner products are typically most efficient when performed using dense array operations: the compressed entries of a sparse vector are ”scattered” to a dense vector with many zeros. The complexity of the dense inner product is $O(n)$.

3. Embarrassingly Shallow Autoencoder

Shallow autoencoders have recently gained significant attention in the recommender system community. In this chapter, we describe one of the most highly acclaimed shallow autoencoder models: Embarrassingly Shallow AutoEncoder (in Reverse order: EASE$^{\text{R}}$) proposed by Steck [2019a]. Despite its simplicity, the model was shown to outperform deep nonlinear models on several popular datasets, achieving new state-of-the-art performance. We explain the model's concept and its strong theoretical advantages, but also its fundamental limitation - costly scaling to domains with large item sets. Finally, we discuss previous attempts at solving this issue.

3.1 Model definition

The situation in which we derive the model assumes the training data are given in the form of a user-item matrix $X \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{I}|}$. The matrix is typically large, *sparse* (it has relatively few nonzero entries), and overdetermined ($|\mathcal{U}| >> |\mathcal{I}|$). Architecturally, EASE$^{\text{R}}$ is a neural network with no hidden layers and no activation, i.e., a linear model $f : \mathbb{R}^{|\mathcal{I}|} \to \mathbb{R}^{|\mathcal{I}|}$. Its parameters are given by a square matrix $\widehat{B} \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{I}|}$.

During the inference, the model predicts ratings as $\vec{r}^T = \vec{u}^T \widehat{B}$, where $\vec{u}$ is the input vector of the user's feedback. To elaborate, EASE$^{\text{R}}$ computes the predicted rating as a simple linear combination of rows of $\widehat{B}$. Moreover, when $\vec{u}$ is sparse, this matrix-vector product combines only a few rows. This simplicity results in quick inference, which is advantageous or even required in many practical applications.

Formally, EASE$^{\text{R}}$ solves the constrained optimization problem

$$\min_B \|X - XB\|_F^2 + \lambda\|B\|_F^2 \text{ s.t. } \operatorname{diag}(B) = \vec{0}, \tag{3.1}$$

where X is the interaction matrix, B is the learned matrix of the model weights, and λ is an L2 regularization hyperparameter. By minimizing the *reconstruction loss* $\|X - XB\|_F^2$, the model adjusts its weight to operate similarly to the identity function of the feature space of user ratings. Essentially, it learns to return a vector similar to the input vector. The constraint on the diagonal entries was first introduced by Ning and Karypis [2011] to prevent the convergence to the trivial solution $\widehat{B} = I$, a typical problem of sparse autoencoders. Thanks to the constraint, the self-similarity of items is prohibited, and the model is forced to generalize when reproducing the input and learn similarities with other items.

3.1.1 Closed-form solution

Thanks to the choice of reconstruction loss $\|X - XB\|_F^2$ and the regularization loss $\|B\|_F^2$, the constrained optimization problem (3.1) is convex, which allows us to express its solution analytically.

In the first step, we transform the constrained optimization problem into an unconstrained one. All constraints in the problem (3.1) are equality constraints. We introduce a vector of Lagrange multipliers $\vec{\mu} \in \mathbb{R}^{|\mathcal{I}|}$ and form the Lagrangian

$$\mathcal{L} = \|X - XB\|_F^2 + \lambda \|B\|_F^2 + 2\vec{\mu}^T \text{diag}(B). \tag{3.2}$$

To satisfy the necessary condition for minimization, we require the partial derivative of the Lagrangian $\mathcal{L}$ w.r.t. B to equal zero. Using the fact that $\|A\|_F^2 = \sum_{i,j} A_{i,j}^2$ and $(AB)_{k,j} = A_{k,:}B_{:,j} = \sum_i A_{k,i}B_{i,j}$, we rewrite

$$\begin{aligned}
\mathcal{L} &= \sum_{k,j} \Big(X_{k,j} - (XB)_{k,j} \Big)^2 + \lambda \sum_{i,j} B_{i,j}^2 + 2 \sum_i \mu_i B_{i,i} \\
&= \sum_{k,j} \Big(X_{k,j}^2 - 2X_{k,j}(XB)_{k,j} + (XB)_{k,j}^2 \Big) + \lambda \sum_{i,j} B_{i,j}^2 + 2 \sum_i \mu_i B_{i,i} \\
&= \sum_{k,j} \Big(X_{k,j}^2 - 2X_{k,j} \sum_i X_{k,i}B_{i,j} + (\sum_i X_{k,i}B_{i,j})^2 \Big) + \lambda \sum_{i,j} B_{i,j}^2 + 2 \sum_i \mu_i B_{i,i}
\end{aligned}$$

and express the partial derivative w.r.t. $B_{i,j}$ as

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial B_{i,j}} &= \sum_k \Big(-2X_{k,j}X_{k,i} + 2X_{k,i} \sum_l X_{k,l}B_{l,j} \Big) + 2\lambda B_{i,j} + 2\mu_i \cdot \mathbf{1}_{i=j} \\
\frac{1}{2}\frac{\partial \mathcal{L}}{\partial B_{i,j}} &= \sum_k X_{k,i} \Big((XB)_{k,j} - X_{k,j} \Big) + \lambda B_{i,j} + \mu_i \cdot \mathbf{1}_{i=j} \\
&= \sum_k X_{k,i} \Big(X(B - I) \Big)_{k,j} + \lambda B_{i,j} + \mu_i \cdot \mathbf{1}_{i=j} \\
&= \Big(X^T X (B - I) \Big)_{i,j} + \lambda B_{i,j} + \mu_i \cdot \mathbf{1}_{i=j} \\
&= \Big((X^T X + \lambda I) B \Big)_{i,j} - (X^T X)_{i,j} + \mu_i \cdot \mathbf{1}_{i=j}.
\end{aligned} \tag{3.3}$$

The partial derivative of the Lagrangian $\mathcal{L}$ w.r.t. B is zero if and only if all partial derivatives of $\mathcal{L}$ w.r.t. $B_{i,j}$ are zero. Hence, by Equation (3.3), the solution $\widehat{B}$ to the original optimization problem (3.1) is a solution to the linear system

$$(X^T X + \lambda I)B = X^T X - \text{diag}(\vec{\mu}),^1 \tag{3.4}$$

for some $\vec{\mu}$ which we need to select so that the constraint $\text{diag}(\widehat{B}) = \vec{0}$ holds.

The matrix $X^T X + \lambda I$ is called the *regularized data Gram matrix* or the *regularized Gramian* of the matrix X. The matrix is symmetric. Before continuing the derivation of the analytical solution, we need to prove the invertibility of the regularized data Gram matrix. At the same time, we will prove several related statements, which will be useful later in the thesis.

Definition 3.1. *A square matrix $A \in \mathbb{R}^{n \times n}$ is* positive definite *if and only if $\vec{x}^T A \vec{x} > 0$ for all $\vec{x} \neq \vec{0}$.*

Proposition 3.1. *For any $\lambda \in \mathbb{R}^+$ and any matrix $X \in \mathbb{R}^{m \times n}$, the matrix $X^T X + \lambda I$ is positive definite.*

[1] For a vector $\vec{v}$, $\text{diag}(\vec{v})$ denotes the square matrix with $\vec{v}$ on the main diagonal. For a matrix A, $\text{diag}(A)$ is the vector of the main diagonal entries.

Proof. The matrix X^TX is positive semi-definite, since $\vec{x}^TX^TX\vec{x} = \|X\vec{x}\|^2 \geq 0$. Moreover, $\vec{x}^T(\lambda I)\vec{x} = \lambda\|\vec{x}\|^2 > 0 \ \forall \vec{x} \neq \vec{0}$. Summing up the two inequalities, we obtain

$$\vec{x}^T(X^TX + \lambda I)\vec{x} = \vec{x}^T(X^TX)\vec{x} + \vec{x}^T(\lambda I)\vec{x} > 0 \ \forall \vec{x} \neq \vec{0},$$

which is the defining condition. □

The invertibility of $X^TX + \lambda I$ is a consequence of Proposition 3.1 and the following simple statement from linear algebra.

Proposition 3.2. *A symmetric positive definite (SPD) matrix $A \in \mathbb{R}^{n\times n}$ is invertible.*

Proof. By the Spectral Theorem (see e.g., Pinkham [2015]), all eigenvalues of A are positive, i.e. 0 is **not** an eigenvalue of A. Hence, the system $A\vec{x} = \vec{0}$ has no non-trivial solution, and so A is invertible. □

Corollary. For any $\lambda \in \mathbb{R}^+$ and any matrix $X \in \mathbb{R}^{m\times n}$, the matrix $X^TX + \lambda I$ is invertible.

Proof. Follows from Proposition 3.1 and Proposition 3.2. □

Lemma. *Let $A \in \mathbb{R}^{n\times n}$ be a symmetric invertible matrix. Then A^{-1} is symmetric.*

Proof. It holds that $(A^{-1})^T = (A^T)^{-1} = A^{-1}$. □

Proposition 3.3. *For any $\lambda \in \mathbb{R}^+$ and any matrix $X \in \mathbb{R}^{m\times n}$, the matrix $(X^TX + \lambda I)^{-1}$ is symmetric positive definite.*

Proof. By Proposition 3.1, $X^TX + \lambda I$ is symmetric positive definite, hence invertible by Corollary 3.1.1. Using the previous lemma, we get that $(X^TX+\lambda I)^{-1}$ is symmetric. Moreover, it follows from the Spectral Theorem for real symmetric matrices that $X^TX + \lambda I$ is orthogonally diagonalizable. Hence, we may write $X^TX + \lambda I = UDU^T$, where U is a real orthogonal matrix (i.e., U is square and $UU^T = I$) and D is a square diagonal matrix. The diagonal entries of D are the eigenvalues of $X^TX + \lambda I$. According to the Spectral Theorem for positive definite matrices, all eigenvalues of $X^TX + \lambda I$ are positive real numbers, which specifically implies that the diagonal entries in D are nonzero. Because of this, we may express the inverse of $X^TX + \lambda I$ as

$$(X^TX + \lambda I)^{-1} = (UDU^T)^{-1} = U^{-T}D^{-1}U^{-1} = UD^{-1}U^T,$$

where we used $U^{-1} = U^T$. This expression shows that $(X^TX + \lambda I)^{-1}$ is orthogonally diagonalizable with positive real eigenvalues, which by the Spectral Theorem means it is positive definite. □

Corollary 3.1.1 ensures the existence of a unique solution for the linear system 3.4. By substituting $\widehat{P} = (X^TX + \lambda I)^{-1}$, we can express the solution as

$$\begin{aligned}
\widehat{B} &= (X^TX + \lambda I)^{-1}(X^TX - \operatorname{diag}(\vec{\mu})) \\
&= \widehat{P}(\widehat{P}^{-1} - \lambda I - \operatorname{diag}(\vec{\mu})) \\
&= I - \lambda\widehat{P} - \widehat{P}\operatorname{diag}(\vec{\mu}) \\
&= I - \widehat{P}\operatorname{diag}(\lambda\vec{1} + \vec{\mu}). \qquad (3.5)
\end{aligned}$$

The vector of Lagrange multipliers $\vec{\mu}$, and hence also $\lambda\vec{1}+\vec{\mu}$, is determined by the constraint $\mathrm{diag}(\widehat{B})=\vec{0}$. Combining Equation (3.5) with the constraint yields

$$\begin{aligned}\vec{0}=\mathrm{diag}(\widehat{B})&=\mathrm{diag}(I)-\mathrm{diag}\Big(\widehat{P}\mathrm{diag}(\lambda\vec{1}+\vec{\mu})\Big)\\&=\vec{1}-\mathrm{diag}(\widehat{P})\odot(\lambda\vec{1}+\vec{\mu}),\end{aligned}\tag{3.6}$$

where $\odot$ denotes the elementwise product of vectors. We used the observation that multiplying $\widehat{P}$ by $\mathrm{diag}(\lambda\vec{1}+\vec{\mu})$ from the right is the same as scaling the columns of $\widehat{P}$ by the corresponding entries of $\lambda\vec{1}+\vec{\mu}$. Equation (3.6) allows us to express the vector $\lambda\vec{1}+\vec{\mu}$ in terms of $\widehat{P}$ (i.e. using only X and λ) as

$$\lambda\vec{1}+\vec{\mu}=\vec{1}\oslash\mathrm{diag}(\widehat{P}),\tag{3.7}$$

with $\oslash$ denoting the elementwise vector division. This operation is well-defined if and only if $\widehat{P}_{i,i}\neq 0\ \forall i\in\{1,\ldots,|\mathcal{I}|\}$. This is satisfied because $\widehat{P}$ is positive definite (Proposition 3.3). By definition, $\vec{x}^T\widehat{P}\vec{x}>0$ for all $\vec{x}\neq\vec{0}$, which for $\vec{x}=\vec{e_i}$ (where $\vec{e_i}$ is the i-th vector of the canonical basis) yields $\vec{e_i}^T\widehat{P}\vec{e_i}=\widehat{P}_{i,i}>0$. Substituting the expression (3.7) into Equation (3.5) yields the desired the closed-form solution:

$$\widehat{B}=I-\widehat{P}\mathrm{diag}(\vec{1}\oslash\mathrm{diag}(\widehat{P}))\tag{3.8}$$

To avoid confusion, we emphasize that $\mathrm{diag}(\widehat{P})$ is a vector and $\mathrm{diag}(\vec{1}\oslash\mathrm{diag}(\widehat{P}))$ is a square diagonal matrix. The multiplication of $-\widehat{P}$ from the right-hand side by the diagonal matrix $\mathrm{diag}(\vec{1}\oslash\mathrm{diag}(\widehat{P}))$ is equivalent to dividing the columns of $-\widehat{P}$ by the corresponding entries of the vector $\mathrm{diag}(\widehat{P})$. It follows that $\mathrm{diag}(-\widehat{P}\mathrm{diag}(\vec{1}\oslash\mathrm{diag}(\widehat{P})))=\vec{1}$ and we may express the learned weights as

$$\widehat{B}_{i,j}=\begin{cases}0 & \text{if } i=j,\\ -\frac{\widehat{P}_{i,j}}{\widehat{P}_{j,j}} & \text{otherwise.}\end{cases}\tag{3.9}$$

3.1.2 Properties of weights

Equation (3.9) shows that the off-diagonal entries are determined by the matrix $\widehat{P}$. Note also that $\widehat{B}$ is generally asymmetric as it is obtained from a symmetric matrix $\widehat{P}$ by scaling from one side only.

Residual connection Let us elaborate on the structure of $\widehat{B}$ as a layer in a neural network. Denoting $\widehat{B_{diag}}=-\widehat{P}\mathrm{diag}(\vec{1}\oslash\mathrm{diag}(\widehat{P}))$ the matrix of weights before applying the zero diagonal constraints, we may write

$$\widehat{B}=\widehat{B_{diag}}+I.$$

We see from the above expression that instead of viewing EASE$^{\text{R}}$ as a linear model with weights $\widehat{B}$ and no bias vector, we may understand it as a linear model with weights $\widehat{B_{diag}}$ and no bias with added *residual connection* from the input to the output:

$$\vec{r}^T=\vec{u}^T\widehat{B}=\vec{u}^T(\widehat{B_{diag}}+I)=\vec{u}^T\widehat{B_{diag}}+\vec{u}^T\tag{3.10}$$

Full-rank component *Full-rank* modeling was shown to benefit recommendation quality in CF tasks, especially in domains with extensive item sets and prominent long-tail (see Steck [2019b] and Steck and Liang [2021]), where low-dimensional latent representations bottleneck the network's expressiveness. Positively, EASE$^{\text{R}}$ includes a full-rank component, as shown in the following.

Proposition 3.4. $\text{rank}(\widehat{B_{diag}}) = |\mathcal{I}|$.

Proof. We know that $\text{rank}(\widehat{B_{diag}}) = \text{rank}(-\widehat{P}\text{diag}(\vec{1} \oslash \text{diag}(\widehat{P})))$. Matrix $\widehat{P}$ is the inverse of the invertible matrix $X^TX + \lambda I$ (see Corollary 3.1.1) and hence $-\widehat{P}$ is invertible. Matrix $\text{diag}(\vec{1} \oslash \text{diag}(\widehat{P}))$ is diagonal with nonzero entries, and therefore it is invertible. A product of invertible matrices is invertible. Finally, an invertible matrix has full rank by the Rank-Nullity Theorem (see, e.g., Pinkham [2015]) because its kernel is trivial. $\square$

This observation partially explains the accuracy gains of EASE$^{\text{R}}$ over the previous state-of-the-art. We refer to Steck and Liang [2021] and references therein for further discussion on the importance of combining full-rank and higher-order (non-linear) modeling in collaborative filtering tasks.

3.2 Model training

The training procedure of EASE$^{\text{R}}$ follows from the closed-form solution (Equation (3.8)) of the optimization problem (3.1) derived in the previous section. We summarize the training procedure in Algorithm 1.

Algorithm 1 The training procedure of EASE$^{\text{R}}$ (Steck [2019a])

input user-item interaction matrix $X \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{I}|}$, L2 regularization $\lambda \in \mathbb{R}^+$
1: $A \leftarrow X^TX + \lambda I$ ▷ the regularized data Gram matrix
2: $\widehat{P} \leftarrow A^{-1}$
3: $\widehat{B_{diag}} \leftarrow \widehat{P}\text{diag}(\vec{1} \oslash \text{diag}(\widehat{P}))$ ▷ scale columns of $\widehat{P}$
4: $\widehat{B} \leftarrow \widehat{B_{diag}} + I$ ▷ zero out the diagonal entries
return $\widehat{B}$

The time complexity of Algorithm 1 is dominated by Step 2, in which the inverse of $X^TX + \lambda I$ is computed. Asymptotically, the complexity of the inversion is about $O(|\mathcal{I}|^{2.3755})$ using the Coppersmith-Winograd algorithm (Coppersmith and Winograd [1990]). This computational complexity considerably lower than the cost of training SLIM (Ning and Karypis [2011]) - the spiritual predecessor of EASE$^{\text{R}}$ - and its variants, which solve $|\mathcal{I}|$ independent (parallelizable) regression problems with total complexity $O(|\mathcal{I}|(|\mathcal{I}|-1)^{2.3755})$. Of course, for efficient inverse computation, the matrix must fit into memory. Otherwise, the computation will be significantly inhibited by data transfers.

The complexity of the dominant Step 2 does not depend on the number of users $|\mathcal{U}|$ or the number of user-item interactions $\texttt{nnz}(X)$. This independence is instrumental in (common) situations when the number of users in the training dataset is much greater than the number of items, i.e., $|\mathcal{U}| >> |\mathcal{I}|$. In such case, Step 1 - which may be performed in the pre-training phase using big data workflows - aggregates and compresses the input data into a matrix of smaller size. We will discuss the meaning of this aggregation in the upcoming section. On the other hand, the aggregation increases the density of data, which may be costly (in terms of storage) when $|\mathcal{I}|$ is too large.

Steps 3 and 4 are in-place operations, i.e., they are performed almost instantly in any realistic scenario. Moreover, Step 4 is optional, and we may include a residual connection connecting the input to the output, as discussed earlier.

3.3 Interpretation and advantages

Encoder meets decoder Despite its name, EASE^R, technically, differs from most autoencoder architectures that use separate encoder and decoder parts. Standard autoencoders first encode the input into a vector in latent feature space. This encoding, called the *latent representation*, is available during the computation. A decoder then decodes the latent vector to create a prediction. EASE^R has no hidden layers. Instead, the layer $\widehat{B}$ both encodes the input into a latent representation and decodes the latent representation to produce an output. As a result, the latent representation of the input is not explicitly available in EASE^R, but it is possible to uncover it by modifying the model to use the matrix $\widehat{B}$ in a factorized form[2].

Aggregate data from many users Because $\widehat{P} = (X^T X + \lambda I)^{-1}$, we see from Equation (3.5) that $X^T X$ provides sufficient statistics for estimating the weight matrix $\widehat{B}$. This observation has two positive consequences. Firstly, if $|\mathcal{U}| >> |\mathcal{I}|$, we may compress the training data by aggregating $X \to X^T X$ before training (perhaps in a big data pipeline) and this compression does not lose information required for model training. Secondly, this aggregation helps battle data sparsity. The main idea here is that the statistic $(X^T X)_{i,j}$ is aggregated through all users in the dataset. Increasing the number of users reduces the uncertainty of this statistic. Therefore, for an accurate estimate of the weights $\widehat{B}$, sparsity of X can be compensated sufficiently increasing the number of users (see the original paper by Steck [2019a] for more details). In large-scale recommendation scenarios such as e-commerce, this is invaluable.

3.3.1 Similarity through user chains

Unlike many neighborhood-based CF methods, which compute (heuristical) item-item relations or similarity as a (potentially rescaled) data Gram matrix $X^T X$, Steck [2019a] shows (under certain assumptions) that the conceptually correct similarity matrix is, in fact, the inverse of $X^T X$. The rest of the section provides a visual explanation of this statement.

CF methods that estimate the item-item relations as the data Gram matrix $X^T X$ perform a specific user feedback aggregation. For a pair of items i, j whose relation we want to estimate, a system of this kind finds all users who interacted with both items. In other words, it identifies **all paths of length exactly 2**

[2] Different factorizations would yield representations inside different feature spaces.

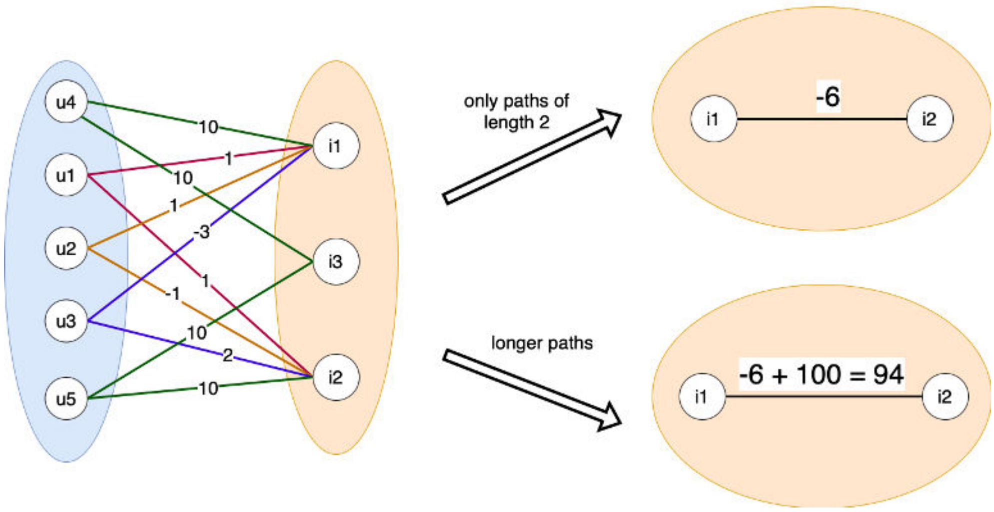


Figure 3.1: Aggregation through chains of users. When computing the item-item similarity using only user pairwise sentiment (how much users like or dislike both items in question), it is easy to miss relevant information. For items `i1` and `i2`, pairwise sentiments of users `u1` (+1 and +1 respectively) and `u2` $(+1, -1)$ cancel out, leaving only the pairwise sentiment of user `u3` $(-3, +2)$ and modeling the similarity of `i1` and `i2` as negative. This aggregation is oblivious to the strong preference of the user `u4` towards items `i1` and `i3` (+10, +10) and the strong preference of the user `u5` towards `i3` and `i2` (+10, +10). Item `i3` links the two pairwise sentiments into a chain and reveals the strongly positive relation between items `i1` and `i2` in this example.

between item vertices in the weighted unoriented bipartite graph

$$\mathcal{G}_X = (\mathcal{V}_X^{\mathcal{U}} \bigcup \mathcal{V}_X^{\mathcal{I}}, \mathcal{E}_X),\ \mathcal{E}_X \subseteq \mathcal{V}_X^{\mathcal{U}} \times \mathcal{V}_X^{\mathcal{I}},\ w : \mathcal{E}_X \to \mathbb{R}$$

(where vertices in $\mathcal{V}_X^{\mathcal{U}}$ represent the users and vertices in $\mathcal{V}_X^{\mathcal{I}}$ represent the items) given by the user-item matrix X:

$$(v_i, v_j) = e_{i,j} \in \mathcal{E}_X \iff X_{i,j} \neq 0, \text{ and } w(e_{i,j}) = X_{i,j}.$$

For each path of length 2 between a pair of item vertices, the weights (i.e., the observed user-item feedback) of the pair of edges are multiplied, and the results for all paths are summed together. Conceptually, this aggregation sums something that can be viewed as a *pairwise sentiment* - if and "how much" a user likes/dislikes both items in a pair, or likes one item and dislikes the other. An example of such aggregation is shown in Figure 3.1, where users `u1`, `u2`, `u3` interacted with both items `i1` and `i2`. For example, we see that while both users `u1`, `u2` provide positive feedback 1 for item `i1`, one of them assigned item `i2` a positive feedback 1 while the other assigned it a negative feedback -1. Both like one item of the pair while having opposite opinions about the second item - their pairwise sentiment towards the pair `i1`, `i2` is orthogonal and cancels out.

This method of aggregation has limited scope. Fundamentally, two main issues result in worse recommendation quality, most noticeably decreased diversity and subpar recommendations for users with niche interests.

Data sparsity A path of length 2 between items `i1` and `i2` exists if and only if there exists a user who interacted with both items `i1` and `i2`. Interactions of these users represent a single entry in the matrix $X^T X$ (alternatively, an edge in the weighted graph $\mathcal{G}_{X^T X}$ of the adjacency matrix $X^T X$). We call these entries (or edges) *direct data* about the relation between an item pair.

A practical observation is that direct data for a random item pair is unlikely to exist on domains with vast item sets. To see why, one needs to realize that in a dataset with $|\mathcal{I}|$ items, there are $\binom{|\mathcal{I}|}{2}$ different item pairs - this number grows approximately quadratically in $|\mathcal{I}|$. It is apparent that with the growing item set size, gathering enough feedback is increasingly challenging. Even collecting enough feedback for 10 000 to 100 000 items could be problematic (depending on the number of users and the amount of feedback they give). This effect is typically even more substantial due to non-uniform item interaction distribution, as items from the long tail are viewed less often.

Imagine a situation when a user with niche interests visits our website. This user has previously positively rated the long-tail item `i1`, and if we could read his mind, we would know that he also very much likes the long-tail item `i2`. Of course, we would want to recommend this item, but our recommender system's view is limited. Based on positive interaction with item `i1`, it can only recommend item `i2` if previously someone else liked both items `i1` and `i2`. If our item catalog is vast, such a user likely does not exist. In a better case, a few other users have seen both items. Then, the relation is estimated, but only with low certainty.

Unawareness of long-distance relations Aggregation via paths of length 2 may be oblivious to important information. In the toy situation illustrated in Figure 3.1, such aggregation estimates the relation between items `i1` and `i2` only using feedback from users `u1`, `u2` and `u3`, which is directly available. However, it misses the fact that both users `u4` and `u5` revealed strong positive sentiment toward item `i3`, while user `u4` very much liked item `i1` and user `u5` showed strong positive sentiment toward item `i2`. This is an example of a chain of user sentiment. If two users agree in preference on an item, it is reasonable to assume one user would like other items the second user liked, and vice versa. Chains of user sentiment may be arbitrarily long, and they are crucial for helping with the data sparsity problem mentioned in the previous paragraph. When direct data is unavailable between items `i1` and `i2`, a recommender system working with sentiment chains might still find chains of feedback by connecting multiple users and recommend item `i2` based on them. In other situations, as in Figure 3.1, incorporating the information from a longer chain might change the estimated relationship between the items. Where direct data (perhaps infrequent and noisy) may suggest a negative relation or dissimilarity, longer sentiment chains could reveal that the relation between the two items is, in fact, positive.

In domains with extensive item sets, where both users and items may have only a few interactions, producing accurate and diverse recommendations may be a complicated task, especially for methods like various neighborhood-based approaches that only consider short paths in the bipartite graph $\mathcal{G}_X$. Compared with that, EASE$^\text{R}$ takes into account much more information in the form of arbitrarily long chains of user sentiment. One possible way to see this is using a known relationship between the inverse matrix and the minors of the original matrix. Using the standard notation where $M_{i,j}$ is the (i,j)-th minor of a square matrix A of dimension n (i.e., the determinant of the submatrix obtained by dropping the i-th row and the j-th column of A) and $\mathrm{adj}(A) = \left((-1)^{i+j} M_{j,i}\right)_{1 \le i,j \le n}$ is the adjugate matrix of A, the inverse of A can be expressed as

$$A^{-1} = \frac{1}{\det(A)} \mathrm{adj}(A).$$

We see that individual entries of A^{-1} are proportional to their respective minors:

$$(A^{-1})_{i,j} = \pm \frac{1}{\det(A)} M_{j,i} \propto M_{j,i}. \tag{3.11}$$

Substituting $A = X^T X + \lambda I$, the above relation allows us to interpret the modeled similarity of items i and j in terms of paths in the user-item graph $\mathcal{G}_X$. The minor $M_{j,i}$ is a sum of terms. Every term represents a product of $n-1$ entries of $X^T X + \lambda I$ from different rows and columns. Since the (k,l)-th entry of $X^T X + \lambda I$ aggregates pairwise sentiment from all users that interacted with both items k and l, the terms in the minor are products of pairwise sentiments (paths of length 2 in $\mathcal{G}_X$). Moreover, a term is nonzero if and only if the entire $(n-1)$-tuple of selected entries of $X^T X + \lambda I$ is nonzero, and hence $M_{j,i}$ ignores "zero" paths[3]. It is easy to verify that every term[4] in $M_{j,i}$ decomposes into

- a *multipath*[5] between i and j in $\mathcal{G}_X$, and
- one or more independent *multicycles* in $\mathcal{G}_X$, each composed of a subset of the remaining items and sets of users connecting pairs of items in the cycle. We allow a trivial multicycle composed of a single item and the set of users who interacted with this item.

This decomposition reveals how EASE$^{\mathrm{R}}$ models the similarity of items: every nonzero term in $M_{j,i}$ represents an aggregation of a set of sentiment chains[6] weighed by the aggregate sentiment of users toward the remaining items (these weights are later normalized by the determinant, which aggregates the "total sentiment" of users toward all items, see Equation (3.11)). This description makes it clear that EASE$^{\mathrm{R}}$ models the similarity of items i and j as the aggregation over **all chains of pairwise sentiment between i and j** (i.e., all paths between i and j in $\mathcal{G}_X$). Apart from its comparably lower computational complexity, this ability to model item-item relations via long chains of users is, in our opinion, the most significant advantage of EASE$^{\mathrm{R}}$.

As a possible direction for future work, we note that the Cholesky decomposition of the SPD matrix $X^T X + \lambda I$ offers another viewpoint at the similarity model. Denoting $LL^T = X^T X + \lambda I$, we may express the (i,j)-th entry of $(X^T X + \lambda I)^{-1}$ as

$$\left((X^T X + \lambda I)^{-1}\right)_{i,j} = e_i^T (L^{-T} L^{-1}) e_j = (L^{-1})_{:,i}^T (L^{-1})_{:,j}.$$

[3]A "zero" path contains a pair of consecutive items k, l with zero aggregate feedback, i.e., $(X^T X + \lambda I)_{k,l} = 0$. This occurs due to cancelation or, more commonly, missing direct interaction data – when no user has seen both k and l.

[4]A term in $M_{j,i}$ corresponds to a permutation of n elements with one edge removed.

[5]Because a pair of items is connected through a *set* of users.

[6]These chains connect the same items but through different users.

The above relation shows that the (i, j)-th inverse entry – i.e., the modeled similarity of items i and j – is the inner product of columns i and j of L^{-1} (it is, therefore, closely related to the cosine similarity of the two columns). It is possible to trace the nonzero entries in the factor L to their origins in the matrix $X^T X + \lambda I$ using the associated elimination tree (see Appendix A.1 for a definition) and also use the same tree to learn how the entries of L^{-1} are created from L (we discuss how this tracing works in Appendix A.2.2).

3.4 Expensive scaling

Nowadays, it is not uncommon to have datasets with millions of items, and the sizes will likely continue to grow. Large item sets often obstruct the adoption of state-of-the-art (deep learning) models in production systems. The complexity of training a deep neural network for the task may be very high, as shown, e.g., in Steck [2019b], where the variational autoencoder MULT-VAE$^{\text{PR}}$ (Liang et al. [2018]) required more than 4 hours to train on a dataset with mere 41,140 items. For this reason, large CF systems are often forced to fallback to much simpler methods, such as ALS (Hastie et al. [2014]) or user-oriented or item-oriented neighborhood-based approaches (e.g.,Resnick et al. [1994] and Sarwar et al. [2001], respectively). As discussed earlier, low-rank models like ALS internally compress information into a low-dimensional latent space. This compression may cause the loss of subtle information about the long tail. Unfortunately, long-tail interactions are fundamental for diverse personalized recommendations in domains with vast item sets.

A solution to this problem is to use full-rank models instead, but before EASE$^{\text{R}}$ was introduced, full-rank models like SLIM (Ning and Karypis [2011]) had been costly to train. EASE$^{\text{R}}$ considerably improved the time complexity for training over SLIM, but inverting a large matrix (Step 2) can still be prohibitively slow for production use. Fortunately, algorithms for inversion are typically well parallelizable, and this issue can be avoided by running the training on larger hardware.

Density of weights More importantly, the learned weight matrix may require too much memory, even if the data-Gram item-item matrix $A = X^T X + \lambda I$ is very sparse. Specifically, a well-known result by Duff et al. [1988] states that the inverse of an **irreducible** matrix is structurally fully dense - even when the original matrix is sparse (see also Gilbert and Liu [1993], Scott and Tůma [2023]).

To understand why this is a problem, recall that any real matrix A represents an adjacency matrix of a weighted oriented graph $\mathcal{G}_A$, and any nonsingular matrix has an (in a sense unique) Frobenius normal form PAP^T, which symmetrically permutes the matrix A to reveal its **irreducible blocks**. There exists a one-to-one correspondence between the **strongly connected components** (where every vertex is reachable from every other vertex via an oriented path) of $\mathcal{G}_A$ and the irreducible blocks of the Frobenius normal form PAP^T. The correspondence can be composed as follows:

1. The permutation p corresponding to the permutation matrix P induces an isomorphism of graphs $\mathcal{G}_A$ and $\mathcal{G}_{PAP^T}$.

2. Vertices $v_{p(i)}, v_{p(j)}$ of $\mathcal{G}_{PAP^T}$ are in the same strongly connected component if and only if $p(i), p(j)$ are column indices in the same irreducible block of matrix PAP^T.

In other words, the irreducibility of a matrix can be understood in terms of the strong connectivity of its graph. A large irreducible block in PAP^T exists if and only if a large strongly connected component exists in $\mathcal{G}_A$. By definition, v_i and v_j belong to the same strongly connected component of the item-item network $\mathcal{G}_{X^TX}$ given by the adjacency matrix X^TX exactly when there exist oriented paths from v_i to v_j and from v_j to v_i in $\mathcal{G}_{X^TX}$. Since X^TX is symmetric, every path can be viewed as if oriented in both directions, and the strong connectivity reduces to connectivity. Moreover, adding diagonal entries to X^TX does not affect the existence of paths between vertices in $\mathcal{G}_{X^TX}$, i.e., vertices v_i to v_j are connected by an (unoriented) path in $\mathcal{G}_{X^TX}$ exactly when they are connected in $\mathcal{G}_{X^TX+\lambda I}$. Finally, paths in $\mathcal{G}_{X^TX+\lambda I}$ are transitive: if there exists a path connecting v_i and v_j and a path connecting v_j and v_k, then v_i and v_k are connected by a path. While a direct edge connecting an arbitrary pair v_i and v_k in $\mathcal{G}_{X^TX+\lambda I}$ might not exist, a path between v_i and v_k is more likely; it only needs some other vertex v_j to which both v_i and v_k connect via a path.

This is the central element driving the issue. Large connected components are likely in the practical applications, as discussed here. Firstly, for an edge v_i and v_j in $\mathcal{G}_{X^TX+\lambda I}$ to exist, there must exist a user who interacted with both items, i.e., there must exist a path of length 2 connecting the two items in the bipartite graph $\mathcal{G}_X$. However, any path connecting the two items in $\mathcal{G}_X$ yields a path between v_i and v_j in $\mathcal{G}_{X^TX+\lambda I}$. In other words, it only needs a chain of pairwise sentiments that start with item i and end with item j, not a "direct" pairwise sentiment for i and j. By gathering more user feedback, we are adding (possibly new) edges to the graph $\mathcal{G}_X$, and the odds of two items being connected by a path grow. Moreover, in real-world applications, the distribution of user-item interactions is not uniform. Small sets of popular items often garner interactions from numerous users. These popular items boost connectivity of $\mathcal{G}_X$ by creating many paths of length more than 2. Consequently, large connected components will likely emerge in the item-item network $\mathcal{G}_{X^TX+\lambda I}$.

To summarize, the wide span of aggregation used by EASE$^\text{R}$ has a trade-off. While considering longer paths in $\mathcal{G}_X$ (i.e., chains of pairwise sentiment) adds significant additional information to the model, it also results in high memory requirements in domains with vast item sets. The difference between the size of A and the resulting model size may be staggering. Therefore, we may expect the model size of EASE$^\text{R}$ to be problematic in large collaborative filtering tasks. If even a single sizeable irreducible component exists in A - which is very likely in practice - the matrix A^{-1} will be quite dense, and the resulting model may get very large. To put the problem with density in resource perspective, for a dataset with 1 million items, the trained model EASE$^\text{R}$ could have up to 1 *trillion* parameters and require up to 4 TB of memory (using `float32`). While this may be tolerable for shorter training periods, deploying such a large model for long-term production inference is unjustifiably expensive, as the entire weight matrix must fit in memory for inference.

3.4.1 Improvements

We end this chapter by describing proposed modifications EASE^R which aim to reduce the model's memory requirements.

Low-rank dense approximation Model ELSA (Vančura et al. [2022]) is based on a dense low-rank approximation of the matrix $\widehat{B_{diag}}$. The training procedure uses a gradient descent approach to optimize model weights. This modification asymptotically improves training complexity (both time and memory) over the basic model EASE^R. Moreover, ELSA even performs slightly better than EASE^R on smaller datasets, likely due to the denoising effect of the low-rank approximation[7], which serves as additional regularization. However, as discussed earlier in this section, low-rank factorization may negatively impact the model performance (see, e.g., Steck [2019b], Steck and Liang [2021]), especially in domains with an extensive long tail, where even subtle dependencies may be crucial.

Full-rank sparse approximation Another possibility is to use a sparse full-rank approximation of $\widehat{B_{diag}}$. Steck [2019b] explained that the item-item network $\mathcal{G}_{(X^TX+\lambda I)^{-1}}$ can be understood as a Markov Random Field (hence the name MRF). The author presented a novel algorithm for learning a sparse approximation of the Markov Random Field based on recent research in sparse inverse covariance estimation (e.g., Banerjee et al. [2008], Friedman et al. [2007]). The MRF method estimates the inverse matrix A^{-1} by inverses of submatrices corresponding to dominant item clusters in the network that are interconnected on their interfaces. This approach can be classified as a type of *overlapping domain decomposition* where the interfaces represent overlaps (see, for example, a general algorithm-based description in a classical text by Smith et al. [1996]).

The method allows for a trade-off between model size, training time and recommendation accuracy. Crucially, the MRF method was shown to closely match the performance of dense EASE^R even at high compression rates (with up to 1000× smaller model than EASE^R on a relatively dense dataset). This efficiency demonstrates the potential of scaling EASE^R-like methods to domains with very large number of items, where the corresponding item-item matrix $X^TX + \lambda I$ is often very sparse. Unfortunately, the training of MRF may still require significant resources. The sparsity pattern in MRF is estimated from a *dense* item correlation matrix, which can be very large[8]. Moreover, even inverting small submatrices may yield memory inefficiency if there are many of them or if their computation stems from the same large and dense item-item matrix. Then, the matrix must be kept in memory throughout the entire training procedure (expensive), or loaded by parts (slow).

[7]Low-rank approximation via the truncated singular value decomposition suppresses high-frequency noise, see Hansen [2010].

[8]The size is quadratic in the number of items. E.g. for Amazon Books dataset (Ni et al. [2019]) with 91 599 items, the matrix contains 8.4B entries (33.6GB using `float32`) - and item sets can be *much* larger.

4. Sparse approximate inverse

Similarly to Steck [2019b], we aim to modify the concept of EASE^R by finding a sparse approximation of its weight matrix. To accomplish this nontrivial task, we turn to numerical mathematics. This chapter reviews the use of sparse approximate inverses in numerical computations, where they serve as an important component in efficient solvers for large linear systems. We then describe various possible methods for computing a sparse approximate inverse. The primary reference here is A comparative study of sparse approximate inverse preconditioners by Benzi and Tůma [1999].

4.1 Motivation

Being able to efficiently and reliably solve large linear systems $A\vec{x} = \vec{b}$ is crucial for a great number of applications in science and engineering. Systems arising from such domains often belong to a specific category: they are square, very large in dimension and *sparse*. For such systems, the use of direct methods is often infeasible due to high memory requirements. Instead, modern solvers typically utilize more memory-efficient iterative methods based on Krylov subspaces (see e.g., a textbook by Liesen and Strakoš [2012]).

The speed of convergence for Krylov subspace methods depends on the problem's spectral properties, and in some cases, convergence to the desired solution may be prohibitively slow. In such situations, we may employ **preconditioning** to improve the problem's properties. The basic idea of preconditioning is to find a suitable transformation of the original problem to a problem with better numerical properties so that the selected iterative method converges faster. For example, we may transform the original system to a form $MA\vec{x} = M\vec{b}$, where the matrix M is the *preconditioner*; there are more possible ways how to apply preconditioning. Over the years, different concepts for construction of preconditioners were developed, with incomplete LU (or Cholesky) factorization being perhaps the most prominent one. We will focus on another important class, which is based on approximate inversion.

As the name suggests, preconditioning based on approximate inversion uses the preconditioner matrix M which approximates the inverse of the coefficient matrix A. The core idea is that if $M \approx A^{-1}$, then applying the preconditioner $A\vec{x} = \vec{b} \rightarrow MA\vec{x} = M\vec{b}$ should yield $MA \approx I$. In other words, the coefficient matrix of the transformed system should be close to the identity, and hence it should be easy to recover the solution $\vec{x}$. Note that conceptually, M need not approximate A^{-1} in the sense that $\|M - A^{-1}\|$ is small (for some choice of the norm). Rather, we need that $\|I - MA\|$ is small, i.e. the left-hand side operator M needs to act similarly to A^{-1}.

Unlike polynomial preconditioners which also approximate A^{-1} but only implicitly, sparse approximate inverse techniques explicitly compute and store the preconditioner matrix $M \approx A^{-1}$. These techniques became of interest for research of algebraic preconditioners because of strong potential for parallel implementation. Another motivation was providing an alternative in situations when other preconditioning methods fail. Incomplete factorization techniques can fail (e.g., on indefinite matrices) due to some form of instability, either in the incomplete factorization phase (small pivots), or in the back substitution phase, or both; see Chow and Saad [1997]. Approximate inverse techniques are typically less prone to these problems, and hence provide an important complement to other preconditioning methods.

Remark. Approximate inverse preconditioners rely on the assumption that it is possible to find a good approximation of the inverse A^{-1} of the coefficient matrix A. This is not evident since the inverse of a sparse matrix is often dense. More precisely, the inverse of an irreducible sparse matrix A is structurally dense (see Duff et al. [1988], Gilbert and Liu [1993], Scott and Tůma [2023]), meaning it is always possible to assign numerical values to the sparsity pattern of an irreducible matrix A in such a way that all entries of the inverse will be nonzero. Nevertheless, many of the entries in A^{-1} are often small in magnitude, making finding the sparse approximate inverse possible. Much of the recent research has been devoted to the problem of selecting the "important" entries of A^{-1} automatically.

The following sections summarize common approaches for finding sparse approximate inverses of a square nonsingular matrix $A \in \mathbb{R}^{n\times n}$. We follow the survey by Benzi and Tůma [1999] and divide the methods into three groups:

1. methods based on Frobenius norm minimization,
2. factorized sparse approximate inversion methods, and
3. methods based on incomplete factorization and subsequent approximate inversion of the factors.

4.2 Frobenius norm minimization methods

Methods based on Frobenius norm minimization find a sparse approximation M of A^{-1} as the solution of the constrained optimization problem

$$\min_{M\in\mathcal{S}}\|I - AM\|_F^2, \tag{4.1}$$

where $\mathcal{S}$ is the set of sparse matrices. This approach was first proposed by Benson [1973]. Denoting $\vec{e_j}$ the j-th vector of the canonical basis (or the j-th column of the identity matrix I), we may write

$$\|I - AM\|_F^2 = \sum_{j=1}^{n} \|\vec{e_j} - A\vec{m}_j\|_2^2.$$

We see that the optimization problem (4.1) decomposes into n independent least squares problems with the same coefficient matrix A, subject to sparsity constraints. The above approach is useful for finding a right approximate inverse, i.e., it finds an approximation which when applied from the right acts closely to the exact inverse. The rest of the section discusses right approximate inverses; a left approximate inverse can be computed by optimizing $\min_{M\in\mathcal{S}}\|I - MA\|_F^2 = \min_{M\in\mathcal{S}}\|I - A^TM^T\|_F^2$ instead.

4.2.1 When sparsity pattern is known

This approach is even more efficient if we further restrict the constraint set $\mathcal{S}$ to only include matrices with a selected sparsity pattern $\mathcal{G}$. Specifically, we may be only interested in matrices M with $\mathcal{S}(M) \subseteq \mathcal{G} \subseteq \{(i,j) \mid 1 \leq i,j \leq n\}$. Then, it is possible to implement efficient parallel computation. As already mentioned, every column $\vec{m_j}$ may be optimized independently and every parallel worker might receive a copy of the coefficient matrix A before computation starts, eliminating communication during the computation. Moreover, each worker only needs a subset of columns of A. To find the vector minimizing $\|\vec{e_j} - A\vec{m_j}\|_2^2$ with prescribed nonzero pattern, we only need to consider columns $A_{:,i}$ for which $(i,j) \in \mathcal{G}$, because the remaining columns correspond to positions where $\vec{m_j}$ is prescribed to be zero. Denote the set of these indices $\mathcal{J} = \{i \mid (i,j) \in \mathcal{G}\}$ and $A_{:,\mathcal{J}}$ the submatrix of A with column indices in $\mathcal{J}$. Furthermore, only nonzero rows of $A_{:,\mathcal{J}}$ need to be considered, and hence the original least squares problem $\|\vec{e_j} - A\vec{m_j}\|_2^2$ is equivalent to

$$\|\vec{\hat{e}_j} - \hat{A}\vec{\hat{m}_j}\|_2^2,$$

where $\mathcal{I}$ denotes the set of indices of nonzero rows in $A_{:,\mathcal{J}}$, $\hat{A} = A_{\mathcal{I},\mathcal{J}} \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{J}|}$ is the relevant submatrix, $\vec{\hat{m}_j} \in \mathbb{R}^{|\mathcal{J}|}$ includes only positions in $\mathcal{J}$, and $\vec{\hat{e}_j} \in \mathbb{R}^{|\mathcal{I}|}$ includes only positions in $\mathcal{I}$. In other words, the original least squares problem reduces to a small least squares problem that can be solved efficiently using dense matrix techniques, e.g., by a dense QR factorization of $\hat{A}$.

A priori selection of sparsity pattern The difficult part of the above approach is properly selecting the constraint set $\mathcal{S}$, which ought to enforce the sparsity of the approximate inverse by only considering entries that vitally contribute to the preconditioner's quality. Relevant positions are known in some cases, e.g., for problems arising from structured discretizations, but not in general. One possible heuristic is to ignore entries in the inverse that are small in absolute value while retaining the large ones. Unfortunately, the positions of large inverse entries are usually unknown for general sparse matrices. Motivated by the empirical observation that entries of A^{-1} located at positions in $\mathcal{S}(A)$ tend to be relatively large, a common choice is to select $\mathcal{S}$ as the set of matrices with the same sparsity structure as the matrix A. However, this choice need not be robust enough for general sparse matrices: large entries A^{-1} may also reside in positions outside $\mathcal{S}(A)$. There are other, more sophisticated strategies for a priori selection of the sparsity pattern (refer to, e.g., Huckle [1999]), which we do not describe here. These approaches are typically more costly yet with little guarantee that they will result in a good preconditioner in a general setting.

4.2.2 Adaptive strategies

Instead of attempting to find and prescribe a good nonzero pattern for M, adaptive strategies start with a simple initial guess (e.g., a diagonal matrix) and successively augment the pattern until a stopping criterion (e.g., $\|\vec{e_j} - A\vec{m_j}\|_2^2 < \varepsilon$ for a selected $\varepsilon > 0$) is satisfied in each column, or the maximum number of nonzeros is reached. The most successful of these approaches, referred to as **SPAI**, was initially introduced by Cosgrove et al. [1992]. However, it was in the subsequent study by Grote and Huckle [1997] that the name SPAI was officially used. Algorithm 2 provides a detailed description of the SPAI algorithm.

Algorithm 2 SPAI algorithm (Grote and Huckle [1997])

---

input $A \in \mathbb{R}^{n\times n}$
For every column $\vec{m}_j$:
1: Select initial sparsity pattern $\mathcal{J}$
2: $\mathcal{I} \leftarrow$ indices corresponding to nonzero rows of $A_{:,\mathcal{J}}$
3: $\widehat{A} \leftarrow A_{\mathcal{I},\mathcal{J}}$
4: Compute QR decomposition of $\widehat{A}$
5: $\widehat{\vec{m}_j} \leftarrow$ least squares solution of $\widehat{A}\vec{x} = \vec{\hat{e}_j}$
6: $\vec{r} \leftarrow \vec{\hat{e}_j} - \widehat{A}\widehat{\vec{m}_j}$
While $\|\vec{r}\|_2^2 \geq \varepsilon$:
7: $\mathcal{L} \leftarrow \{l \mid \vec{r}_l \neq 0\}$
8: $\tilde{\mathcal{J}} \leftarrow \{j \mid \exists l \in \mathcal{L} : A_{l,j} \neq 0\} \setminus \mathcal{J}$
For $k \in \tilde{\mathcal{J}}$:
9: $\rho_k^2 \leftarrow \|\vec{r}\|_2^2 - (\vec{r}^T A\vec{e_k})^2 / \|A\vec{e_k}\|_2^2$
10: $\tilde{\mathcal{J}} \leftarrow \{k \in \tilde{\mathcal{J}} \mid \rho_k \text{ is large}\}$
11: $\tilde{\mathcal{I}} \leftarrow$ indices corresponding to nonzero rows of $A_{:,\tilde{\mathcal{J}}}$
12: $\mathcal{I} \leftarrow \mathcal{I} \cup \tilde{\mathcal{I}}$, $\mathcal{J} \leftarrow \mathcal{J} \cup \tilde{\mathcal{J}}$
13: $\widehat{A} \leftarrow A_{\mathcal{I},\mathcal{J}}$
14: Compute QR decomposition of $\widehat{A}$
15: $\widehat{\vec{m}_j} \leftarrow$ least squares solution of $\widehat{A}\vec{x} = \vec{\hat{e}_j}$
16: $\vec{r} \leftarrow \vec{\hat{e}_j} - \widehat{A}\widehat{\vec{m}_j}$
return $M = \begin{bmatrix} \vec{m}_1 & \vec{m}_2 & \dots & \vec{m}_n \end{bmatrix}$

Apart from the initial sparsity pattern, the algorithm requires several parameters: 1) tolerance ε on the residuals, 2) either maximum number of new nonzeros, or a tolerance δ to select a subset of $\tilde{\mathcal{J}}$ in Step 10, and 3) the maximum number of iterations of the While cycle.

Unfortunately, the serial cost of constructing the SPAI preconditioner may be high, possibly with higher memory requirements, too. To address these issues, Chow and Saad [1998] proposed to start with an initial guess for the inverse and then use an iterative algorithm to refine it. Their approach is based on minimizing the residuals corresponding to the columns of the approximate inverse. Since the residual vector obtained in each step can be dense, the residual minimization must be followed by sparsification. The simplest way to sparsify is to use tolerance-based dropping[1]. The basic version, called the **Minimal Residual algorithm (MR)**, is shown in Algorithm 3.

At each step, MR algorithm computes the current residual $\vec{r_j}$ and then performs a one-dimensional minimization of the residual norm $\|\vec{e_j} - A\vec{m}_j\|_2^2$ in the direction $\vec{r_j}$. This can be further improved by self-preconditioning, which results in better quality of the preconditioner, but with added computation cost (we refer

[1]The authors also proposed a more effective strategy.

Algorithm 3 MR algorithm (Chow and Saad [1998])

input $A \in \mathbb{R}^{n\times n}$, initial guess M_0
$M \leftarrow M_0$
For every column $\vec{m}_j$ of M:
  For $i = 1, 2, \ldots,$ `max_iter`:
    $\vec{r_j} \leftarrow \vec{e_j} - A\vec{m}_j$
    $\alpha_j \leftarrow \vec{r_j}^T A\vec{r_j} / \|A\vec{r_j}\|_2^2$
    $\vec{m}_j \leftarrow \vec{m}_j + \alpha_j \vec{r_j}$
    $\vec{m}_j \leftarrow$ `sparsify`$(\vec{m}_j)$
return M

to the survey paper by Benzi and Tůma [1999] and the original paper by Chow and Saad [1998] for details).

The user must select the `max_iter` parameter (or other stopping criterion) and specify how to sparsify the approximate solutions $\vec{m}_j$, e.g., by specifying the number of entries that are kept. Most importantly, the user needs to specify the initial guess M_0. Typical choices include $M_0 = 0$, $M_0 = \alpha I$ or $M_0 = \alpha A^T$ for a properly selected alpha.

4.3 Factorized sparse approximate inverse

Factorized sparse approximate inverse approaches are based on the following observation. If the matrix A can be factorized as

$$A = LDU \tag{4.2}$$

where L is unit lower triangular, D is diagonal and U is unit upper triangular, we may express its inverse as

$$A^{-1} = U^{-1}D^{-1}L^{-1} = ZD^{-1}W^T,$$

where $Z = U^{-1}$ and $W = L^{-T}$ are unit upper triangular matrices. In general, the inverse factors Z and W may be dense. The factorization (Eq. (4.2)) generally creates fill-in in both factors L and U (i.e., L is generally denser than the lower triangular part of A and similarly for U). Furthermore, any square unit (lower or upper) triangular matrix X can be written as $X = I + N$, where N is a triagular matrix with zero diagonal. It is easy to verify that $N^n = 0$ (where n is the dimension of N) and using the relation $1 - x^n = (1-x)(1 + x + \ldots x^{n-1})$ with $x = -N$ we obtain

$$I = I - (-N)^n = (I + N)\Big(I + \sum_{k=1}^{n-1}(-1)^k N^k\Big),$$

from which we obtain

$$X^{-1} = (I + N)^{-1} = I + \sum_{k=1}^{n-1}(-1)^k N^k \tag{4.3}$$

for a nonsingular matrix X. Consequently, if A is an irreducible band matrix, Z and W will be entirely filled above the diagonal. To obtain a factorized sparse approximate inverse of A, we need to find nonsingular sparse approximations $\widehat{Z} \approx Z$, $\widehat{D} \approx D$, and $\widehat{W} \approx W$. Then, the factorized approximate inverse is

$$\widehat{Z}\widehat{D}^{-1}\widehat{W}^T \approx A^{-1}. \tag{4.4}$$

Conceptually, there are two approaches to this task. The first, discussed in this section, is to construct the factors $\widehat{Z}$, $\widehat{D}$, and $\widehat{W}$ directly. We mention three methods of this type. Another possibility (discussed in Section 4.4) is to construct an incomplete factorization $A \approx \widehat{L}\widehat{D}\widehat{U}$ and approximately invert the (incomplete) factors.

4.3.1 FSAI method

Kolotilina and Yeremin [1993] proposed the **FSAI** preconditioner, which we describe in the simplified setting when A is symmetric positive definite (SPD). This method requires a priori selection of the sparsity pattern $\mathcal{S}_L$ of the computed approximation of L^{-1} (where L is the Cholesky factor of A). $\mathcal{S}_L$ must include the main diagonal. Common choices for the sparsity pattern are

$$\mathcal{S}_L = \{(i,j) \mid (i,j) \in \mathcal{S}(A^k), i \geq j\}$$

for a small positive integer k, e.g., $k = 1, 2, 3$.

Given $\mathcal{S}_L$, a lower triangular matrix $\widehat{G}_L$ is computed as a solution of the matrix equation

$$(A\widehat{G}_L)_{i,j} = I_{i,j},\ (i,j) \in \mathcal{S}_L.$$

The computation decomposes into solutions of smaller SPD linear systems, one for each column of $\widehat{G}_L$. The individual linear systems may be solved in parallel. For column j of $\widehat{G}_L$, the size of its corresponding linear system is equal to the number of nonzeros allowed in that column. All diagonal entries of $\widehat{G}_L$ are positive. Let $\vec{d} = \mathrm{diag}(\widehat{G}_L)$, $\widehat{D} = \mathrm{diag}(\vec{d})^{-1}$ and $G_L = \widehat{D}^{\frac{1}{2}}\widehat{G}_L$. Then the preconditioned matrix $G_L A G_L^T$ is SPD, and all its diagonal entries are 1. Therefore, we may view $G_L A G_L^T$ as an approximation of the identity matrix ($G_L A G_L^T \approx I$), and hence construct the approximate inverse of A as

$$A^{-1} \approx G_L^T G_L.$$

Furthermore, it can be shown that the approximate factor satisfies

$$G_L = \mathrm{argmin}_{X:\ \mathcal{S}(X) \subseteq \mathcal{S}_L} \|I - XL\|_F^2,$$

connecting FSAI to the Frobenius norm minimization methods of Section 4.2.

FSAI can be extended to work for nonsymmetric matrices. However, in such cases, one must pay attention to the individual small linear systems which are no longer SPD. Several efficiency improvements to the algorithm were proposed in recent years, too, for instance, block and later supernodal variants with parallel implementation (see, e.g., Janna et al. [2010, 2013], Ferronato et al. [2014], Janna et al. [2015a], Ferronato and Pini [2018]). The advancements mentioned provide foundations for the state-of-the-art high-performance preconditioning software FSAIPACK (Janna et al. [2015b]).

4.3.2 Incomplete biconjugation

The **AINV** method (first proposed by Benzi [1993]; for detailed description, refer also to Benzi et al. [1996] and Benzi and Tůma [1998]) computes the factorized approximate inverse of A using incomplete (bi)conjugation. The appeal of this approach is in its robustness: AINV works for general nonsingular matrices and does not require a priori sparsity pattern selection.

Two sets of n-dimensional real vectors $\{\vec{z_i}\}_{i=1}^n, \{\vec{w_i}\}_{i=1}^n$ are called A-*biconjugate* if they satisfy

$$\vec{w_i}^T A \vec{z_j} = 0 \iff i \neq j.$$

Given a pair of matrices Z, W with A-biconjugate columns, $Z = \begin{bmatrix} \vec{z_1} & \vec{z_2} & \dots & \vec{z_n} \end{bmatrix}$ and $W = \begin{bmatrix} \vec{w_1} & \vec{w_2} & \dots & \vec{w_n} \end{bmatrix}$, it holds that

$$W^T A Z = D = \begin{bmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d_n \end{bmatrix}.$$

We can use the above relation to express the inverse of A as

$$A^{-1} = Z D^{-1} W^T, \tag{4.5}$$

or using the outer product of columns of matrices Z and W as the sum of rank one matrices

$$A^{-1} = \sum_{i=1}^{n} \frac{\vec{z_i} \vec{w_i}^T}{d_i}. \tag{4.6}$$

A pair of matrices with A-biconjugate columns Z and W can be computed using a biconjugation process (see Algorithm 4) applied to the columns of any two nonsingular matrices $Z^{(0)}, W^{(0)} \in \mathbb{R}^{n \times n}$. There are infinitely many pairs of A-

---

Algorithm 4 Biconjugation process

input $A \in \mathbb{R}^{n \times n}$, initial sets of vectors $\{\vec{z_i}^{(0)}\}_{i=1}^n, \{\vec{w_i}^{(0)}\}_{i=1}^n$
For $i = 1, 2, \dots, n$:
  For $j = i, i+1, \dots, n$:
1: $d_j^{(i-1)} \leftarrow A_{i,:} \vec{z_j}^{(i-1)}$
2: $q_j^{(i-1)} \leftarrow A_{:,i}^T \vec{w_j}^{(i-1)}$
  If $i = n$:
3: Go to Step (6)
  For $j = i, i+1, \dots, n$:
4: $\vec{z_j}^{(i)} \leftarrow \vec{z_j}^{(i-1)} - \frac{d_j^{(i-1)}}{d_i^{(i-1)}} \vec{z_i}^{(i-1)}$
5: $\vec{w_j}^{(i)} \leftarrow \vec{w_j}^{(i-1)} - \frac{q_j^{(i-1)}}{q_i^{(i-1)}} \vec{w_i}^{(i-1)}$
6: $Z \leftarrow \begin{bmatrix} \vec{z_1}^{(i-1)} & \vec{z_2}^{(i-1)} & \dots & \vec{z_n}^{(i-1)} \end{bmatrix}$
7: $W \leftarrow \begin{bmatrix} \vec{w_1}^{(i-1)} & \vec{w_2}^{(i-1)} & \dots & \vec{w_n}^{(i-1)} \end{bmatrix}$
8: $D \leftarrow \mathrm{diag}(d_1, d_2, \dots, d_n)$
return Z, D, W

---

biconjugate vector sets, each stemming from a different choice of initial sets. It is convenient to apply the biconjugation process to canonical basis vectors, i.e., to choose $Z^{(0)} = W^{(0)} = I$. The biconjugation process can be viewed as a two-sided generalized Gram-Schmidt orthogonalization with respect to the bilinear form associated with A. If A is SPD, then $Z = W$, and we only perform the process for Z. The algorithm then performs a conjugate Gram-Schmidt process (for the "energy" inner product $\langle \vec{x}, \vec{y} \rangle = \vec{x}^T A \vec{y}$).

For the initial guess $Z^{(0)} = W^{(0)} = I$, Algorithm 4 does not break down (due to zero division) in exact arithmetic if and only if all the leading principal minors of A are nonzero, which is precisely when the LU decomposition of A can be computed without pivoting. If this is satisfied, one can verify that the factors computed by the biconjugation algorithm satisfy $Z = U^{-1}$ and $W = L^{-T}$ for the factors L, U from the LDU factorization of A, and D is the same matrix in both the LDU factorization and in the factorization resulting from Algorithm 4.

Moreover, when $Z^{(0)} = W^{(0)} = I$, vectors $\vec{z_j}^{(i)}$ and $\vec{w_j}^{(i)}$ are initially very sparse thanks to the choice of the initialization, but they start filling in rapidly due to steps (4) and (5). Sparsity in the inverse factor is preserved by carrying out the biconjugation process incompletely, similar to how incomplete LU factorizations work. In particular, we apply numerical dropping, either based on positions (i.e., by restricting the sparsity patterns of the computed factors) or based on drop tolerance, where entries are zeroed out if they are small in magnitude[2]. The second approach is much more robust and effective, especially for problems with unstructured sparsity patterns. Because of incompleteness, Algorithm 4 may break down unless A satisfies additional, stronger conditions (see Benzi and Tůma [1999] and Benzi et al. [1996] for details).

4.3.3 Bordering approach

The last approach we present in this section is based on bordering. We briefly describe the **Approximate Inverse by Bordering (AIB)** method, a modification of the approach proposed by Saad [1996].

The bordering scheme

$$\begin{bmatrix} W^T_{1:k,1:k} & \vec{0} \\ W^T_{1:k,k+1} & 1 \end{bmatrix} \begin{bmatrix} A_{1:k,1:k} & A_{1:k,k+1} \\ A_{k+1,1:k} & A_{k+1,k+1} \end{bmatrix} \begin{bmatrix} Z_{1:k,1:k} & Z_{1:k,k+1} \\ \vec{0}^T & 1 \end{bmatrix} = \begin{bmatrix} D_{1:k,1:k} & \vec{0} \\ \vec{0}^T & D_{k+1,k+1} \end{bmatrix}$$

uncovers an n-step process for computing the factors Z, D and W, assuming A has an LU factorization. Start by setting $Z_{1,1} = W_{1,1} = 1$ and $D_{1,1} = A_{1,1}$. Assuming we have already computed the leading submatrices $Z_{1:k,1:k}, W_{1:k,1:k}$ and $D_{1:k,1:k}$, we set $D_{1:k,k+1} = \vec{0}$, $D_{k+1,1:k} = \vec{0}^T$ and use above expression to obtain

$$\begin{aligned} Z_{1:k,k+1} &= -Z_{1:k,1:k} D^{-1}_{1:k,1:k} W^T_{1:k,1:k} A_{1:k,k+1} \\ W_{1:k,k+1} &= -W_{1:k,1:k} D^{-1}_{1:k,1:k} Z^T_{1:k,1:k} A^T_{k+1,1:k} \end{aligned}$$

and finally

$$D_{k+1,k+1} = A_{k+1,k+1} + W^T_{1:k,k+1} A_{1:k,1:k} Z_{1:k,k+1} + A_{k+1,1:k} Z_{1:k,k+1} + W^T_{1:k,k+1} A_{1:k,k+1}.$$

[2]Tolerance-based dropping is popular in incomplete LU (or Cholesky) factorization methods.

Here, the computations of the inverse factors are interconnected (compare that with the biconjugation Algorithm 4, where the two factors are computed independently of one another). For symmetric A, once again, $Z = W$, and only half the computation is needed. Furthermore, if A is SPD, the diagonal entries of D are all positive (if the computation is carried out in exact arithmetic), and the AIB preconditioner is well-defined. In general, diagonal modifications may be necessary to prevent breakdowns.

A sparse approximate factorization of A^{-1} is obtained when the bordering process is carried out incompletely, e.g., by dropping small elements in the computed vectors $Z_{1:k,k+1}$ and $W_{1:k,k+1}$. In addition to performing a matrix-sparse vector product with A_k, each step of the bordering algorithm performs four sparse matrix-vector products (with $Z_{1:k,1:k}, W_{1:k,1:k}$, and their transposes). These operations account for most of the work and must be computed efficiently by exploiting the sparsity of both the matrix and the vector.

4.4 Inverse incomplete factorization techniques

Compared with approaches presented in Section 4.3, methods in this section find an approximate inverse in the form from Equation (4.4) in two stages:

1. Compute an incomplete LU (or LDU) decomposition of A using standard techniques for incomplete factorization to obtain factors $\widehat{L}$, $(\widehat{D},)$ and $\widehat{U}$.

2. Find sparse approximate inverses of $\widehat{L}$ and $\widehat{U}$.

Note that different methods for incomplete factorization and approximate inversion of the factors result in different preconditioners.

When the incomplete factors $\widehat{L}, \widehat{U}$ are available, it is possible to compute their approximate inverses by inexactly solving $2n$ triangular systems

$$\widehat{L}\vec{x_j} = \vec{e_j},\ \widehat{U}\vec{y_j} = \vec{e_j} \text{ for } 1 \leq j \leq n, \tag{4.7}$$

where n is the dimension of $\widehat{L}$. These linear systems are independent, so it is possible to implement their solution in parallel. The individual systems are solved only approximately to preserve the required sparsity in the resulting factors and to save computation costs. Here, there are, once again, several possible ways:

1. We can prescribe sparsity patterns $\mathcal{S}_L$, $\mathcal{S}_U$ for the approximate inverses of $\widehat{L}$ and $\widehat{U}$, and proceed by constrained minimization of the Frobenius norms

   $$\|I - \widehat{L}X\|_F^2,\ \|I - \widehat{U}Y\|_F^2,$$

   as discussed in Section 4.2.1.

2. Alternatively, we can use adaptive strategies from Section 4.2.2 or approaches proposed by Calgaro et al. [2010], which compute sparse approximate inverse factors via incremental Frobenius norm minimization.

3. According to van Duin and Wijshoff [1996], the most accurate method to compute the solutions of the linear systems (4.7) is forward substitution for the lower triangular systems $\widehat{L}\vec{x_j} = \vec{e_j}$ and back substitution for the

upper triangular systems $\widehat{U}\vec{y_j} = \vec{e_j}$. Sparsity is enforced by dropping entries in the solution vectors based on position or tolerance. For each system, performing the sparsification during the substitution process rather than after completion is more practical.

4.5 Comparison of approaches

4.5.1 Frobenius norm minimization methods

With a fixed prescribed sparsity pattern (Section 4.2.1), Frobenius norm minimization is one of the fastest ways to compute sparse approximate inversion. This is due to the method's decomposition into many small, embarrassingly parallel subproblems. In more complex problems, however, the positions of dominant inverse entries are unknown, and heuristical pattern selection often results in inverses of poor quality; the resulting preconditioners perform below par with more sophisticated methods in these cases.

Adaptive methods (Section 4.2.2) sacrifice some of the speed by iteratively correcting the sparsity pattern, which generally improves the quality of the preconditioner. Positively, SPAI and MR are still well-parallelizable (at least in theory[3]), with the latter being typically much less computationally demanding. While MR iterations use only simple and efficient sparse operations (sparse vector addition, sparse dot products, and sparse matrix-vector products)[4], SPAI iterations compute a QR decomposition of a matrix and solve a least-squares system. Of course, the serial cost of MR may still be relatively high, especially when using self-preconditioning.

A serious limitation in terms of applicability is that preconditioners based on Frobenius norm minimization cannot be used with the conjugate gradient method to solve SPD problems since, in general, the computed preconditioner matrix M will not be positive definite, even if A is. Although unlikely in practice, even nonsingularity of M is not guaranteed. Moreover, preconditioners computed by adaptive strategies need not even be symmetric.

While incomplete factorization preconditioners are sensitive to reorderings, the SPAI and MR preconditioners are not. The advantage is that A can be reordered and partitioned arbitrarily without affecting the resulting quality, which is convenient, e.g., for load balancing. However, this also means that we cannot use reordering to reduce fill-in or improve the quality of the result. If, for example, no small entries exist in A^{-1}, adaptive methods cannot find the dominant sparsity pattern regardless of the used reordering, since the inverse of a permutation of A is just a permutation of A^{-1}. This differs for the preconditioners described in Sections 4.3 and 4.4.

A vital advantage of the methods based on Frobenius norm minimization is their flexibility. The adaptive methods like SPAI or MR can be restarted, with the computed solution used as the new initial guess. They are often memory efficient, which is especially true for MR, for which the only storage needed is that for the factor M. Moreover, MR can work in situations A is not explicitly

[3]Because of the adaptive and irregular nature of the computations, exploiting the inherent parallelism is not straightforward in practice.

[4]And both methods need to select the positions of the entries to-be-kept after sparsification.

available (e.g., A is only given as a function) since only matrix-vector products with A are needed. Lastly, we can apply these approaches to improve a given preconditioner.

SPAI and MR preconditioners were demonstrated to solve very hard problems for which more standard techniques, like incomplete LU, fail (see, e.g., Saad [1995]). In this sense, they offer a valuable addition to other well-established methods. Finally, the substantial parallelism potential of these methods makes them applicable for solving very large problems, including in distributed memory environments.

4.5.2 Factorized sparse approximate inverse

Factorized sparse approximate inverses (discussed in Section 4.3) implicitly impose structural requirements on the computed approximate inverse. As a result, they avoid certain problems that limit the applicability of SPAI or MR preconditioners. Primarily, the methods listed in Section 4.3 can be used as preconditioning for the conjugate gradient method. It is clear that if A is SPD, then $Z = W$ and the approximate inverse matrix $M = ZD^{-1}Z^T$ is SPD as long as all diagonal entries of D are positive. Moreover, if M was successfully constructed, the diagonal entries of D must all be nonzero, and since Z has full rank because it is unit upper triangular, M is nonsingular.

Furthermore, factorized approaches exhibit two characteristics that favorably impact the compression-accuracy trade-off. First, they are sensitive to reorderings of the coefficient matrix A. This attribute can be leveraged to reduce fill-in within the inverse factors and enhance the effectiveness of the resulting preconditioner, as discussed in, e.g., Benzi and Tůma [1998]. Secondly, factorized approaches yield factors that collectively represent a denser approximate inverse matrix compared to non-factorized methods while utilizing the same amount of storage. Hence, assuming they can provide better approximations of A^{-1} is reasonable. This observation was argued by Chow [1997] and experimentally validated by Benzi and Tůma [1999]. In addition, factorized forms can be less expensive to compute (at least for symmetric A) and typically require fewer user-selected parameters, making them easier to use.

Nevertheless, factorized approaches face challenges of their own. As incomplete (inverse) factorization methods, they are prone to breakdowns, similar to incomplete LU. Although strategies exist to mitigate such issues (for instance, using diagonal shifts), the required shifts (perhaps large or numerous) may negatively affect the resulting preconditioner. Additionally, when applying factorized preconditioners, we must perform two consecutive matrix-vector multiplications with the factors, adding sequentiality.

Regarding the specific methods, the FSAI preconditioner can be efficiently implemented in parallel and is suitable for high-performance computing (Janna et al. [2015b]). It has been successfully employed to solve challenging, structured problems, e.g., in Kolotilina and Yeremin [1995]. The method's main drawback lies in the requirement of predefining the sparsity pattern of the approximate inverse factors, which makes it difficult to use for problems with general sparsity patterns. Moreover, while the method can be extended to handle nonsymmetric matrices A, solving local linear systems in such cases becomes more challenging.

On the other hand, the AINV method is more robust, applicable to general nonsingular matrices, and does not necessitate a priori selection of sparsity patterns. However, the biconjugation process in AINV is highly sequential, limiting possible parallelization. Lastly, implementing AINV on distributed memory machines is very difficult in the presented formulation.

4.5.3 Inverse incomplete factorization techniques

Approaches based on inverting incomplete factorizations (discussed in Section 4.4 share some of the advantages of the factorized approximate inverse methods of Section 4.3, namely the implications on the structure of the resulting approximate inverse (e.g., symmetric positive definiteness). However, they are subject to other limitations that the preconditioners of other classes do not have.

One disadvantage follows from the requirement that an incomplete factorization does not break down; we cannot use the methods of this category when an incomplete factorization does not exist or when it is unstable[5]. The parallel efficiency of these techniques is also reduced when A is not SPD, since in that case, the construction phase of the preconditioner includes a highly sequential incomplete LU factorization[6].

The second disadvantage is the involvement of two levels of approximation or incompleteness, instead of just one like in methods described in Sections 4.2 and 4.3. This can lead to significant quality degradation of the resulting approximate inverse or the preconditioner.

Lastly, using the inverse incomplete factorization methods in practice is not straightforward due to the large number of parameters typically needed to be selected by the user.

[5]These situations can occur for highly nonsymmetric, indefinite problems, see, e.g., Chow and Saad [1997] and Elman [1986].

[6]The sequentiality is less of an issue when A is SPD; sparse (incomplete) Cholesky decomposition provides more room for parallelization.

5. Enhancing scalability of Embarrassingly Shallow Autoencoder

As discussed in Section 3.4, the applicability of the recommender system EASE^{R} in domains with large item sets may be prohibited by the model's high memory requirements. The main objective of this thesis is to alleviate this issue. Our proposed solution is a method for computing a sparse approximation of the full-rank component $\widehat{B_{diag}}$ of the weight matrix $\widehat{B}$ of EASE^{R}, which we describe in this Chapter. We also published a more concise explanation of the approach in our recently accepted paper (Spišák et al. [2023]).

5.1 Method selection

Since computing, storing, and using the inverse of a large matrix may be too expensive for large CF applications, we aim to find its sparse approximation. This approach agrees with Steck [2019b], who builds their MRF model as a sparse approximation of EASE^{R}. The MRF model is based on methods developed for sparse covariance approximation. We opted for a different approach, which employs numerical methods for sparse approximate inversion reviewed in Chapter 4.

Methods for sparse approximate inversion presented in Chapter 4 differ in their intended use cases. Some methods work for general sparse matrices, while others are better-suited matrices with specific properties. Some approaches assume the knowledge of, for example, the dominant sparsity pattern, which can be known least partially in particular applications (e.g., sparse systems arising from discretizations of structured problems). Analyzing the specific properties of our problem will allow us to select an appropriate method tailored to our requirements with minimal unnecessary overhead.

5.1.1 Properties of the problem

Optimization objective Our main goal is to find an accurate sparse approximation of $(X^T X + \lambda I)^{-1}$, where $X \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{I}|}$ is a sparse input user-item interaction matrix and $\lambda > 0$ is the L2 regularization hyperparameter. Let us elaborate on the meaning of "accuracy" in this task.

The weight matrix $\widehat{B}$ of EASE^{R} is obtained by rescaling the columns of the matrix $(X^T X + \lambda I)^{-1}$ and adding the identity matrix, and acts as an autoencoder when applied from the right on a row vector of user feedback (see Section 3.1). Therefore, we are *not* looking for an approximate inverse M which minimizes $\|M - (X^T X + \lambda I)^{-1}\|$ (in some norm). Instead, we aim to find M which *operates* (when applied from the right) as closely to A^{-1} as possible. Formally, we want to minimize

$$\|I - (X^T X + \lambda I)M\|_F.$$

The choice of Frobenius norm agrees with the optimization problem (3.1).

Large size We assume that the item set size $|\mathcal{I}|$ is very large, with potentially millions of items. Moreover, we also assume that $|\mathcal{U}|$ is large (possibly $|\mathcal{U}| > |\mathcal{I}|$). Hence, the selected method must be fast (for instance, by allowing a good amount of parallelization). At the same time, we are looking for a method that does not require significant memory to perform its computation or one where we can control memory usage.

Furthermore, due to the potentially extensive problem size, obtaining a highly accurate approximation may be computationally expensive. In such cases, even less accurate approximation may suffice. For this reason, we prefer iterative approaches, which may yield useful approximation after a few iterations but can be further refined with additional iterations if we so choose.

Unknown sparsity pattern Unlike in certain structured problems, the collaborative filtering task does not pose restrictions on the sparsity structure of the item-item matrix $\widehat{P} = (X^T X + \lambda I)^{-1}$. More precisely, in EASE$^{\text{R}}$, the only information available is that $\widehat{P}$ is symmetric and has a nonzero diagonal. Neither the sparsity pattern of $\widehat{P}$ nor the positions of its dominant entries are known a priori. Therefore, we need the chosen method to automatically identify the dominant sparsity pattern.

Symmetric positive definiteness By Propositions 3.1 and 3.3, both matrices $X^T X + \lambda I$ and $(X^T X + \lambda I)^{-1}$ are symmetric positive definite. Therefore, it makes sense to primarily consider methods tailored to matrices with this property and perhaps focus on methods that incorporate factorization.

Symmetry helps avoid some sequentiality (e.g., back substitutions) in algorithms and positively affects computation time by reducing the number of floating-point operations needed.

Factorization can provide an additional layer of compression: compared with a sparse square matrix A with a set number of nonzeros $\texttt{nnz}(A)$, a product of two matrices B, C with the same dimensions as A and $\texttt{nnz}(B) + \texttt{nnz}(C) = \texttt{nnz}(A)$ often represents a denser operator ($\texttt{nnz}(BC) > \texttt{nnz}(A)$). In other words, a factorized approximation may be more accurate for the same amount of storage.

Full rank Related to the previous point, we require the selected method to preserve an important property of EASE$^{\text{R}}$: full rank of $\widehat{B_{diag}}$ (see Section 3.1.2). For this reason, the constructed approximate inverse needs to be nonsingular.

Data Gram matrix is denser The Gram matrix $X^T X$ is often considerably denser than X, even when X is sparse (see the densities for datasets used in our experiments in Table 6.1). Crucially, $X^T X$ must fit in memory for the duration of training; otherwise, data transfers would substantially hinder the computation of the approximate inverse. Even though we do not expect the size of $X^T X$ to cause issues in practice[1], the ability to compute the inverse without explicitly calculating $X^T X$ would be an added benefit.

[1]Since it is not very likely that $X^T X$ is very large and simultaneously contains a huge number of nonzero entries.

Ability to prescribe model size Lastly, we require the ability to a priori select the density of the computed approximate inverse arbitrarily, based on available resources.

5.1.2 Selected approach

From the outset, we do not consider methods working with fixed sparsity patterns, as our application does not provide knowledge about the positions of dominant inverse entries. Instead, we focus on methods that find the dominant entries automatically. Moreover, instead of prescribing the total number of entries in absolute terms, we construct the method to work with a user-specified parameter `target_density`, which determines the density of the resulting approximate matrix (or factors). Ideally, with a higher allowed density, we aim to obtain better approximations of the inverse matrix. However, sometimes we must use a very sparse approximation due to memory limitations (e.g., during inference).

We have decided to use an approach based on factorization (see Sections 4.3 and 4.4) for the following reasons:

1. Since the approximated inverse matrix $\widehat{P} = (X^T X + \lambda I)^{-1}$ is symmetric positive definite, it is sensible to prioritize an approximation that preserves this property. Factorization-based methods are capable of this.

2. We require the approximation to have full rank, i.e., to be nonsingular, which factorization-based methods can guarantee.

3. Having the inverse available only in factorized form does not severely limit inference speed. Performing two sparse matrix-vector products during inference instead of one is acceptable.

4. Rather, the approximate inverse stored in a factorized way improves model compression, as the sparse factors together represent a much denser matrix. The approach may provide a more accurate approximation of weights at equal storage costs.

5. Finally, symmetric factorization should improve the total training complexity by reducing the number of floating-point operations.

The selected approach must facilitate the training of models for large-scale problems involving potentially millions of items. Because of this, we are looking for a fast method with a substantial degree of parallelism, which was the decisive factor why we opted not to use the AINV method or the bordering approach. In our preliminary experiments, the sequential nature of the AINV method rendered it too slow. Hence, we base our method on the concept from Section 4.4: **incomplete factorization with subsequent approximate inversion of the factors**.

Because the matrix $A = X^T X + \lambda I$ is SPD, we use an incomplete Cholesky factorization method for the factorization part. Incomplete Cholesky factorization methods offer superior performance compared to incomplete LU factorization, which is more limited in terms of parallelization due to the possible need for pivoting. Furthermore, reorderings (which correspond to item permutations) can be exploited to significantly reduce fill-in during the computation of the factors, thereby minimizing information lost in the incomplete factor. As a bonus, (incomplete) Cholesky factorization of a regularized Gram matrix $X^T X + \lambda I$ can be computed without prior construction of $X^T X + \lambda I$ (as first discussed by Björck [1996] in the context of solving least squares). We discuss possible approaches for approximate Cholesky factorization in Section 5.3.1.

To compute the approximate inverse factor, we came up with a modification of the MR algorithm. The MR algorithm iteratively adjusts the approximation, exhibits the least sequential nature among the available methods, and offers good potential for single instruction multiple data (SIMD) parallelism. The MR iterations also enable trading accuracy for training time. Our modification of MR (discussed in detail in Section 5.3.3) enhances the method's ability to identify general sparsity patterns, allows for controllable memory utilization, and results in a more uniform quality of approximation of individual columns (hence the name Uniform Minimal Residual algorithm (UMR)). Additionally, we identified an effective initial guess heuristic for our situation, allowing the method to converge to an accurate approximate inverse factor quickly. We further discuss the choice of initial guess in Section 5.3.2.

5.2 Model definition

5.2.1 Optimization objective

The model we propose solves the same optimization objective as EASE$^{\text{R}}$, namely

$$\min_B \|X - XB\|_F^2 + \lambda \|B\|_F^2 \text{ s.t. } \operatorname{diag}(B) = \vec{0}.$$

However, instead of using the closed-form solution derived in Section 3.1.1, we use its sparse approximation in a factorized form (as explained in Section 5.1). To expand upon, we use the decomposition of the weight matrix $\widehat{B}$ of EASE$^{\text{R}}$ into a full-rank component $\widehat{B_{diag}}$ and a residual connection, i.e., $\widehat{B} = \widehat{B_{diag}} + I$; see Section 3.1.2. Since $\widehat{B_{diag}}$ is a rescaled version of $(X^T X + \lambda I)^{-1}$, we will compute a *factorized* sparse approximation of $\widehat{B_{diag}}$ by applying the same scaling to the factorized approximate inverse of $X^T X + \lambda I$.

Approximation method

The factorized sparse approximate inverse is found via incomplete Cholesky factorization of $X^T X + \lambda I$ with subsequent approximate inversion of the factors (the framework from Section 4.4). Specifically, the problem of finding a sparse approximation of $(X^T X + \lambda I)^{-1}$ decomposes into two simpler problems:

1. Sparse approximate *square-root-free* Cholesky decomposition of $X^T X + \lambda I$:

$$X^T X + \lambda I \approx \widehat{L}\widehat{D}\widehat{L}^T.$$

2. Finding a sparse approximate inverse K of the matrix $\widehat{L}$.

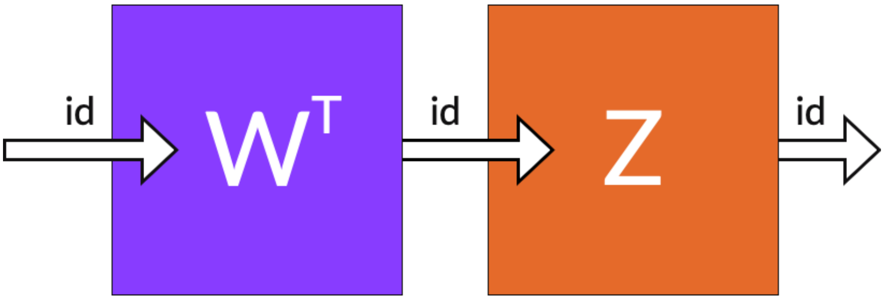


Figure 5.1: SANSA is a sparse nonsymmetric encoder-decoder model. Additionally, an input-output residual connection is added to prevent the self-similarity of input items. However, in situations where we disallow recommending input items, masking the prediction vector is sufficient. (Spišák et al. [2023])

In the first step we compute a sparse approximate decomposition $\widehat{L}\widehat{D}\widehat{L}^T$ with small $\|L - \widehat{L}\|_F$ and $\|D - \widehat{D}\|_F$. Then, we find a sparse $K \approx \widehat{L}^{-1}$ such that $\|I - \widehat{L}K\|_F$ is small. This way,

$$\|I - D\widehat{D}^{-1}\|_F \text{ is small and } \|I - LK\|_F \text{ should be small.}$$

Note that there are two levels of approximation involved for L: $\widehat{L} \approx L$ and $K \approx \widehat{L}^{-1}$. The diagonal factor D, on the other hand, is inverted with higher precision. Extracting the diagonal in step one (by computing an (incomplete) *square-root-free* Cholesky decomposition) should, therefore, benefit approximation quality. The final product $K^T\widehat{D}^{-1}K$ should create a good sparse approximation of $(X^TX + \lambda I)^{-1}$, since

$$\|I - (X^TX + \lambda I)(K^T\widehat{D}^{-1}K)\|_F = \|I - LDL^TK^T\widehat{D}^{-1}K\|_F$$

should be relatively small. While this approach may not be optimal, it is clear that the computed approximations converge to $(X^TX + \lambda I)^{-1}$ as we increase the allowed density (and perform a sufficient number of iterations for finding K). To summarize, the two-step process results in an approximate inverse in the form

$$(X^TX + \lambda I)^{-1} \approx M = K^T\widehat{D}^{-1}K,$$

where M *operates* (when applied from the right) similarly to $(X^TX + \lambda I)^{-1}$, i.e., M approximately minimizes $\|I - (X^TX + \lambda I)M\|_F$. This is the main objective set in Section 5.1.

5.2.2 Architecture

We use the resulting factorized approximation $K^T\widehat{D}^{-1}K$ to build the encoder layer W^T and the decoder layer Z of our model titled **Scalable Approximate NonSymmetric Autoencoder** (SANSA). Formally, SANSA approximates the encoder-decoder matrix $\widehat{B}$ of EASE[R] using a two-layer linear model (with identity activation functions) and added residual connection between input and output: $\widehat{B} \approx W^TZ + I$. The proposed architecture is illustrated in Figure 5.1.

Inference

The inference is analogous to the original model EASE[R]. For a vector of user's feedback $\vec{u}$, the ratings are obtained by two sparse matrix-vector multiplications and adding the input: $\vec{r}^T = (\vec{u}^T W^T)Z + \vec{u}^T$. Note that the illustration in Figure 5.1 does not show the residual connection, which forces the diagonal of $\widehat{B}$ to zero to prevent the self-similarity of input items. Predicting input items is often forbidden in many practical applications. In these cases, the residual connection can be omitted and replaced by simply masking the predicted rating vector $\vec{r}$ on the positions i where $\vec{u}_i \neq 0$.

Storage compression

Motivated by memory-critical problems, we design the architecture so that the maximum density of weights W^T and Z can be *selected* by a parameter. The compound weights $W^T Z + I$ converges to the weights $\widehat{B} = \widehat{B_{diag}} + I$ of EASE[R] as the allowed density increases. However, implicitly representing $\widehat{B_{diag}}$ as $W^T Z$ yields a much denser operator than an unfactorized approximation (like MRF) at equal storage cost would. Apart from the possibility of obtaining better approximation at equal compression (of the dense, uncompressed $\widehat{B_{diag}}$), factorization provides one more advantage for recommender system applications. Compared to sparse single-layer models such as MRF, a sparse linear model with two layers and the same number of parameters will generate denser prediction vectors from sparse input interactions. In other words, a two-layer model can recommend more items based on sparse input, which could be helpful in practice.

5.3 Model training

Even though SANSA uses different encoder and decoder layers, only encoder training is compute-intensive. The decoder is simply a rescaled copy of the encoder (as discussed at the end of this section). We use the methodology selected in Section 5.1 to train the encoder layer. We discuss details about the selected methods for sparse approximate Cholesky decomposition in Section 5.3.1 and propose a convenient choice of the initial guess for the inverse factor in Section 5.3.2. Section 5.3.3 describes our modification of the MR algorithm. The final training procedure of the model is then described in detail in Section 5.3.4. Section 5.4 delves into implementation details.

5.3.1 Sparse (and approximate) Cholesky factorization

The efficiency of sparse factorization methods stems from their ability to ignore everything but the nonzero data. However, the elimination process inside the sparse Cholesky factorization generates new nonzero entries - *fill-in* - in the resulting factor L. [2] Large amounts of generated fill-in can cause practical problems: the complexity of the most critical steps in the factorization is highly dependent on the amount of fill-in, and the computed factor L might require significantly

[2]This is a consequence of **Parter's rule**; see Chapter 3 of the book by Scott and Tůma [2023].

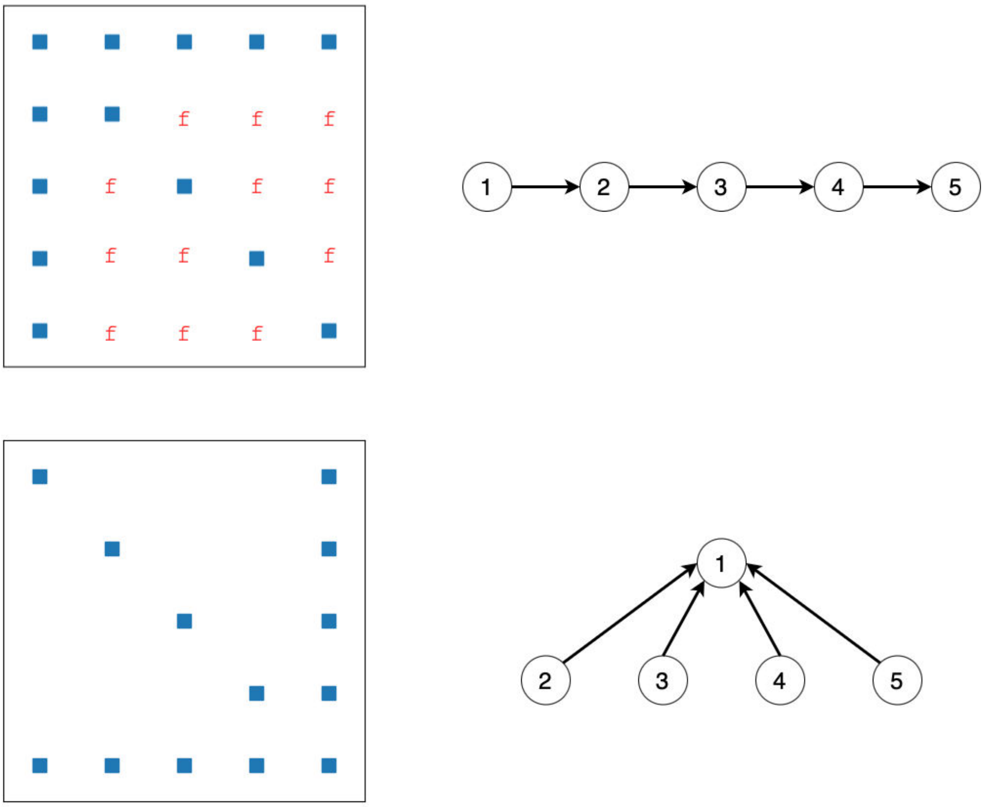


Figure 5.2: Change of elimination tree with symmetric permutation. The factorization of the first matrix is sequential, and the resulting Cholesky factor is completely filled below the main diagonal (f denotes the positions of new fill-in entries in $\mathcal{S}(L + L^T)$). The factorization of the symmetrically permuted matrix is well parallelizable and produces a sparse Cholesky factor.

more memory than the original matrix A. Fortunately, we can determine the number of created fill-in entries and their precise positions by examining the sparsity pattern $\mathcal{S}(A)$ before the numerical factorization begins. This is done during the factorization's initial stage, known as the *symbolic* phase, when the sparsity pattern $\mathcal{S}(L)$ is found by constructing and examining the *elimination tree* of the matrix A (see Appendix A.1 and references therein). Apart from information on fill-in entries, the elimination tree also reveals constraints on the elimination order of columns and hence the parallelization potential of the factorization.

Improving efficiency using reorderings

In general, the elimination trees of a matrix A and its symmetric permutation PAP^T are not only different but need not even be isomorphic (as demonstrated in Figure 5.2). This property can be exploited to

- reduce the amount of fill-in in the factorization, and
- increase the factorization's parallelism.

Therefore, modern sparse Cholesky factorization starts by finding a permutation that reduces fill-in and enhances parallelizability. Depending on the used algorithm, the permutation balances the amount of created fill-in and parallelizability by finding a better initial elimination graph[3]. Besides lowering memory requirements for storing the computed factor, minimizing fill-in *reduces information loss when the factorization is computed incompletely.*

Since the problem of finding a permutation to minimize fill-in is NP-complete, fill-in reduction is based on heuristics. The algorithms frequently use the sparsity pattern $\mathcal{S}(A)$ alone; numerical properties must also be considered for non-definite matrices to make the matrix factorizable. For brevity, we only note that our implementation uses a variant of the widely used approximate minimum degree (AMD) algorithm, which is often relatively less expensive than other methods. More details on fill-in-reducing reorderings can be found in Chapter 8 in Scott and Tůma [2023].

When reordering is not enough

The computed factor L may be too large even after applying fill-in reducing permutation. For instance, when the input matrix A is dense, no reordering can prevent L from being dense. This is common in CF tasks; see Table 6.1. In such cases, memory limitations force us to compute the sparse Cholesky factorization only approximately – incompletely. Approximation may be applied at three points: before, during, and after the factorization.

1. Sparsification before factorization is suboptimal since it loses information before the elimination process even begins. This may lead to significant quality deterioration, as clearly shown, e.g., in a structural mechanics problem (Benzi et al. [2001]). On the other hand, it may help decrease the time and memory requirements if a large matrix strongly fills.

2. A better way is to create approximate factors $\widehat{L}, \widehat{D}$ by sparsifying the complete factor L by dropping entries smallest in magnitude. This approximation approach is optimal in the following sense. Let $A \in \mathbb{R}^{m \times n}$ be a sparse matrix with $\mathtt{nnz}(A)$ nonzero entries, $k \leq \mathtt{nnz}(A)$ and let $A_{(k)}$ be the matrix obtained by keeping only the k entries of A largest in absolute value and zeroing out the rest. Then

   $$A_{(k)} = \operatorname{argmin}_{\substack{B \in \mathbb{R}^{m \times n} \\ \mathtt{nnz}(B) = k}} \|A - B\|_F$$

   Moreover, preserving more entries results in a more accurate approximation. Both statements follow immediately from the definition of the Frobenius norm, which measures the element-wise distance between matrices.

   Unfortunately, the amount of generated fill-in (that needs to be stored) may limit the applicability of this variant for denser inputs.

3. Finally, it is possible to sparsify *during* factorization through incomplete Cholesky factorization (ICF). We use our implementation of the sophisticated column-oriented algorithm proposed by Lin and Moré [1999]. Conceptually, this algorithm builds on the Crout version of Cholesky factorization,

[3] Mind the difference: the elimination graph and the elimination tree are different structures. See Scott and Tůma [2023], Section 3.2.

used in the Yale Sparse Matrix Package (Eisenstat et al. [1981]) and later for efficient incomplete Cholesky factorization by Jones and Plassmann [1995].

Compared with the original algorithm, our implementation uses a heuristic approach for selecting the nonzero entry count for the computed column based on the density of the already computed part. The maximum number of allowed nonzeros in the j-th column (`max_j_nnz`) is computed as

$$\texttt{max\_j\_nnz} = \frac{\texttt{max\_nnz} - \texttt{nnz\_so\_far}}{n - (j - 1)},$$

where `max_nnz` is the maximum number of nonzeros in the computed factor (prescribed as a parameter) and `nnz_so_far` is the number of nonzeros in columns 1 through $j - 1$. This way, storage saved by very sparse initial columns is provided to the subsequent columns, staying consistent with the assumption that the columns later in the factorization of a *fill-in reduced* matrix should be more crucial for the resulting quality.

Crucially for practice, ICF can operate with *prescribed, almost arbitrarily small* memory overhead, but with some trade-offs. Incomplete Cholesky factorization may break down, and to prevent this, a more robust regularization of A or preprocessing may be necessary. Additionally, compared to the a posteriori sparsified complete factorization, higher allowed density need not result in better approximation.

Implemented variants

Users of SANSA can select from two sparse (approximate) Cholesky factorization variants. When training memory is not severely limiting, the preferable method is SANSA (CHOLMOD), which uses the numerically exact block column code CHOLMOD (Chen et al. [2008]), which is then sparsified to the target density. When training memory is restrictive (as in domains with large item sets), we can use SANSA (ICF), which constructs $\widehat{L}$ by sparsification on the fly using ICF (Lin and Moré [1999]). For SANSA (ICF), users may select to compute denser $\widehat{L}$ to help stabilize the factorization. The factor is sparsified to the target density afterward. Unsurprisingly, when ICF is used for a denser A, we should expect a less accurate approximation due to the loss of information during factorization. This decrease in quality manifested in lower recommendation quality in the experiment in Section 6.4.2. Finally, while CHOLMOD does not require initial construction of the Gram matrix $X^T X$ to compute the decomposition, our implementation of ICF does not support this.[4].

5.3.2 Choice of initial guess

An approximate inverse of a nonsingular matrix A can be found using Schultz iterative process (Schulz [1933]), which computes the next approximation $X^{(k+1)}$ as

$$X^{(k+1)} = X^{(k)}(2I - AX^{(k)}).$$

[4]However, modifying the algorithm to use $X^T X$ implicitly from X is straightforward.

Observe that when starting from initial guess $X^{(0)} = I$, the first iteration step simplifies to

$$X^{(1)} = X^{(0)}(2I - AX^{(0)}) = 2I - A. \tag{5.1}$$

This step is essentially free because it only requires inverting the signs of entries of A and diagonal modification, which are easily performed in place.

The choice of initial guess $X^{(1)}$ suits the inverse incomplete factorization approach from Section 4.4. We identified two supporting arguments, which we thoroughly explain in Appendix A.2. The two observations hint that under certain assumptions (high sparsity of $\widehat{L}$ and wide and shallow elimination tree corresponding to $\widehat{L}$), the initial guess $2I - \widehat{L}$ for Minimal Residual-type methods should be very close to $\widehat{L}^{-1}$, needing only minor refinement. This claim is further supported by the experiments in Sections 6.4.3 and 6.4.4.

5.3.3 Uniform Minimal Residual algorithm

To compute the resulting sparse approximate factor $K \approx \widehat{L}^{-1}$, we use a modification of the MR algorithm (Algorithm 3) with the mentioned choice of initial guess $K^{(0)} = 2I - \widehat{L}$. The modification aims to find more general sparsity patterns and create an approximate inverse where the quality of approximation of individual columns is uniform - hence the name UMR. We achieve this uniformity by performing two types of iterations - *UMR scans* and *UMR finetune steps*, with the main idea being that after a certain number of updates, some of the columns might no longer need further refinement.

UMR scans and finetune steps UMR first performs the specified number of UMR scans. A single UMR scan performs one residual minimization on the entire matrix. Instead of updating each column separately as in Algorithm 3, we run the updates on *batches* of columns to vectorize the computation. As in Algorithm 3, column updates are followed by sparsification. The sparsification keeps only the inverse entries largest in magnitude to preserve the specified target density of the factor. However, contrary to Algorithm 3, we employ *global sparsification* instead of the standard column-by-column (or block-by-block) one. Considering entries throughout the entire matrix allows us to find non-homogeneous sparsity patterns. We briefly experimented with different approaches but observed a considerable improvement in the resulting approximation quality when using global sparsification versus local sparsification strategies.

After the specified number of UMR scans, the algorithm switches to UMR finetune steps. Instead of performing the residual update on the entire matrix (like in a UMR scan), the UMR finetune step selects a batch of the least accurately approximated columns and updates only them. As in UMR scans, global sparsification is used after the updates.

Residual matrix and training loss At the start of each UMR scan and each UMR finetune step, the residual matrix $R^{(i)} = I - \widehat{L}K^{(i)}$ (where $K^{(i)}$ is the current approximation of $\widehat{L}^{-1}$) is computed. The residual matrix expresses a *training loss* to indicate the current approximation quality. Note that the loss is meaningful in our context since the mean of squared column norms of $R^{(i)}$ is the relative residual norm squared:

$$\frac{1}{|\mathcal{I}|}\sum_{j=1}^{|\mathcal{I}|}\|R^{(i)}_{:,j}\|^2 = \frac{1}{|\mathcal{I}|}\|R^{(i)}\|_F^2 = \Big(\frac{\|R^{(i)}\|_F}{\|I\|_F}\Big)^2.$$

A threshold on the training loss is used as a stopping criterion for the iterative process. The norms of columns of $R^{(i)}$ are also used to select columns for modification. Columns with residual norms below a specified threshold[5] (e.g., single precision machine epsilon) are assumed to be inverted correctly and ignored in UMR scans. UMR finetune steps select columns with highest residual norm[6].

Implementation notes In our implementation, UMR scans and finetune steps are performed using batches of columns of dynamically computed size. This allows us to limit the memory cost of computation to a small multiple of the prescribed model size. On the other hand, the computation of the residual matrix $R^{(i)}$ and the training loss is not performed in small batches. This choice is deliberate because being able to compute the product of two sparse matrices is a reasonable assumption. Unless the matrices are huge, single-step computation of the residual matrix adds only a modest memory overhead but saves a small amount of time. *If*, however, memory is the limiting factor, we can eliminate most of the memory overhead in the current implementation

- by computing the residual matrix $R^{(i)}$ in batches, and
- by extracting columns of X^TX from X during the incomplete factorization (as mentioned Section 5.3.1)

The above changes will make our already memory-efficient implementation (see Section 6.4.4) even cheaper.

5.3.4 Training procedure

Training the encoder The first step of the training procedure is the computation of a sparse approximate LDL^T decomposition of the matrix $A = X^TX + \lambda I$. As explained in Section 5.3.1, we need first to find a fill-in reducing permutation of matrix A (represented by matrix P). In our implementation, we use the COLAMD algorithm (Davis et al. [2004]) for this objective. Then, we compute a sparse approximate Cholesky decomposition of a symmetrically permuted matrix PAP^T. The training algorithm provides two possible methods:

1. exact Cholesky decomposition using the CHOLMOD algorithm (Chen et al. [2008]) with subsequent sparsification of the computed factor, or
2. an incomplete Cholesky factorization via the ICF algorithm (Lin and Moré [1999]).

[5]The threshold starts large (only the worst columns are fixed) and decreases in later iterations.

[6]More precisely: since the inverted matrix is lower triangular, we select columns with large *length-normalized* residuals. During experimentation, this approach appeared to improve the convergence speed.

Denoting L_0 the computed factor, this step yields a decomposition in the form $L_0 L_0^T \approx PAP^T$. Next, we extract the diagonal vector $\vec{d_0}$ of L_0 and rescale the columns of L_0 by dividing the j-th column by the j-th entry of $\vec{d_0}$. Note that all entries of $\vec{d_0}$ are nonzero; otherwise, the factorization would have failed. Denote $\widehat{L}$ the rescaled version of L_0 and $\widehat{D} = \mathrm{diag}(\vec{d_0}^2)$ (where the second power is applied element-wise). At this point, we have computed a permutation vector defining matrix P, a dense vector of the diagonal of $\widehat{D}$ and a lower triangular matrix $\widehat{L}$ with unit diagonal such that

$$\widehat{L}\widehat{D}\widehat{L}^T \approx PAP^T.$$

In the second step, we compute the diagonal of $\widehat{D}^{-1}$ by simply inverting the elements of $\widehat{D}$.

The third step computes the initial guess $K^{(0)} = 2I - \widehat{L}$ for the approximate inverse of $\widehat{L}$. Algorithmically, the initial guess $K^{(0)}$ is obtained from $\widehat{L}$ by simply inverting the signs of the subdiagonal entries, an essentially free step. As discussed in Section 5.3.2, this is equivalent to using a step of the Schultz iterative method (Schulz [1933]) applied on the initial guess I.

If $K^{(0)}$ is not an accurate enough approximation of $\widehat{L}^{-1}$, the fourth step of the encoder training refines it using the iterative algorithm UMR described in Section 5.3.3. UMR first performs the specified number of UMR scans updating the entire matrix and then performs the specified number of UMR finetune steps, which target the least accurately approximated columns. Note that at the start of each UMR scan, and each UMR finetune step, the residual matrix $R^{(i)} = I - \widehat{L}K^{(i)}$ (where $K^{(i)}$ is the current approximation of $\widehat{L}^{-1}$) is computed.

After the iterative process stops, we have a sparse $K \approx \widehat{L}^{-1}$, a dense vector representing $\widehat{D}^{-1}$, and a vector representing P such that

$$P^T K^T \widehat{D}^{-1} K P \approx P^T \widehat{L}^{-T} \widehat{D}^{-1} \widehat{L}^{-1} P \approx A^{-1}.$$

Finally, the encoder layer of SANSA, the matrix W^T, is obtained by transposing the column-permuted approximate inverse K:

$$W^T = (KP)^T$$

Note that W^T is a full-rank sparse matrix but no longer lower triangular. Summarizing, the factorized approximation of A^{-1} is expressed using the encoder W^T and the diagonal matrix $\widehat{D}^{-1}$ as $W^T \widehat{D}^{-1} W \approx A^{-1}$.

Building the decoder We use the above factorization to obtain an approximation of the matrix $\widehat{B_{diag}}$ from EASE$^{\mathrm{R}}$ (the matrix of weights with extracted residual connection, see Section 3.1.2). We need to apply column scaling to our factorized matrix. In EASE$^{\mathrm{R}}$, the j-th column of $\widehat{B_{diag}}$ is obtained by scaling the the j-th column of A^{-1} by $-A^{-1}_{jj}$. Using the diagonal vector $\vec{q}$ of matrix $W^T \widehat{D}^{-1} W$, we express the approximation of $\widehat{B_{diag}}$ as

$$\widehat{B_{diag}} \approx -W^T \widehat{D}^{-1} W \mathrm{diag}(\vec{q})^{-1}.$$

Using the associativity of matrix multiplication, we may rewrite the above expression as

$$\widehat{B_{diag}} \approx W^T Z,$$

where

$$Z = -(\widehat{D}^{-1}W)\mathrm{diag}(\vec{q})^{-1}.$$

This expression shows that the scaling can be applied non-symmetrically - only to the decoder part. Hence, SANSA is a *nonsymmetric* autoencoder with encoder layer W^T and decoder layer Z and $\widehat{B_{diag}} \approx W^T Z$.

We end this section by showing how the diagonal vector $\vec{q}$ of matrix $W^T\widehat{D}^{-1}W$ can be computed simultaneously with the matrix decoder layer. First, the rows of W are scaled by the corresponding diagonal entries of $\widehat{D}^{-1}$, all of which are *nonzero.* The resulting matrix $Z_0 = \widehat{D}^{-1}W$ has the same sparsity pattern as W, and hence we can cheaply compute the diagonal of matrix $W^T Z_0$ via the following procedure:

1. Create a copy of matrix W.
2. Multiply the values of W by the values of Z_0.
3. Reduce-sum along the columns to obtain the vector $\vec{q}$.

Remark. While Step 1 is cheap in general and Step 3 is cheap when the matrix is stored using a column-oriented storage format (e.g., CSC), the second step is efficient only when both matrices have the same sparsity pattern, which, luckily, is the case here.

Finally, we scale the columns of Z_0 by $-\vec{q}$ to obtain the decoder matrix Z. We summarize the training procedure in Algorithm 5.3.4.

Algorithm 5 SANSA (Spišák et al. [2023])

input user-item interaction matrix X, L2 regularization λ
Compute $LDL^T \approx P(X^TX + \lambda I)P^T$ (for some permutation matrix P)
Compute $K \approx L^{-1}$
$W \leftarrow KP$
$Z_0 \leftarrow D^{-1}W$
$\vec{q} \leftarrow \mathrm{diag}(W^T Z_0)$
$Z \leftarrow$ scale the columns of Z_0 by $-\vec{q}$
return W^T, Z

5.4 Implementation

We implemented SANSA in Python 3.10 using common libraries `NumPy`[7], `SciPy`[8] (in particular its `sparse` module), `Numba`[9], `pandas`[10] and `scikit-learn`[11]. Moreover, as mentioned, SANSA (CHOLMOD) computes Cholesky decomposition using CHOLMOD Chen et al. [2008], and SANSA (ICF) uses COLAMD Davis et al. [2004] to compute the fill-reducing permutation. CHOLMOD and COLAMD

[7] `https://numpy.org`
[8] `https://scipy.org`
[9] `https://numba.pydata.org`
[10] `https://pandas.pydata.org`
[11] `https://scikit-learn.org`

are available as parts of SuiteSparse[12]. The Python interface for SuiteSparse is provided by `scikit-sparse`[13]. Additionally, we implemented two models as baselines for comparison:

1. the original model EASE^{R}, and
2. MRF, the other sparse approximate variant of EASE^{R} (see Section 3.4.1).

For MRF, we used the code available at `https://github.com/hasteck/MRF_NeurIPS_2019` with minor (primarily organizational) modifications.

The repository is available at `https://github.com/matospiso/sansa` (see Appendix A.3 for more information).

[12]`https://people.engr.tamu.edu/davis/suitesparse.html`
[13]`https://github.com/scikit-sparse/scikit-sparse`

6. Experiments

In the previous chapter, we proposed two methods for constructing sparse approximations of EASE^R to facilitate the construction of creating a smaller model in order to reduce inference-time memory requirements. The two methods suit different RS scenarios, depending on whether *training* memory limitations play a decisive role. In our experiments, we address both scenarios and, for logical coherence, validate the robustness of the more accurate but also more demanding SANSA (CHOLMOD) approach in Section 6.4.2. Later, in Section 6.4.4, we demonstrate our main contribution – the unparalleled scalability of SANSA (ICF) to domains with large item sets.

We published the experiments from this section in our short paper (Spišák et al. [2023]). All experiment codes along with the instructions on how to replicate the experiments, the results, and experiment logs are available at `https://github.com/matospiso/sansa` (see also Appendix A.3).

6.1 Datasets

The evaluation was conducted on five popular datasets from various media domains, which were preprocessed and filtered for certain activity levels of users and items:

- **MovieLens 20M (ML-20M)** (Harper and Konstan [2015]) is a dataset of user-movie ratings collected from a movie recommendation service. For preprocessing, we follow the method of Liang et al. [2018]. Explicit feedback entries are binarized by keeping ratings of four or higher and interpreted as implicit feedback. Users with fewer than five rated movies are filtered out. The resulting preprocessed dataset contains 136 677 users, 20 720 movies, and circa 10 million interactions.

- **Netflix Prize (Netflix)** (Bennett and Lanning [2006]) is a dataset of user-movie rating data from the famous Netflix Prize. We follow the preprocessing method of Liang et al. [2018], which is the same as for the ML-20M dataset: convert the ratings to implicit feedback by keeping ratings of four or higher and filter out users with fewer than five rated movies. The final preprocessed dataset contains 463 435 users, 17 769 items, and about 57 million interactions.

- **Million Song Dataset (MSD)** (Bertin-Mahieux et al. [2011]) contains user-song play counts. We again follow the preprocessing as described by Liang et al. [2018]. The play counts are binarized and interpreted as implicit feedback. We only keep songs that are listened to by at least 200 users and then filter out users with fewer than 20 songs in their listening history[1]. The preprocessed dataset contains 571 355 users, 41 140 items, and about 34 million interactions.

[1] The order of filtering operations is missing in the original preprocessing description.

- **Goodbooks-10k** (Zajac [2017]) is a dataset of user-book ratings. We preprocess the data according to Vančura et al. [2022], who used the same preprocessing as Liang et al. [2018] for ML-20M and Netflix datasets (see above points). The resulting dataset contains 53 366 users, 10 000 items, and about 4 million interactions.

- **Amazon Books** dataset is a part of the Amazon review data (Ni et al. [2019]), a widely used dataset for product recommendation. To correspond with a popular recommendation benchmark BarsMatch (Community [2023]), we use the version of Amazon Books preprocessed according to Wang et al. [2019], which filters out users and items with less than ten interactions. The interactions are treated as implicit feedback. The exact version of the dataset is publicly available and contains 52 643 users, 91 599 items, and about 3 million interactions.

6.1.1 Splits

To remain consistent with well-known benchmarks (i.e., with the many papers referencing Liang et al. [2018] and with BarsMatch, which uses the evaluation protocol of Wang et al. [2019]), we use the evaluation setup as described in the original papers. The papers differ in their choice of data-splitting procedure. Figure 6.1 illustrates two possible approaches. **Strong generalization** means *horizontal* splitting – the training, validation, and test sets are disjoint in terms of users. The other possibility is **weak generalization** (*vertical* splitting), where the training sets are disjoint in user-item interactions but not in terms of users. Strong generalization is relatively more difficult than weak generalization, where the user's click history also appears during the training. Liang et al. [2018] consider it more realistic and robust, but it has a caveat: strong generalization requires feedback from many unique users. Otherwise, removing a part of the users for training will negatively affect the resulting model.

- To agree with Vančura et al. [2022] and Liang et al. [2018], the evaluation for Goodbooks-10k, ML-20M, Netflix, and MSD was conducted in terms of *strong generalization*. For every user in validation and test splits, 20% of interactions were filtered and used as prediction targets. For future work, it is worth considering that *randomly* selecting target interactions does not accurately reflect the real-world task of recommending items based on *previous* interactions. A more realistic approach would involve selecting a percentage of the *newest* interactions as targets.

- To agree with BarsMatch (Community [2023]) and Wang et al. [2019], we use *weak generalization* for Amazon Books, where 80% of interactions of each user are selected for the training set and the remaining 20% form the test set. We use the exact training and test splits by Wang et al. [2019], which are publicly available. Moreover, we randomly select 10% of the training interactions for each user for the validation set. Test set entries serve as targets for test predictions, which use feedback in the training and validation sets as input (see Figure 6.1).

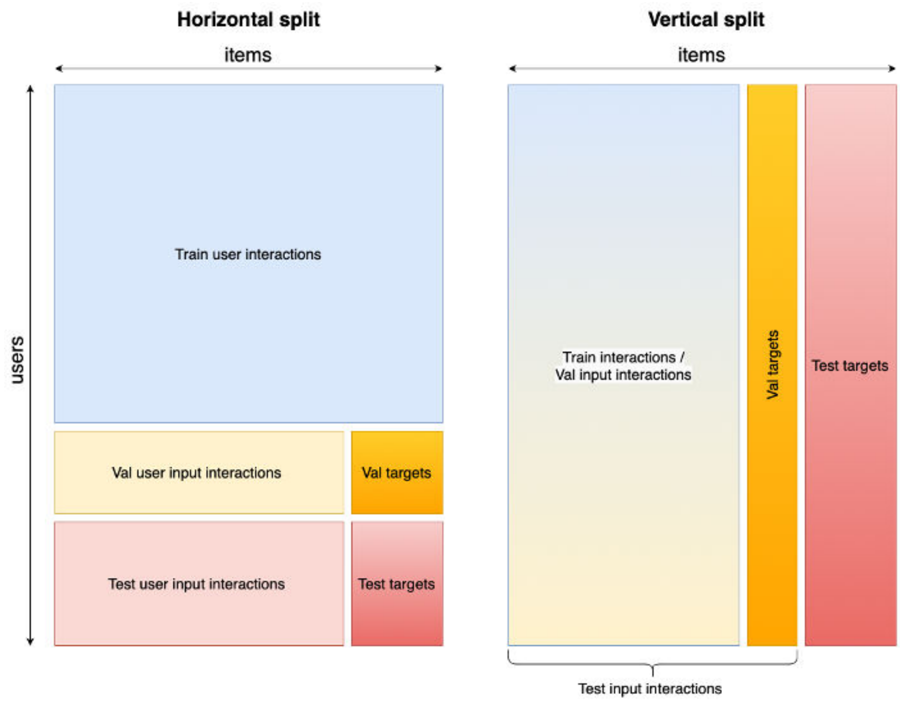


Figure 6.1: Comparison of data splitting approaches. Horizontal splitting (also called strong generalization) creates training, validation, and test sets with disjoint user sets. The other possible way is to use vertical splitting (weak generalization). The figure visualizes only the "directions" of splitting and the resulting proportions; target items vary for different users.

For reproducibility, we implemented an experiment pipeline that selects the correct preprocessing and splitting procedure for each of the five datasets. Moreover, the `README` file in our repository also includes instructions on how to obtain the correct dataset files for the pipeline input.

Based on the sizes of item sets and densities of the item relation graphs corresponding to the training splits, we divide the datasets into three categories:

- Goodbooks-10k, ML-20M, Netflix are **small and dense** datasets. They contain 10 000 to 20 720 items, which are *densely connected*, since even though the interaction densities in the training set are between 0.35% and 0.77%, the corresponding item-item matrices A used in training have densities in the range of 21.25% to 70.76%.

- MSD is larger than the previous three datasets with 41 140 items, and hence we say it is a **medium-sized, dense** dataset: its training item-item matrix A has a density of 41.54%.

- With 91 599 items, the training set interaction density 0.062%, and item-item matrix density 3.94%, Amazon Books is the closest dataset to an archetypal real-world use case for SANSA – large e-commerce scenarios, where interaction (and consequently also item-item) matrices are enormous and sparse. Therefore, Amazon Books is a **large and sparse** dataset.

dataset	Goodb.-10k	ML-20M	MSD	Netflix	Amazon B.
# of users	48 366	116 677	471 355	383 435	52 643
# of items	10 000	20 720	41 140	17 769	91 599
# of inter.	3 735 397	8 516 174	27 765 348	47 098 414	2 380 730
density (%)	0.7723	0.3523	0.1432	0.6913	0.0619
item-item density (%)	39.5522	21.2540	41.5369	70.7583	3.9371

Table 6.1: Attributes of training splits. The item-item matrices are significantly denser ($X \to X^T X + \lambda I$ increases density, from around 50× for Goodbooks-10k to around 300× for MSD). As a result, the item-item matrices, their Cholesky factors, and the inverse item-item matrices and their factors are dense. However, on even larger domains, datasets, and the matrices and factors can be much sparser, as hinted by Amazon Books, which is the only dataset with a relatively sparse item-item network (this is a consequence of its highly sparse input data).

Table 6.1 summarizes the training set sizes and densities. Note even though the evaluated datasets (i.e., the corresponding training matrices X) are sparse, their item-item matrices $X^T X + \lambda I$ are significantly denser. However, the densities of $X^T X + \lambda I$ will decrease if we increase the item set size and preserve the amount of feedback per user.

6.2 Metrics

We evaluate the model performance using two ranking-based metrics: recall and the normalized discounted cumulative gain (nDCG) at the top k positions. Both metrics compare the predicted rank of target items (determined by sorting according to the predicted scores) with their actual rank for each user.

Mathematically, for a user u, let $\mathtt{p}(u)$ denote the array that contains all items from I sorted by the predicted item scores for this user (i.e., $\mathtt{p}(u)_i$ is the item with the i-th highest score for the user u), and let $I_u \subset I$ denote the set of target items for the user u. Furthermore, for a set S, denote $\mathbb{I}_S(\cdot)$ its indicator function, that is, $\mathbb{I}_S(x) = 1$ if $x \in S$ else $\mathbb{I}_S(x) = 0$.

Definition 6.1 (recall@k)**.** *For a user u, we define*

$$\text{recall@}k(u) = \frac{\sum_{i=1}^{k} \mathbb{I}_{I_u}(\mathtt{p}(u)_i)}{\min(k, |I_u|)}.$$

Remark. The evaluation protocol of Liang et al. [2018] uses the above definition of recall (and hence we also use it for evaluation on Goodbooks-10k, ML-20M, Netflix, and MSD). In this definition, the expression in the denominator is truncated to at most $|I_u|$, which normalizes the metric to obtain a maximum value of 1 when all items recommended within the top k positions are among the target items. There exists another possible definition that does not truncate the denominator:

$$\text{recall@}k(u) = \frac{\sum_{i=1}^{k} \mathbb{I}_{I_u}(\mathtt{p}(u)_i)}{|I_u|}.$$

We use this expression to evaluate Amazon Books to remain consistent with the BarsMatch benchmark (Community [2023]) and Wang et al. [2019].

Personally, I dislike the second definition of recall because the resulting metric is not interpretable when the number of target items for a user is unknown: if $k < |I_u|$, then recall@$k(u) \le \frac{k}{|I_u|} < 1$. In other words, we do not know the maximum attainable value in this situation. When evaluating the performance as the average recall@k over many users (that typically have different numbers of target items), the value loses its upper reference point and interpretability.

While recall@k treats all items within the top k predicted items equally important, nDCG@k incorporates a monotonically increasing discount factor to emphasize the significance of higher ranks over lower ones. This metric, therefore, penalizes incorrect order of recommendations – the best match for a user should be the first recommended item, and so on.

Definition 6.2 (nDCG@k)**.** *For a user u, let $0 \le rel_i \le 1$ denote the relevance of the i-th predicted item*[2]*. The* discounted cumulative gain (DCG) *@ k is defined as*

$$\mathrm{DCG@}k(u) = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)}.$$

The ideal discounted cumulative gain (IDCG) *@ k is defined as*

$$\mathrm{IDCG@}k(u) = \sum_{i=1}^{best_k(u)} \frac{2^{rel_i} - 1}{\log_2(i+1)},$$

where $best_k(u)$ is the k-prefix of the sorted array of relevance values of all items in I (i.e., for $i \le k$, $best_k(u)_i$ is the relevance of the i-th most relevant item). IDCG@$k(u)$ *represents the maximum attainable value of* DCG@k *for the user u. Finally, the* nDCG *@ k is defined as*

$$\mathrm{nDCG@}k(u) = \frac{\mathrm{DCG@}k(u)}{\mathrm{IDCG@}k(u)}.$$

For a user u, similarly to the first definition of recall, the above definition of nDCG@k satisfies $0 \le$ nDCG@$k(u) \le 1$ for all choices of positive integers k and all users u.

Our experiments on Goodbooks-10k, ML-20M, MSD, and Netflix evaluated model accuracy using the same metrics as Liang et al. [2018], i.e., recall@20, recall@50, and nDCG@100. We evaluate models on Amazon Books using the same metrics as the BarsMatch benchmark (Community [2023]) and Wang et al. [2019], i.e., recall@20 (the second kind!) and nDCG@20. Finally, we used the `perf_counter` function from Python's standard module `time` to measure function execution time and the `memory-profiler` module to measure memory utilization during the program execution.

[2] In case of implicit feedback, we consider all target items (equally) relevant with $rel = 1$ and hence we may use $2^{rel_i} - 1 = \mathbb{I}_{I_u}(\mathtt{p}(u)_i)$.

6.3 Baselines

We briefly discuss the two primary baselines used in the experiments.

We trained EASE^R for all datasets except Amazon Books, where we merely report the results of EASE^R from Community [2023] due to extreme computational requirements needed for training. EASE^R uses only a single hyperparameter – L2 regularization – which we keep consistent with the value of L2 regularization used by SANSA (CHOLMOD). Compared with that, SANSA (ICF) uses different regularization due to additional scaling applied to the matrix $X^TX + \lambda I$ used to prevent breakdowns in the incomplete factorization (see Lin and Moré [1999]). Apart from L2 regularization, both variants of SANSA use only a parameter for prescribing weight density and parameters for setting the number of iterations, which can be selected by observing training loss.

Additionally, we compare SANSA with the scalable modification of EASE^R proposed by the same author – MRF (Steck [2019b]; see our overview in Section 3.4.1). The MRF method, as the only other full-rank sparse modification of EASE^R, is the current state-of-the-art. The hyperparameter selection for MRF is complex; hence, we often reuse the parameters from the code accompanying the original paper. Unlike EASE^R and SANSA (CHOLMOD), and similarly to SANSA (ICF), MRF *normalizes* its item-item matrix $X^TX + \lambda I$ and a different L2 regularization is needed compared to EASE^R. On each dataset, we trained MRF with different choices of r (typically $r \in \{0, 0.1, 0.5\}$). Furthermore, we used $\alpha = 0.75$ and kept the `maxInColumn` at 1000. For direct comparison, we selected `threshold` on all datasets by trial and error so that the densities of models correspond to the ones selected for SANSA.

Model configurations used in each experiment are available in the repository[3].

6.4 Results

6.4.1 Robustness on small, dense datasets

For the first experiment, we compare the recommendation accuracy of SANSA (CHOLMOD) against EASE^R and MRF on Goodbooks-10k, MovieLens 20M and Netflix. The purpose of this experiment is to verify the robustness of SANSA (CHOLMOD) in situations where the item set size is not large but where the corresponding item-item network is densely connected (see the density of item-item network in Table 6.1). These scenarios challenge the sparse approximation approach. Since $X^TX + \lambda I$ is dense, it could happen that no good sparse approximation of $(X^TX + \lambda I)^{-1}$ exists. Moreover, the Cholesky factors of $X^TX + \lambda I$ will also be dense. Luckily, in real-world recommender system applications and with enough user feedback sampled, $X^TX + \lambda I$ will be close to diagonally dominant; typically, only a few nondiagonal entries will be comparable or larger in magnitude than the diagonal entries. Then, the computed Cholesky factor should have relatively few large subdiagonal entries, and we should be able to find good sparse approximations of the factor and its inverse.

The experiment results in Table 6.2 reveal that EASE^R (a cutting-edge CF model, see, e.g., Vančura et al. [2022]) can be accurately approximated by 50

[3] `https://github.com/matospiso/sansa`

density	model	recall@20	recall@50	nDCG@100
Goodbooks-10k				
0.5%	MRF ($r = 0$)	0.364	0.497	0.507
	MRF ($r = 0.5$)	0.350	0.488	0.490
	SANSA (CHOLMOD)	0.355	0.493	0.499
1.0%	MRF ($r = 0$)	0.363	0.501	0.508
	MRF ($r = 0.5$)	0.352	0.492	0.493
	SANSA (CHOLMOD)	0.356	0.494	0.500
2.0%	MRF ($r = 0$)	0.363	0.501	0.507
	MRF ($r = 0.5$)	0.354	0.493	0.495
	SANSA (CHOLMOD)	0.357	0.498	0.501
	EASE^{R}	0.357	0.494	0.499
MovieLens 20M				
0.5%	MRF ($r = 0$)	0.390	0.515	0.420
	MRF ($r = 0.5$)	0.380	0.508	0.412
	SANSA (CHOLMOD)	0.383	0.512	0.413
1.0%	MRF ($r = 0$)	0.390	0.516	0.420
	MRF ($r = 0.5$)	0.385	0.513	0.416
	SANSA (CHOLMOD)	0.386	0.516	0.417
2.0%	MRF ($r = 0$)	0.390	0.516	0.420
	MRF ($r = 0.5$)	0.386	0.515	0.418
	SANSA (CHOLMOD)	0.388	0.518	0.420
	EASE^{R}	0.392	0.521	0.422
Netflix Prize				
0.5%	MRF ($r = 0$)	0.360	0.441	0.391
	MRF ($r = 0.5$)	0.352	0.435	0.385
	SANSA (CHOLMOD)	0.354	0.433	0.383
1.0%	MRF ($r = 0$)	0.362	0.442	0.392
	MRF ($r = 0.5$)	0.357	0.439	0.389
	SANSA (CHOLMOD)	0.354	0.435	0.384
2.0%	MRF ($r = 0$)	0.362	0.443	0.392
	MRF ($r = 0.5$)	0.359	0.442	0.391
	SANSA (CHOLMOD)	0.356	0.438	0.386
	EASE^{R}	0.362	0.444	0.393

Table 6.2: Recommendation accuracy on small, dense datasets. MRF and SANSA (CHOLMOD) match performance of EASE^{R} even at very high compression levels. The standard errors in the experiments are about 0.004, 0.003, and 0.001 in the Goodbooks-10k, ML-20M, and Netflix experiments, respectively.

to 200 times sparser full-rank models even on these densely connected domains. This provides the first empirical evidence for the robustness of our approach.

There is an interesting practical implication: on domains of this type, it is feasible to create very small models[4] that provide sufficiently accurate recommendations even standalone or add these mini models to ensemble RS at little to no extra cost.

6.4.2 Robustness and efficiency on medium-sized, dense dataset

For the second experiment, we evaluate the accuracy and training time of the a posteriori sparsified SANSA (CHOLMOD) variant in a more computationally demanding setting. We selected MSD for this test since it is among the largest benchmarks for CF with 41 140 items, yet it is still small enough to allow the complete sparse factorization on a moderately sized `m6i.4xlarge` instance with 64 GB RAM. Additionally, the dense connectedness of MSD challenges the robustness and efficiency of sparse approaches. Although the interaction density is only 0.14%, the item-item matrix $X^T X + \lambda I$ is 41.54% dense (see Table 6.1).

The results in Table 6.3 show the robustness of SANSA: if we allow sufficient weight density and train long enough, SANSA (CHOLMOD) will achieve the same accuracy as dense EASE$^{\text{R}}$. Table 6.3 and Figure 6.2 also illustrate that the quality of approximation of EASE$^{\text{R}}$ improves as we allow higher weight density of SANSA (CHOLMOD). Furthermore, we see that even very sparse models can approximate EASE$^{\text{R}}$ with high accuracy. Despite the high density of the item-item network, even 50 – 1 000 times sparser full-rank models perform on par with EASE$^{\text{R}}$.

In terms of accuracy, SANSA (CHOLMOD) performs in between MRF ($r = 0$) and pruned MRF ($r = 0.5$) on all tested density levels, see Figure 6.2. [5] At 0.1% density, imposing the computed sparsity pattern is inexpensive, and MRF trains three to four times faster than SANSA (CHOLMOD). However, masking operations on sparse matrices (essential for MRF training) do not scale well as the number of nonzeros increases. Consequently, training MRF becomes expensive as the total number of nonzero elements in the approximation increases since r-pruning does not help with this problem. Compared with that, SANSA performs its training using efficient sparse matrix operations[6] and at 2% density, SANSA (CHOLMOD) trains almost as fast as the pruned MRF ($r = 0.5$). For completeness, SANSA (ICF) performed about 6-17% worse than SANSA (CHOLMOD) on MSD, but this is anticipated: incomplete factorization of a dense matrix loses information during computation and requires more robust regularization to prevent breakdowns.

Finally, vectors predicted by a *two-layer* SANSA are significantly denser than the vectors produced by a single-layer sparse model MRF with equal density. Therefore, SANSA can recommend more items from sparse inputs than MRF, which may be desirable in practice.

[4] SANSA and MRF with weight density 0.5% on MovieLens 20M have only around 2 million parameters.

[5] A more accurate comparison is difficult, as MRF uses additional data normalization to tackle popularity bias, see Steck [2019b] and Section 3.4.1.

[6] Cholesky factorization, sparse matrix-matrix multiplications, sparsifications and element-wise operations working on contiguous memory, and (block) column manipulations.

Million Song Dataset					
density	model	recall@20	recall@50	nDCG@100	training time
	MRF ($r = 0$)	0.330	0.421	0.385	64 s
0.1%	MRF ($r = 0.5$)	0.326	0.417	0.380	55 s
	SANSA (CH.)	0.328	0.422	0.383	200 s
	SANSA (ICF)	0.288	0.385	0.346	190 s
	MRF ($r = 0$)	0.333	0.427	0.389	183 s
0.5%	MRF ($r = 0.5$)	0.329	0.424	0.384	90 s
	SANSA (CH.)	0.331	0.426	0.387	253 s
	SANSA (ICF)	0.276	0.370	0.337	632 s
	MRF ($r = 0$)	0.333	0.428	0.390	1031 s
2.0%	MRF ($r = 0.5$)	0.329	0.426	0.385	457 s
	SANSA (CH.)	0.332	0.427	0.388	502 s
	SANSA (ICF)	0.298	0.399	0.359	528 s
	EASE^{R}	0.332	0.428	0.388	312 s
results from Shenbin et al. [2020] and Liang et al. [2018]					
	RECVAE	0.276	0.374	0.326	——
	WMF	0.257	0.312	0.257	——
	MULT-VAE^{PR}	0.266	0.364	0.316	——

Table 6.3: Highly compressed SANSA (CHOLMOD) and MRF models achieve accuracy comparable to EASE^{R} even on dense datasets. As the number of nonzeros in the approximation increases, imposing sparsity via masking becomes a performance bottleneck for MRF; SANSA uses efficient sparse operations and scales better. At 0.5% density, incomplete factorization suffered breakdowns and required restarts with diagonal shifts. The diagonal shifts introduced additional regularization, resulting in decreased recommendation accuracy. The standard error is about 0.001.

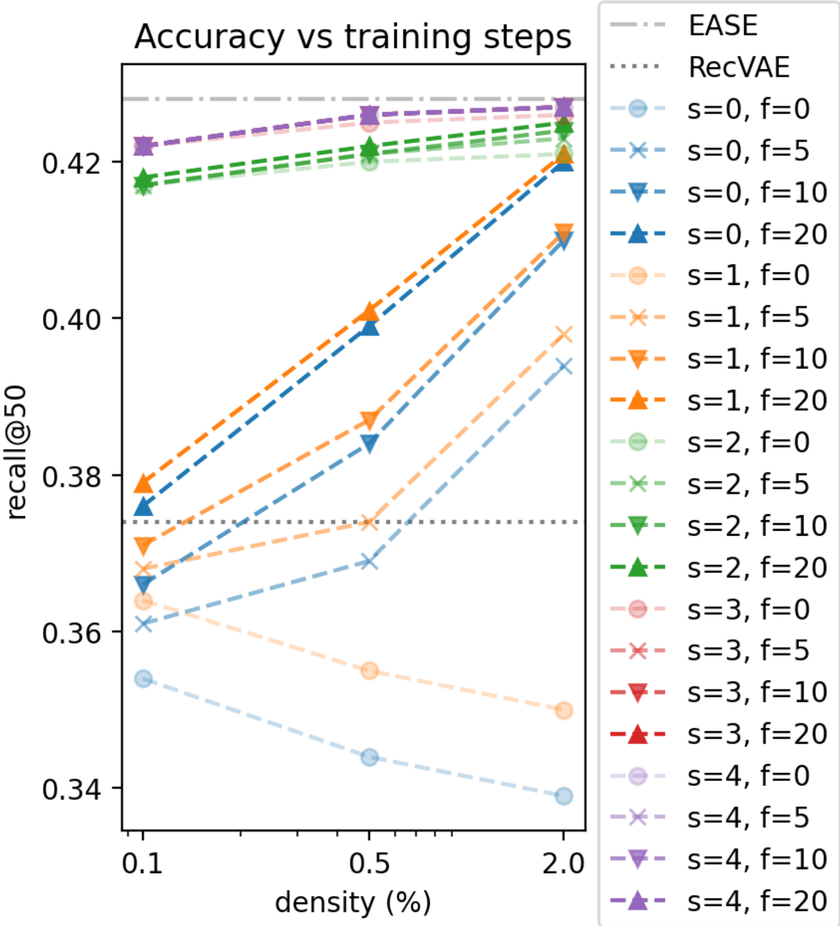


Figure 6.2: Accuracy of SANSA (CHOLMOD) on MSD after various numbers of training scans s and finetune steps f. The initial guess for UMR provides an approximation of EASE^{R} with recommendation accuracy comparable to that of other low-rank models; see also Table 6.3. Even very few short UMR iterations can push the performance close to that of EASE^{R}. (Spišák et al. [2023])

6.4.3 Trading accuracy for shorter training

In larger domains, trading recommendation accuracy for shorter training may be desirable. For example, MRF uses parameter r to prune dependencies between item clusters. SANSA provides a similar (although less interpretable) possibility: applying fewer UMR iterations yields a coarser approximation of the weight matrix of EASE^{R}. To analyze the trade-off, we compared checkpoints of SANSA (CHOLMOD) trained for different numbers of UMR scans s and finetune steps f against two baselines: EASE^{R} to measure the distance from the uncompressed model (i.e., the qualitative decrease compared to no compression), and a deep variational autoencoder RECVAE Shenbin et al. [2020], ranked second on MSD according to Shenbin et al. [2020]. For perspective, we include the reported performance of RECVAE in Table 6.3 along with two more relevant baselines[7]: a linear *low-rank* factorization model WMF (Hu et al. [2008]) and a multinomial variational autoencoder $\text{MULT-VAE}^{\text{PR}}$ (Liang et al. [2018]).

[7] As reported by Liang et al. [2018].

The results in Figure 6.2 show that even short training can produce a close approximation of the dense EASE^R. Notably, we can train very sparse models for but a few short UMR iterations and obtain performance close to state-of-the-art. Various early checkpoints of SANSA (CHOLMOD) (Figure 6.2) and even the ICF variant outperform other competing models on MSD (see Table 6.3). To conclude, very sparse or coarse approximations of EASE^R can be competitive yet very cheap and practical (e.g., for ensemble models).

6.4.4 Extreme scalability

In the final experiment, we demonstrate the main contribution of this thesis and our paper: the ability of SANSA (ICF) to scale to extremely large datasets. We select Amazon Books for this experiment because it is the largest and sparsest popular benchmark dataset with 91 599 items, interaction density 0.062%, and item-item matrix density 3.94%. Due to its proportions, Amazon Books proves very challenging even for state-of-the-art recommender systems and, at the same time, is the nearest benchmark to the intended production use cases of SANSA. We empirically demonstrate our method's scalability and efficiency by measuring training time and memory requirements using tools mentioned at the end of Section 6.2. As for the recommendation accuracy, we follow the evaluation protocol of the BarsMatch benchmark (Community [2023]) and measure recall@20[8] and nDCG@20 for reproducibility. We compare SANSA (ICF) with MRF and the results of other state-of-the-art models from Community [2023].

In domains with very large item sets, we can exploit the inevitable sparsity of dominant information in the item-item network (i.e., the sparsity of $X^T X$; see the properties of Amazon Books in Table 6.1) to avoid restrictive memory requirements (otherwise inevitable on such large item sets) and create a strongly compressed model without losing important information. To elaborate, when $X^T X + \lambda I$ is sparse, we will likely find a good fill-reducing permutation so that the resulting Cholesky factor L is likely sparse, too, and the sparse incomplete factor $\hat{L}$ computed by ICF should be close to L (see Section 5.3.1). Moreover, when the elimination tree associated with $\hat{L}$ is wide, the free initial guess $2I - \hat{L}$ is very close to the exact $\hat{L}^{-1}$, needing little to no refinement (see Section 5.3.2 and references therein). As a result, our method essentially reduces the sparse approximate inversion to a cheaply obtained incomplete factorization. This shortcut drastically reduces training time and memory requirements.

Therefore, it is by no surprise that on Amazon Books, SANSA (ICF) with about 0.84 million parameters (i.e., 10 000 times compressed compared to the dense EASE^R) trains orders of magnitude faster than any other non-sparse state-of-the-art method (dense autoencoders, nearest neighbors approaches or graph neural networks), as shown in Table 6.4. Furthermore, SANSA (ICF) trains more than three times faster than MRF with equally sparse weights – partially due to the discussed performance bottleneck caused by masking but also due to the large number of leading clusters requiring inversion. Moreover, this speedup was achieved on a single-core `r6i.large` instance (2 vCPUs, 16 GB RAM), which is much smaller compared to `r6i.4xlarge` with 8 cores (16 vCPUs, 128 GB RAM)

[8] According to the second definition, see Section 6.2.

Amazon Books							
				results from Community [2023]:			
	SANSA (ICF)	MRF ($r = 0$)	MRF ($r = 0.5$)	EASE^{R}	SLIM	ITEM-CF	ULTRA-GCN
recall@20	**0.077**	0.071	0.069	0.071	0.075	0.074	0.068
nDCG@20	**0.064**	0.058	0.055	0.057	0.060	0.061	0.056
TRAINING RESOURCES							
vCPU	**2**	16	16	28	28	28	20*
memory usage (GB):							
peak	**9.18**	96.45	96.58	—— not measured; costly ——			
average	**3.87**	49.12	49.75	—— not measured; costly ——			
time	**49 s**	172 s	167 s	222 m	316 m	57 m	45 m

*and RTX 2080 GPU

Table 6.4: Thanks to end-to-end sparse training procedure, SANSA (ICF) trains three times faster using 2 vCPUs than MRF on 16 vCPUs, and orders of magnitude faster than any leading-edge model with dense weights, in particular EASE^{R} and SLIM. The training of SANSA requires minuscule memory, unparalleled even with MRF. As a bonus, it also achieves new state-of-the-art accuracy. The standard error in the accuracy measurements is about 0.0005. (Spišák et al. [2023])

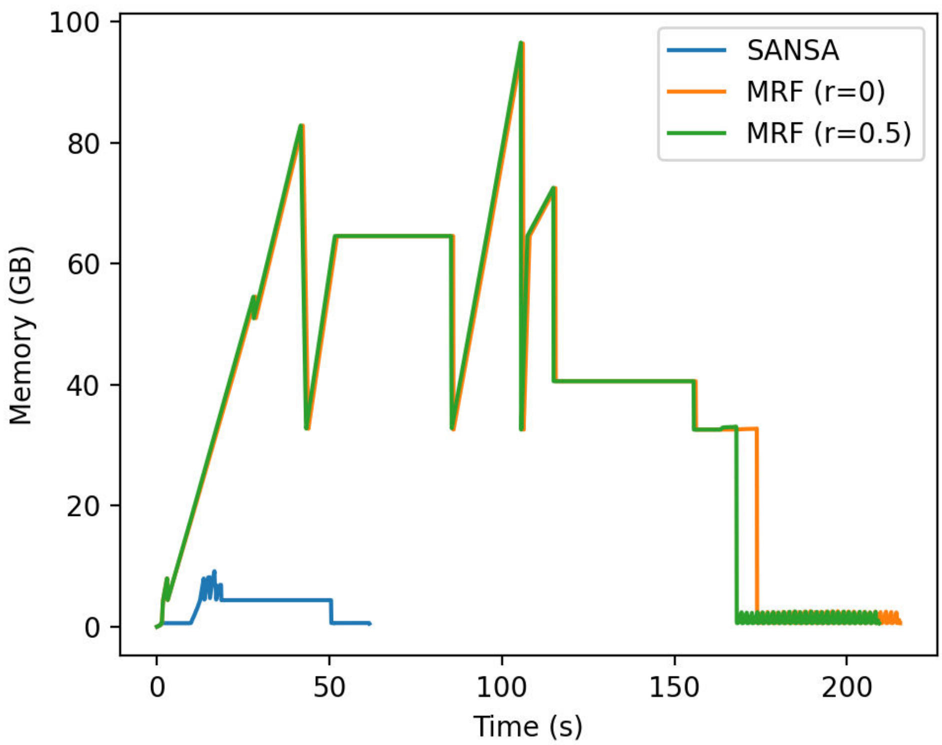


Figure 6.3: Comparison of time and memory usage of SANSA (ICF) versus MRF on Amazon Books. The final ”flatline” on each graph corresponds to the evaluation phase in the pipeline. (Spišák et al. [2023])

needed for MRF[9]. As such, training SANSA (ICF) is much more cost-effective compared to other models, e.g., in terms of total floating-point operations (FLOPs). In addition, thanks to efficient sparse operations, the training procedure of SANSA (ICF) requires ten times less memory (see Table 6.4 and Figure 6.3) than the training of MRF, and this edge can be improved further: our code keeps up to 2 copies of $X^T X$ in memory (which amounts to roughly 8 GB for Amazon Books). This overhead can be eliminated by implicitly constructing $X^T X$ during ICF (see Section 5.3.1). For perspective, SANSA (ICF) for Amazon Books could then be trained on a Raspberry Pi or a smartphone.

Finally, while beating EASE$^{\text{R}}$ in terms of accuracy was never our goal, we surpassed its reported performance on Amazon Books by a non-trivial margin and, with it, the current state-of-the-art. Since MRF does not outperform EASE$^{\text{R}}$ in our experiment, we do not think the accuracy improvement is due to "*reducing the number of trust-busters in the sense of eliminating generally popular items that are unrelated to the user's interests*," as proposed by Steck [2019c]. However, SANSA (ICF) differs from EASE$^{\text{R}}$ (and SANSA (CHOLMOD) and MRF) in scaling (and, hence, also in the used L2 regularization). We include the scaling in our code to help stabilize the incomplete factorization (refer to the original paper by Lin and Moré [1999]). Considering time constraints, we were unable to examine the impact of the scaling on recommendations or explore its interpretation; we leave this for future research.

[9] MRF cannot be tested on a smaller instance due to training memory requirements approaching 100 GB. Instances used in Community [2023] are even larger, with up to 28 vCPUs and more than 500 GB of RAM, or 20 vCPUs and a GPU.

Conclusion

This thesis proposes a solution to the challenge of scaling state-of-the-art collaborative filtering models to domains with large item catalogs. The solution we propose is based on the linear model EASE^{R} by Steck [2019a]. Compared with other approaches, EASE^{R} is able to use long-distance information in the bipartite network of user feedback. This information is crucial for accurate and diverse CF modeling but expensive to extract and store on vast domains.

Our main contribution is a robust and efficient method to compute a sparse approximation of the potentially extensive model EASE^{R} using contemporary numerical methods for sparse approximate inversion. The robustness of the approach stems from the ability of modern approximate inversion techniques to reliably find dominant inverse entries (and, hence, also dominant weights of EASE^{R}). The method offers strong compression, further improved by factorization. In terms of efficiency, our end-to-end sparse method is better suited for the task than previous approaches that attempt to overpower the problem using operations tailored to dense structures. Consequently, the resulting model, SANSA, trains faster and with minuscule memory requirements.

To summarize, SANSA provides a robust yet attainable baseline capable of scaling to millions of items for researchers and industry applications.

Future work

The experiments carried out in this thesis provide evidence for the viability of employing the SANSA approach in building accurate sparse autoencoders, as well as in generating sparse approximate inverses of large-dimensional sparse SPD matrices. We intend to conduct further, more specific experiments to deepen our understanding of the method's behavior. Moreover, even though the proposed method considerably improves the scalability of state-of-the-art collaborative filtering, we see room for improvement in several aspects of the approach. Additionally, we see a few applications for the method outside the recommender system domain.

Experiments The next step is to evaluate SANSA in experiments on even larger and sparser datasets, in an online setting, and against other baselines. Examining metrics beyond accuracy (such as diversity) is also important. We want to assess the method's behavior (and, for example, the effect of sparsification) on various user segments and compare observations with other state-of-the-art benchmarks. Additionally, it would be interesting to test how a full-rank, sparse approximation of EASE^{R} (SANSA) performs in an online experiment with an extensive set of items for recommendation (i.e., hundreds of thousands or millions of items) compared to a low-rank, dense approximation like ELSA. Such an experiment could reveal whether the use of a full-rank component in SANSA provides a decisive advantage over low-rank models as theorized and if this advantage is a consequence of high catalog entropy, as argued by Steck and Liang [2021].

In order to further test the scalability of SANSA, we conducted an additional test on a much larger scale, surpassing the experiments in this thesis (which all involved fewer than a hundred thousand items). The test involved approximately

2 million items and around 50 million interactions. Remarkably, under these conditions, we were able to train SANSA (ICF) in less than an hour, with training memory requirements in lower tens of gigabytes.

Method improvements Our long-term goal is to enhance the method by introducing parallel reorderings and incomplete factorization. One approach we plan to explore is tree parallelism for ICF based on a nested dissection technique (see, e.g., the papers by Lipton et al. [1979] and Karypis and Kumar [1998]). This technique involves splitting the computation into multiple independent subproblems, each assigned to a separate thread, then remerging them using small separators. Doing so can achieve a notable speedup in the factorization process with minimal additional memory requirements. Additionally, the nested dissection approach should reveal an "arterial structure" of the item-item network. Understanding this structure and the latent embeddings produced by SANSA will contribute to improved interpretability. However, most of the current computation time overhead can be eliminated by simply switching to a parallel algorithm for finding a fill-in reducing permutation. We expect this strategy to enable the SANSA method to handle even larger collaborative filtering tasks.

We also suggest combining SANSA with a nonlinear component, e.g., for the subset of most popular items. The added part would allow the model to learn nonlinear dependencies for at least part of the items to fix the residual error of the linear model.

Other use cases Finally, based on the results of the thesis, we recognize three possible applications for our approach beyond recommender systems:

1. Since the original idea of the method comes from preconditioners, it will be interesting to see how the method used to train SANSA performs in preconditioning tasks on very large, SPD problems with general sparsity patterns.

2. Furthermore, our approach could offer superior efficiency compared to existing methods, such as the one used by Steck [2019b], for estimating large inverse covariance matrices in statistical problems.

3. Finally, we believe scalable sparse autoencoders will prove helpful in natural language applications.

Bibliography

Onureena Banerjee, Laurent El Ghaoui, and Alexandre d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *J. Mach. Learn. Res.*, 9:485–516, 2008. URL `http://dblp.uni-trier.de/db/journals/jmlr/jmlr9.html#BanerjeeGd08`.

Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2021.

James Bennett and Stanley Lanning. The netflix prize. *Proceedings of KDD Cup and Workshop*, Vol. 2007, 11 2006.

M. W. Benson. Iterative solution of large scale iterative systems, 1973. Master thesis.

M. Benzi and M. Tůma. A comparative study of sparse approximate inverse preconditioners. *ANM*, 30(2-3):305–340, 1999.

M. Benzi, R. Kouhia, and M. Tůma. Stabilized and block approximate inverse preconditioners for problems in solid and structural mechanics. *Comput. Methods Appl. Mech. Engrg.*, 190(49-50):6533–6554, 2001.

Michele Benzi. A direct row-projection method for sparse linear systems. *Ph. D. Thesis, North Carolina State University*, 1993.

Michele Benzi and Miroslav Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 19 (3):968–994, 1998. doi: 10.1137/S1064827595294691. URL `https://doi.org/10.1137/S1064827595294691`.

Michele Benzi, Carl D. Meyer, and Miroslav Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J Sci Comput*, 17: 1135–1149, 07 1996. doi: 10.1137/S1064827594271421.

Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In Anssi Klapuri and Colby Leider, editors, *Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011, Miami, Florida, USA, October 24-28, 2011*, pages 591–596. University of Miami, 2011. URL `http://ismir2011.ismir.net/papers/OS6-1.pdf`.

Christopher Bishop. *Pattern Recognition and Machine Learning.* Springer, January 2006. URL `https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/`.

Å. Björck. *Numerical methods for Least Squares Problems.* SIAM, Philadelphia, 1996.

Caterina Calgaro, Jean-Paul Chehab, and Yousef Saad. Incremental incomplete LU factorizations with applications. *Numer. Linear Algebra Appl.*, 17(5):811–837, 2010. ISSN 1070-5325,1099-1506. doi: 10.1002/nla.756. URL `https://doi.org/10.1002/nla.756`.

Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3), oct 2008. ISSN 0098-3500. doi: 10.1145/1391989.1391995. URL `https://doi.org/10.1145/1391989.1391995`.

Jeongwhan Choi, Seoyoung Hong, Noseong Park, and Sung-Bae Cho. Perturbation-recovery method for recommendation, 2022.

F. Chollet. *Deep Learning with Python, Second Edition*. Manning, 2021. ISBN 9781617296864. URL `https://www.manning.com/books/deep-learning-with-python-second-edition`.

Edmond Chow. *Robust preconditioning for sparse linear systems*. PhD thesis, Department of Computer Science, University of Minnesota, Minneapolis, MN, USA, 1997.

Edmond Chow and Yousef Saad. Experimental study of ilu preconditioners for indefinite matrices. *J. Comput. Appl. Math.*, 86(2):387–414, dec 1997. ISSN 0377-0427. doi: 10.1016/S0377-0427(97)00171-4. URL `https://doi.org/10.1016/S0377-0427(97)00171-4`.

Edmond Chow and Yousef Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM J. Sci. Comput.*, 19(3):995–1023, may 1998. ISSN 1064-8275. doi: 10.1137/S1064827594270415. URL `https://doi.org/10.1137/S1064827594270415`.

Andrzej Cichocki and Anh-Huy Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E92.A(3):708–721, 2009. doi: 10.1587/transfun.E92.A.708.

The BARS Community. Barsmatch: A benchmark for candidate item matching, 2023. URL `https://openbenchmark.github.io/BARS/candidate_matching/`.

Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. ISSN 0747-7171. doi: https://doi.org/10.1016/S0747-7171(08)80013-2. URL `https://www.sciencedirect.com/science/article/pii/S0747717108800132`. Computational algebraic complexity editorial.

J. D. F. Cosgrove, Díaz, and A. Griewank. Approximate inverse preconditioning for sparse linear systems. *Int. J. Comput. Math.*, 44:91–110, 1992.

Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, and Esmond G. Ng. A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):353–376, sep 2004. ISSN 0098-3500. doi: 10.1145/1024074.1024079. URL `https://doi.org/10.1145/1024074.1024079`.

Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, jan 2004. ISSN 1046-8188. doi: 10.1145/963770.963776. URL `https://doi.org/10.1145/963770.963776`.

I. S. Duff, A. M. Erisman, C. W. Gear, and J. K. Reid. Sparsity structure and Gaussian elimination. *ACM SIGNUM Newsletter*, 23(2):2–8, 1988.

I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 01 2017. ISBN 9780198508380. doi: 10.1093/acprof:oso/9780198508380.001.0001. URL `https://doi.org/10.1093/acprof:oso/9780198508380.001.0001`.

Stanley C. Eisenstat, Martin H. Schultz, and Andrew H. Sherman. Algorithms and data structures for sparse symmetric gaussian elimination. *Siam Journal on Scientific and Statistical Computing*, 2:225–237, 1981.

Howard C. Elman. A stability analysis of incomplete lu factorizations. *Mathematics of Computation*, 47(175):191–217, 1986. ISSN 00255718, 10886842. URL `http://www.jstor.org/stable/2008089`.

K. Falk. *Practical Recommender Systems*. Manning, 2019. ISBN 9781617292705. URL `https://www.manning.com/books/practical-recommender-systems`.

Massimiliano Ferronato and Giorgio Pini. A supernodal block factorized sparse approximate inverse for non-symmetric linear systems. *Numerical Algorithms*, 78(1):333–354, 2018. ISSN 1017-1398. doi: 10.1007/s11075-017-0378-x. URL `https://doi.org/10.1007/s11075-017-0378-x`.

Massimiliano Ferronato, Carlo Janna, and Giorgio Pini. A generalized Block FSAI preconditioner for nonsymmetric linear systems. *Journal of Computational and Applied Mathematics*, 256:230–241, 2014. ISSN 0377-0427. doi: 10.1016/j.cam.2013.07.049. URL `https://doi.org/10.1016/j.cam.2013.07.049`.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 12 2007. ISSN 1465-4644. doi: 10.1093/biostatistics/kxm045. URL `https://doi.org/10.1093/biostatistics/kxm045`.

Simon Funk. Netflix update: Try this at home, 2006. URL `https://sifter.org/~simon/journal/20061211.html`.

Cédric Févotte and Jérôme Idier. Algorithms for Nonnegative Matrix Factorization with the β-Divergence. *Neural Computation*, 23(9):2421–2456, 09 2011. ISSN 0899-7667. doi: 10.1162/NECO_a_00168. URL `https://doi.org/10.1162/NECO_a_00168`.

John R. Gilbert and Joseph W. H. Liu. Elimination structures for unsymmetric sparse \$lu\$ factors. *SIAM Journal on Matrix Analysis and Applications*, 14(2):334–352, 1993. doi: 10.1137/0614024. URL `https://doi.org/10.1137/0614024`.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore and London, fourth edition, 1996.

Marcus J. Grote and Thomas Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, 18(3):838–853, may 1997. ISSN

1064-8275. doi: 10.1137/S1064827594276552. URL https://doi.org/10.1137/S1064827594276552.

Per Christian Hansen. *Discrete Inverse Problems*. Society for Industrial and Applied Mathematics, 2010. doi: 10.1137/1.9780898718836. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898718836.

F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015. ISSN 2160-6455. doi: 10.1145/2827872. URL https://doi.org/10.1145/2827872.

Trevor J. Hastie, Rahul Mazumder, J. Lee, and Reza Bosagh Zadeh. Matrix completion and low-rank svd via fast alternating least squares. *Journal of machine learning research : JMLR*, 16:3367–3402, 2014.

Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation, 2020.

Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, page 263–272, USA, 2008. IEEE Computer Society. ISBN 9780769535029. doi: 10.1109/ICDM.2008.22. URL https://doi.org/10.1109/ICDM.2008.22.

T. Huckle. Approximate sparsity patterns for the inverse of a matrix and preconditioning. *Applied Numerical Mathematics. An IMACS Journal*, 30(2-3): 291–303, 1999.

Carlo Janna, Massimilano Ferronato, and Giuseppe Gambolati. A block FSAI-ILU parallel preconditioner for symmetric positive definite linear systems. *SIAM Journal on Scientific Computing*, 32(5):2468–2484, 2010. ISSN 1064-8275. doi: 10.1137/090779760. URL https://doi.org/10.1137/090779760.

Carlo Janna, Massimilano Ferronato, and Giuseppe Gambolati. Enhanced block FSAI preconditioning using domain decomposition techniques. *SIAM Journal on Scientific Computing*, 35(5):S229–S249, 2013. ISSN 1064-8275. doi: 10.1137/120880860. URL https://doi.org/10.1137/120880860.

Carlo Janna, Massimiliano Ferronato, and Giuseppe Gambolati. The use of supernodes in factored sparse approximate inverse preconditioning. *SIAM Journal on Scientific Computing*, 37(1):C72–C94, 2015a. ISSN 1064-8275. doi: 10.1137/140956026. URL https://doi.org/10.1137/140956026.

Carlo Janna, Massimiliano Ferronato, Flavio Sartoretto, and Giuseppe Gambolati. FSAIPACK: a software package for high-performance factored sparse approximate inverse preconditioning. *Association for Computing Machinery. Transactions on Mathematical Software*, 41(2):Art. 10, 1–26, 2015b. ISSN 0098-3500. doi: 10.1145/2629475. URL https://doi.org/10.1145/2629475.

Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Knowledge-based recommendation*, page 81–123. Cambridge University Press, 2010. doi: 10.1017/CBO9780511763113.006.

A. Jennings. A compact storage scheme for the solution of symmetric linear simultaneous equations. *Computing J.*, 9:281–285, 1966.

M. T. Jones and P. E. Plassmann. An improved incomplete Cholesky factorization. *ACM Transactions on Mathematical Software*, 21(1):5–17, 1995.

James M. Joyce. *Kullback-Leibler Divergence*, pages 720–722. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-04898-2. doi: 10.1007/978-3-642-04898-2_327. URL https://doi.org/10.1007/978-3-642-04898-2_327.

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998. doi: 10.1137/S1064827595287997. URL https://doi.org/10.1137/S1064827595287997.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditionings i. theory. *SIAM Journal on Matrix Analysis and Applications*, 14(1):45–58, 1993. doi: 10.1137/0614004. URL https://doi.org/10.1137/0614004.

L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditioning ii: Solution of 3d fe systems on massively parallel computers. *International Journal of High Speed Computing*, 07(02):191–215, 1995. doi: 10.1142/S0129053395000117. URL https://doi.org/10.1142/S0129053395000117.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, aug 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.263. URL https://doi.org/10.1109/MC.2009.263.

Solomon Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

Yann LeCun. *Modeles connexionnistes de l'apprentissage (connectionist learning models)*. PhD thesis, Universite P. et M. Curie (Paris 6), 1987.

Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 689–698, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356398. doi: 10.1145/3178876.3186150. URL https://doi.org/10.1145/3178876.3186150.

J. Liesen and Z. Strakoš. *Krylov Subspace Methods: Principles and Analysis*. Numerical Mathematics and Scientific Computation. OUP Oxford, 2012. ISBN 9780191630323.

Chih-Jen Lin and Jorge J. Moré. Incomplete cholesky factorizations with limited memory. *SIAM J. Sci. Comput.*, 21:24–45, 1999.

Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358, 1979. doi: 10.1137/0716027. URL `https://doi.org/10.1137/0716027`.

Joseph W. Liu. A compact row storage scheme for cholesky factors using elimination trees. *ACM Trans. Math. Softw.*, 12(2):127–148, jun 1986. ISSN 0098-3500. doi: 10.1145/6497.6499. URL `https://doi.org/10.1145/6497.6499`.

Joseph W. H. Liu. Reordering sparse matrices for parallel elimination. *Parallel Computing*, 11(1):73–91, 1989. ISSN 0167-8191. doi: https://doi.org/10.1016/0167-8191(89)90064-1. URL `https://www.sciencedirect.com/science/article/pii/0167819189900641`.

Kelong Mao, Jieming Zhu, Xi Xiao, Biao Lu, Zhaowei Wang, and Xiuqiang He. Ultragcn: Ultra simplification of graph convolutional networks for recommendation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, page 1253–1262, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384469. doi: 10.1145/3459637.3482291. URL `https://doi.org/10.1145/3459637.3482291`.

Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1018. URL `https://aclanthology.org/D19-1018`.

Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, ICDM '11, page 497–506, USA, 2011. IEEE Computer Society. ISBN 9780769544083. doi: 10.1109/ICDM.2011.134. URL `https://doi.org/10.1109/ICDM.2011.134`.

H. C. Pinkham. *Linear Algebra*. Springer, 2015. URL `https://www.math.columbia.edu/~pinkham/HCP_LinearAlgebra.pdf`.

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Conference on Computer Supported Cooperative Work*, 1994.

F. Ricci, L. Rokach, and B. Shapira. *Recommender Systems Handbook*. Springer US, 2022. ISBN 9781071621974. URL `https://link.springer.com/book/10.1007/978-1-0716-2197-4`.

Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computations. Technical Report 90–20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990.

Y. Saad. Preconditioned Krylov subspace methods for CFD applications. In W. G. Habashi, editor, *Solution Techniques for large-scale CFD Problems*, pages 141–157. J. Wiley and sons, 1995.

Y. Saad. *Iterative Methods for Sparse Linear Systems*. Computer Science Series. PWS Publishing Company, 1996. ISBN 9780534947767.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 285–295, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581133480. doi: 10.1145/371920.372071. URL https://doi.org/10.1145/371920.372071.

G. Schulz. Iterative Berechnung der reziproken Matrix. *ZAMM*, 13:57–59, 1933.

Jennifer Scott and Miroslav Tůma. *Algorithms for Sparse Linear Systems*. Springer, first edition, 2023. URL https://link.springer.com/book/10.1007/978-3-031-25820-6.

Ilya Shenbin, Anton Alekseev, Elena Tutubalina, Valentin Malykh, and Sergey I. Nikolenko. Recvae: A new variational autoencoder for top-n recommendations with implicit feedback. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, page 528–536, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368223. doi: 10.1145/3336191.3371831. URL https://doi.org/10.1145/3336191.3371831.

Barry F. Smith, Petter E. Bjorstad, and William D. Gropp. *Domain decomposition*. Cambridge University Press, Cambridge, UK, 1996. ISBN 0-521-49589-X. Parallel multilevel methods for elliptic partial differential equations.

Martin Spišák, Radek Bartyzal, Antonín Hoskovec, Ladislav Peška, and Miroslav Tůma. Scalable approximate nonsymmetric autoencoder for collaborative filtering. In *Proceedings of the 17th ACM Conference on Recommender Systems*, RecSys '23, New York, NY, USA, 2023. Association for Computing Machinery. To appear.

Harald Steck. Embarrassingly shallow autoencoders for sparse data. In *The World Wide Web Conference*, WWW '19, page 3251–3257, New York, NY, USA, 2019a. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313710. URL https://doi.org/10.1145/3308558.3313710.

Harald Steck. Markov random fields for collaborative filtering. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019b. Curran Associates Inc.

Harald Steck. Collaborative filtering via high-dimensional regression, 2019c.

Harald Steck and Dawen Liang. Negative interactions for improved collaborative filtering: Don't go deeper, go higher. In *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, page 34–43, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384582. doi: 10.1145/3460231.3474273. URL https://doi.org/10.1145/3460231.3474273.

A.C.N. van Duin and H. Wijshoff. Scalable parallel preconditioning with the sparse approximate inverse of triangular systems. Preprint, 1996.

Vojtěch Vančura, Rodrigo Alves, Petr Kasalický, and Pavel Kordík. Scalable linear shallow autoencoder for collaborative filtering. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 604–609, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3551482. URL https://doi.org/10.1145/3523227.3551482.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 165–174, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361729. doi: 10.1145/3331184.3331267. URL https://doi.org/10.1145/3331184.3331267.

Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, page 153–162, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450337168. doi: 10.1145/2835776.2835837. URL https://doi.org/10.1145/2835776.2835837.

Zygmunt Zajac. Goodbooks-10k: a new dataset for book recommendations. http://fastml.com/goodbooks-10k, 2017.

Junhai Zhai, Sufang Zhang, Junfen Chen, and Qiang He. Autoencoder and its various variants. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 415–419, 2018. doi: 10.1109/SMC.2018.00080.

List of Figures

1.1 Example of a user-item rating/interaction matrix. (Falk [2019]) . 10
1.2 An example bipartite graph $\mathcal{G}_X$ of user-item feedback. 11
1.3 Item-item relation graph $\mathcal{G}_\mathcal{I}$ obtained from $\mathcal{G}_X$. 11
1.4 User-based neighborhood-based filtering. (Falk [2019]) 13
1.5 Autoencoder architecture. (Bank et al. [2021]) 16

3.1 Aggregation through chains of users. 27

5.1 Architecture of sansa. (Spišák et al. [2023]) 49
5.2 Change of elimination tree with symmetric permutation. 51

6.1 Comparison of data splitting approaches. 61
6.2 Accuracy of sansa (cholmod) on MSD after various numbers of training scans s and finetune steps f. (Spišák et al. [2023]) 68
6.3 Comparison of time and memory usage of sansa (icf) versus mrf on Amazon Books. (Spišák et al. [2023]) 70

A.1 An illustration of a sparse matrix A with a symmetric sparsity pattern and its elimination tree $\mathcal{T}$. (Scott and Tůma [2023]) . . . 86

List of Tables

6.1 Attributes of training splits. 62
6.2 Recommendation accuracy on small, dense datasets. 65
6.3 Performance comparison on medium-sized, dense MSD dataset. . 67
6.4 State-of-the-art accuracy and scalability of SANSA (ICF). (Spišák et al. [2023]) 70

List of Abbreviations

RS recommender system

CF collaborative filtering

SVD singular value decomposition

TSVD truncated singular value decomposition

ALS Alternating Least Squares

NMF Non-negative Matrix Factorization

VAE variational autoencoder

COO COOrdinate format

CSR Compressed Sparse Row format

CSC Compressed Sparse Column format

SPD symmetric positive definite

ICF incomplete Cholesky factorization

MR Minimal Residual algorithm

AIB Approximate Inverse by Bordering

UMR Uniform Minimal Residual algorithm

FLOPs floating-point operations

SIMD single instruction multiple data

DCG discounted cumulative gain

IDCG ideal discounted cumulative gain

nDCG normalized discounted cumulative gain

ML-20M MovieLens 20M

MSD Million Song Dataset

A. Appendix

A.1 Elimination tree of sparse Cholesky factorization

A fundamental difference between dense and sparse Cholesky factorizations is that, in the latter, each column of the Cholesky factor L depends on only a subset of the previous columns. By uncovering these dependencies, the *symbolic phase* of the sparse Cholesky factorization is Thoroughout the section, $A \in \mathbb{R}^{n \times n}$ is an SPD matrix and L is its Cholesky factor.

A.1.1 Elimination tree

Let $\texttt{parent}(j)$ denote the row index of the first subdiagonal nonzero entry in the j-th column of L, i.e.,

$$\texttt{parent}(j) = \min\{i \mid i > j \text{ and } L_{i,j} \neq 0\}.$$

Denoting $\mathcal{I}$ the set of column indices of A, the function $\texttt{parent}$ can be used to define a directed acyclic graph

$$\mathcal{T} = (\mathcal{I}, \{(i, \texttt{parent}(i)) \mid i \in \mathcal{I}\}).$$

The graph $\mathcal{T}$ is called the **elimination tree** of A. Despite the terminology, the elimination tree need not be connected and generally is a forest. Let us assume that $\mathcal{T}$ is connected. Then, the last column of the factorized matrix does not have a parent and is the *root* of the elimination tree. Columns with no descendants (i.e., $i \in \mathcal{I}$ such that $\nexists j \in \mathcal{I} : i = \texttt{parent}(j)$) are the *leafs* of the elimination tree. An example of a matrix and its elimination tree is shown in Figure A.1. In the usual manner, directional arrows are omitted from the tree plot.

For details on the efficient construction of the elimination tree and its uses and implications, see Chapter 4 in the book by Scott and Tůma [2023].

A.1.2 Order of elimination and tree parallelism

The elimination tree captures the *sparse* elimination dependencies in the Cholesky factorization of A and, most crucially, determines the order in which the factorization must proceed. The i-th column can be processed only when all its descendants have been processed. On the other hand, disjoint column sequences - prefixes of paths from leaves of the elimination tree to the root with disjoint vertex sets - can be computed separately during the factorization process.

In this way, the elimination tree of A determines the possibilities for tree parallelism during factorization. The elimination tree should ideally be as ”wide” as possible.

A.1.3 Column replication principle

The **column replication principle** states that if $L_{i,j} \neq 0$ for $i > j$, then the subdiagonal part of the i-th column of L has nonzero entries in the positions of

$$
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{pmatrix}
* & & & & * & * & & \\
& * & & * & * & & & * \\
& & * & * & & & & * \\
& * & * & * & f & & & f \\
* & * & & f & * & f & & f \\
* & & & & f & * & & f \\
& & & & & & * & * \\
& * & * & f & f & f & * & *
\end{pmatrix}
\end{array}
$$

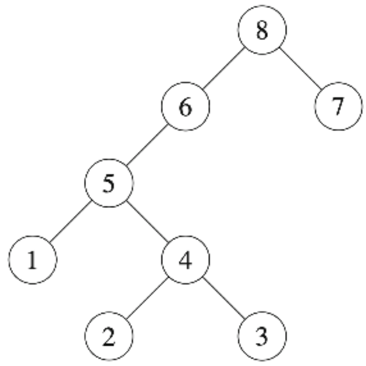


Figure A.1: An illustration of a sparse matrix A with a symmetric sparsity pattern and its elimination tree $\mathcal{T}$. The root vertex is 8. The filled entries in $\mathcal{S}(L + L^T)$ are denoted by f. (Scott and Tůma [2023])

nonzero entries in the j-th column of L (and possibly some other positions, too):

$$\mathcal{S}(L_{i:n,j}) \subseteq \mathcal{S}(L_{i:n,i}).$$

A.1.4 Equivalent condition for existence of a fill-in entry

Together with the column replication principle, the elimination tree reveals how fill-in propagates into the constructed factor. For a new fill-in entry to appear at a given position in the factor L, there needs to exist a nonzero entry in the same row of L in some previous column. Moreover, since $\texttt{parent}(j)$ denotes the first column of L to which the nonzero pattern of $L_{:,j}$ is replicated, column replication follows the paths from leaf vertices to the root in the elimination tree. Finally, it holds for leaf vertices i that $\mathcal{S}(L_{:,i}) = \mathcal{S}(A_{:,i})$. We have briefly summarized observations leading to the following theorem.

Theorem A.1 (Equivalent condition for existence of a fill-in entry (Liu [1986]))**.** *Let $A \in \mathbb{R}^{n\times n}$ be SPD, and let L be its Cholesky factor. If $A_{i,j} = 0$ for some $1 \le j < i \le n$, then a filled entry $L_{i,j} \neq 0$ exist if and only if there exist $k < j$ and $t \ge 1$ such that $A_{i,k} \neq 0$ and $j = \texttt{parent}^t(k)$.*[1]

Proof. See Liu [1986], Theorem 2.4. □

Therefore, any new fill-in entry in $L_{:,j}$ originates in some descendant of vertex j in the elimination tree. More precisely, the possible fill-in in $L_{i,j}$ originates in the *leaf* descendants of the i-th *row subtree* and is propagated toward the root until it reaches vertex j (see, e.g., Scott and Tůma [2023], Section 4.2). At that point, the entry $L_{i,j}$ is filled, and the fill-in propagates to further ascendants.

Observations stated here can be generalized for sparse LU factorization, although the non-symmetric case is more complicated. Refer to the book by Scott and Tůma [2023] and classical texts by Duff et al. [1988] and Gilbert and Liu [1993] for detailed explanation of the symbolic phase of sparse Cholesky (and LU) factorization and its relation to directed acyclic graphs.

[1] $\texttt{parent}^t$ denotes the composition of t parent mappings.

A.2 Supporting arguments for the choice of initial guess

A.2.1 When the factor to-be-inverted is sparse

The iteration step

$$X^{(1)} = 2I - A.$$

(see also Formula (5.1)) *exactly inverts* matrices $A = I + N$, where N is a strictly (lower, or upper) triangular matrix satisfying $N^2 = 0$, since

$$(I + N)(I - N) = I - N^2 = I$$

by the assumption. When $\widehat{L}$ is sparse, few subdiagonal entries of $\widehat{L}$ exist and are mostly small in magnitude. Hence, the subdiagonal part of $\widehat{L}$ (denoted N) satisfies $N^2 \approx 0$.

A.2.2 Column elimination matrices and elimination tree

The initial guess is also closely linked to column elimination matrices and their inverses. The process of Gaussian elimination of a matrix $A = LU$ can be expressed using the column elimination matrices $E_1, E_2 \ldots, E_n$[2] as

$$E_n E_{n-1} \cdots E_1 A = U,$$

see, e.g., Golub and Van Loan [1996]. The elimination process is captured in the columns of the lower triangular factor L, which at the same time satisfies

$$L = E_1^{-1} E_2^{-1} \cdots E_n^{-1}.$$

All elimination matrices E_k are unit lower triangular. Furthermore, all non-diagonal nonzeros reside in the subdiagonal part of the k-th column. It follows that $(E_k - I)^2 = 0$ and the iteration step (5.1) computes the exact inverse for $A = E_k$ (take $N = E_k - I$ in A.2.1).

Let us express the j-th column of L^{-1} using the elimination matrices:

$$\begin{aligned}
(L^{-1})_{:,j} &= (E_n E_{n-1} \cdots E_1)_{:,j} \\
&= (E_n E_{n-1} \cdots E_2)(E_1)_{:,j} \\
&= (E_n E_{n-1} \cdots E_2) I_{:,j} \\
&= (E_n E_{n-1} \cdots E_2)_{:,j} \\
&\vdots \\
&= (E_n E_{n-1} \cdots E_j)_{:,j} \\
&= (E_n E_{n-1} \cdots E_{j+1})(E_j)_{:,j} \qquad \text{(A.1)}
\end{aligned}$$

Examining the above expression, we see that

$$(L^{-1})_{:,n} = (E_n)_{:,n} = I_{:,n} = (2I - L)_{:,n}$$

[2] We set $E_n = I$ since no elimination is needed for the final column of A.

(since $E_n = I$), and

$$(L^{-1})_{:,n-1} = (E_n E_{n-1})_{:,n-1} = (E_{n-1})_{:,n-1} = (2I - E_{n-1}^{-1})_{:,n-1} = (2I - L)_{:,n-1},$$

where we used the fact that $L_{:,j} = (E_j^{-1})_{:,j}$ for all $j \in \{1, \ldots, n\}$. In other words, the iteration step (5.1) correctly computes the final two columns of L^{-1}. As explained below, if A is sparse, even more columns can be inverted correctly.

The expression (A.1) reveals that values in the j-th column of L^{-1} are based on the values in the j-th column of the elimination matrix E_j. However, later positions in the j-th column recursively depend on prior positions since, for instance, the l-th row of $E_{j+1}E_j$ is the linear combination of rows of E_j with coefficients in the l-th row of E_{j+1}. This linear combination always contains the l-th row of E_j with coefficient 1, and if $l > j+1$ and $(E_{j+1})_{l,j+1} \neq 0$, then the linear combination also includes the $(j+1)$-st row of E_j with a nonzero coefficient $(E_{j+1})_{l,j+1}$. In other words, the multiplication by E_{j+1} modifies the positions $j+2, \ldots n$ in $(E_j)_{:,j}$ by adding a $(E_j)_{j+1,j}$-multiple of $(E_{j+1})_{j+2:,j+1}$, written formally,

$$(E_{j+1}E_j)_{:,j} = (E_j)_{:,j} + (E_j)_{j+1,j} \cdot \Big((E_{j+1})_{:,j+1} - I_{:,j+1}\Big). \tag{A.2}$$

The multiplication by E_{j+2} modifies the positions $j+3, \ldots n$ in $(E_{j+1}E_j)_{:,j}$ and so on.

When $(E_j)_{j+1,j} = 0$, Equation (A.2) yields

$$(E_{j+1}E_j)_{:,j} = (E_j)_{:,j}. \tag{A.3}$$

Moreover, similarly to (A.2) we may express

$$(E_{j+2}(E_{j+1}E_j))_{:,j} = (E_{j+1}E_j)_{:,j} + (E_{j+1}E_j)_{j+2,j} \cdot \Big((E_{j+2})_{:,j+2} - I_{:,j+2}\Big),$$

and if $(E_j)_{j+1,j} = 0$, Equation (A.3) gives us

$$(E_{j+2}E_j)_{:,j} = (E_{j+2}(E_{j+1}E_j))_{:,j} = (E_j)_{:,j} + (E_j)_{j+2,j} \cdot \Big((E_{j+2})_{:,j+2} - I_{:,j+2}\Big).$$

By induction, it, therefore, follows that

$$\text{if } (E_j)_{i,j} = 0 \text{ for all } i = \{j+1, \ldots, k\},\; j \leq k < n \text{ and } (E_j)_{k+1,j} \neq 0, \tag{A.4}$$

then

$$(E_{k+1}E_k \cdots E_{j+1}E_j)_{:,j} = (E_{k+1}E_j)_{:,j}. \tag{A.5}$$

Consequently, if A is sparse, its elimination matrices do not affect all subsequent columns. In the simple case when A is SPD, the condition (A.4) means that $k+1$ is the parent of j in the elimination tree ($\texttt{parent}(j) = k+1$; see Appendix A.1). If $k > j$ is not an ancestor of j, sparsity patterns of $(E_k)_{k:,k}$ and $(E_j)_{k,j}$ do not overlap, therefore hence

$$(E_k E_j)_{:,j} = (E_j)_{:,j}.^3 \tag{A.6}$$

[3] The equality can be proven analogously to (A.2)

Finally, denoting $\texttt{parent}(j)$, $\texttt{parent}^2(j)$,... $\texttt{parent}^t(j)$ all ancestors of j, the telescopic property (A.5) together with (A.6) gives us

$$\begin{aligned}(L^{-1})_{:,j} &= (E_n E_{n-1} \cdots E_1)_{:,j} \\ &= (E_n E_{n-1} \cdots E_j)_{:,j} \\ &= (E_{\texttt{parent}^t(j)} E_{\texttt{parent}^{t-1}(j)} \cdots E_{\texttt{parent}(j)} E_j)_{:,j}\,,\end{aligned} \tag{A.7}$$

In particular, it follows from (A.7) that if $\texttt{parent}(j) = n$, then

$$(L^{-1})_{:,j} = (E_n E_j)_{:,j} = (E_j)_{:,j} = (2I - E_j^{-1})_{:,j} = (2I - L)_{:,j}\,, \tag{A.8}$$

i.e., *every direct descendant of the root column is inverted correctly by the iteration step* (5.1). Indirect descendants j of the root column are only affected by their ancestors. The affected positions of $(E_j)_{:,j}$ are exactly the union of subdiagonal nonzero entries of the ancestors, which can be expressed as

$$\bigcup_{i=1}^{t} \left(\{\texttt{parent}^i(j) + 1, \dots n\} \cup \mathcal{S}\Big((E_{\texttt{parent}^i(j)})_{:,\texttt{parent}^i(j)}\Big) \right).$$

To conclude, if the elimination tree of A is wide and shallow, many of the columns of L^{-1} are inverted correctly or with only a few incorrect positions. This assumption on the structure of the elimination tree is likely satisfied when the factor L is very sparse. Additionally, a suitable choice of *reordering* should help; some reordering algorithms aim at constructing the elimination tree as wide as possible (refer to, e.g., the article by Liu [1989]).

A.3 Repository

The repository is available at `https://github.com/matospiso/sansa`. It contains all model codes, codes of the experiment pipeline (dataset preprocessing, splitting, running model training, and evaluation), and experiment scripts. It also includes all experimental results and complete logs for each experiment. The `README` file also includes instructions on how to run the experiments to replicate our results.

A.3.1 File organization

The root directory contains five folders:

1. `datasets`: Contains Python modules for loading and preprocessing individual datasets (`amazonbook.py`, `goodbooks10.py`, `movielens20.py`, `msd.py`, `netflix.py`), the abstract base class for all datasets (`dataset.py`), and a module for creating dataset splits (`split.py`). Additionally, dataset files should be stored inside `datasets/data`; see Section A.3.2.

2. `evaluation`: Includes logging (`logs.py`), metrics definitions (`metrics.py`), and evaluation functions (`evaluate.py`). The experiment pipeline is defined in `pipeline.py`, with pipeline steps (used in logging) defined in `steps.py`.

3. `experiments`: Contains subfolders for three experiments: `accuracy` – test recommendation accuracy (and measure training time), `memory` – test memory requirements, and `shorter_training` – test accuracy of various early checkpoints of SANSA, and one mock experiment: `sandbox`.

   Each experiment folder contains subfolders for all datasets on which we ran the experiment, and inside each dataset's subfolder are experiment scripts named `run_experiment{_optional_specifiers}.py`. Moreover, the subfolders contain information about the AWS instance used to conduct the experiment, complete console logs, and experiment results stored as `json` in the `results` subfolder. Lastly, the datasets' subfolders contain Jupyter notebooks for result inspection.

4. `models`: Includes the abstract base class for all models (`model.py`) and implementations of EASE[R] (`ease.py`), MRF (`mrf.py`), and SANSA (`sansa.py`).

5. `sparseinv`: The implementations of mathematical computations used for training. The LDL^T decomposition of $P(X^T X + \lambda I)P^T$ using ICF and CHOLMOD (Chen et al. [2008]) is defined `ldlt.py`, a separate implementation of ICF is in `icf.py`. The implementation of UMR and the approximate inversion function is in `ainv.py`. This folder also includes utility functions necessary for efficient implementation (in `utils.py`) and a hotfix enabling multi-threaded sparse matrix multiplication (unsupported in vanilla `SciPy`) in `matmat.py`.

A.3.2 Setup

The setup steps necessary to reproduce the experiment results are 1. downloading the datasets and 2. setting up a virtual environment with necessary packages.

Datasets

Five datasets are available for experiments:

1. `goodbooks10`: Goodbooks-10k dataset[4].
2. `movielens20`: MovieLens 20M dataset [5].
3. `msd`: Million Song Dataset [6].
4. `netflix`: Netflix Prize dataset [7].
5. `amazonbook`: Amazon Books dataset [8].

The dataset files should be located inside `datasets/data/{dataset_name}`.

Setting up a virtual environment using Conda

Below we provide instructions on how to install necessary packages inside a virtual environment using Conda[9]. Updating the Conda installation before installation is recommended:

```
conda update -n base -c conda-forge conda
```

There are two possible ways to set up the virtual environment:

1. Recommended Intel optimized (but also works on AMD).

```
conda create -n sansa python==3.10.9
conda activate sansa

conda install -c intel numpy==1.22.3 scipy==1.7.3
conda install -c conda-forge suitesparse==5.10.1 \
    scikit-sparse==0.4.8

pip install sparse-dot-mkl==0.8.3 black==23.3.0 numba==0.57.0 \
    memory-profiler==0.61.0 pandas==2.0.1 scikit-learn==1.2.2 \
    fastparquet==2023.4.0 matplotlib==3.7.1 jupyter==1.0.0
```

[4] https://github.com/zygmuntz/goodbooks-10k
[5] https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset
[6] https://www.kaggle.com/competitions/msdchallenge/data
[7] https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data
[8] https://github.com/kuandeng/LightGCN/tree/master/Data/amazon-book
[9] https://docs.conda.io/en/latest/

2. Compatibility mode Works on Apple Silicon. Works when MKL[10] is not available.

```
conda create -n sansa-nomkl python==3.10.9
conda activate sansa-nomkl

conda install -c conda-forge suitesparse==5.10.1 \
    scikit-sparse==0.4.8

pip install numpy==1.22.3 scipy==1.7.3 black==23.3.0 \
    numba==0.57.0 memory-profiler==0.61.0 pandas==2.0.1 \
    scikit-learn==1.2.2 fastparquet==2023.4.0 matplotlib==3.7.1 \
    jupyter==1.0.0
```

A.3.3 Reproducing the results

1. Download the datasets and store them in the `datasets/data` folder.

2. Set up a virtual environment using the instructions above.

3. Inside the virtual environment, run experiments from the **root** directory:

```
python experiments/{experiment_name}/{dataset_name}/\
    run_experiment{_optional_specifiers}.py
```

The experiment results are stored in `json` files inside

`experiments/{experiment_name}/{dataset_name}/results`

Each results file also contains information about the dataset and model config used in the experiment. The results can be inspected in Jupyter notebooks:

`experiments/{experiment_name}/{dataset_name}/results_summary.ipynb`

[10] `https://www.intel.com/content/www/us/en/docs/onemkl/get-started-guide/2023-0/overview.html`