MetaRec: Meta-Learning Meets Recommendation Systems

by

James Le

Submitted to the
B. Thomas Golisano College of Computing and Information Sciences
Department of Computer Science
in partial fulfillment of the requirements for the
Master of Science Degree
at the Rochester Institute of Technology

Abstract

Artificial neural networks (ANNs) have recently received increasing attention as powerful modeling tools to improve the performance of recommendation systems. Meta-learning, on the other hand, is a paradigm that has re-surged in popularity within the broader machine learning community over the past several years. In this thesis, we will explore the intersection of these two domains and work on developing methods for integrating meta-learning to design more accurate and flexible recommendation systems.

In the present work, we propose a meta-learning framework for the design of collaborative filtering methods in recommendation systems, drawing from ideas, models, and solutions from modern approaches in both the meta-learning and recommendation system literature, applying them to recommendation tasks to obtain improved generalization performance.

Our proposed framework, *MetaRec*, includes and unifies the main state-of-the-art models in recommendation systems, extending them to be flexibly configured and efficiently operate with limited data. We empirically test the architectures created under our MetaRec framework on several recommendation benchmark datasets using a plethora of evaluation metrics and find that by taking a meta-learning approach to the collaborative filtering problem, we observe notable gains in predictive performance.

Acknowledgments

A Master's Thesis track is a long journey through unknown territories, soaring peaks, and deep as well as dark troughs. I am thankful for the many people I got to meet and who accompanied me along the way. First, I'd like to thank my thesis advisor Alexander Ororbia II. The freedom you gave me to pursue my interests and follow my curiosity has made all the difference. Thank you for your support and for fostering a productive research environment within the Neural Adaptive Computing (NAC) lab, even among uncertain times and virtual working conditions this past year.

I am grateful for the support from my family and friends at home in Vietnam and here in the United States. Special thanks to my dad Nam, my mom Hang, and my uncle Duc.

I am very fortunate to have met mentors along the way who took a chance on me and helped me grow. Special thanks goes to Richard Zanibbi and Christopher Kanan – the classes that I took taught by them enabled me to cultivate valuable research skills. Also big thanks to past and present colleagues at ZestAI, Limbik, Full-Stack Deep Learning, and SnorkelAI – where I learned how machine learning works and is applied in the real world.

I also want to thank my labmates in the NAC lab. In particular, I am greatly appreciative of the discussions I had with Michael Peechatt, Hitesh Vidaiya, and Ankur Mali – my collaboration with them on the meta-learning survey planted the seed for several significant ideas in this thesis.

I am grateful for all the amazing people that I got the chance to know at numerous in-person and virtual events. I am looking forward to seeing you all at future ones.

I would like to thank Alex Ororbia, Richard Zanibbi, and Christopher Homan for feedback on drafts of this thesis. Lastly, I would like to express my gratitude to the Department of Computer Science at Rochester Institute of Technology for a wonderful two and a half year long journey of academic pursuit.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

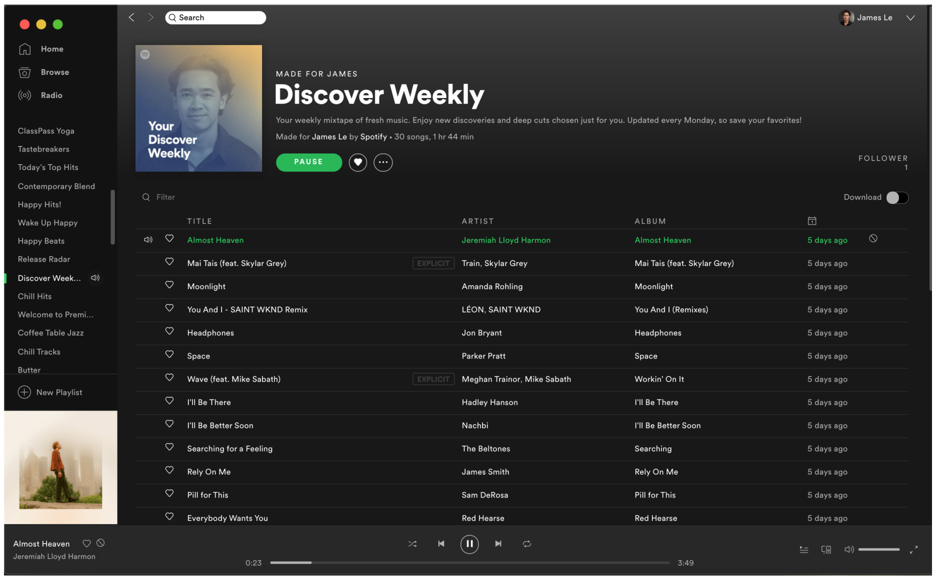
1.1 Overview

With the explosive growth of digital information over the past decade, consumers often face the dilemma of choosing among the abundance of choices. As a result, automatic recommendations are essential for facilitating a better user experience and reducing information overload. Overall, recommendation systems have played an indispensable role in various information filtering and data mining systems to boost business value and facilitate decision-making processes. There has been a great deal of work done in recent years, in both industrial and academic settings, to craft newer, more robust approaches to recommendation systems.

There are various techniques to design these systems, ranging from the simple (e.g., based only on other rated items from the same user) to the extremely complex. Complex recommendation systems leverage a variety of different data sources and often utilize non-linear learning methods. Thus, the recommendation task provides an excellent problem space to apply machine learning methodology. As users continue to consume content and yield more data, we can build machine learning-based systems to exploit this data to provide better and better recommendations. This thesis's research goal is to develop learning models and methods that can effectively offer recommendations to users, tailoring to explicit and implicit preferences that can generalize to other, previously unseen users.

1.1.1 Functions

There are many reasons why recommendation systems would be deployed in the business world. We highlight four essential functions:



MADE FOR JAMES

Discover Weekly

Your weekly mixtape of fresh music. Enjoy new discoveries and deep cuts chosen just for you. Updated every Monday, so save your favorites!

Made for James Le by Spotify • 30 songs, 1 hr 44 min

TITLE		ARTIST	ALBUM	
Almost Heaven		Jeremiah Lloyd Harmon	Almost Heaven	5 days ago
Mai Tais (feat. Skylar Grey)	EXPLICIT	Train, Skylar Grey	Mai Tais (feat. Skylar Grey)	5 days ago
Moonlight		Amanda Rohling	Moonlight	5 days ago
You And I - SAINT WKND Remix		LÉON, SAINT WKND	You And I (Remixes)	5 days ago
Headphones		Jon Bryant	Headphones	5 days ago
Space		Parker Pratt	Space	5 days ago
Wave (feat. Mike Sabath)	EXPLICIT	Meghan Trainor, Mike Sabath	Workin' On It	5 days ago
I'll Be There		Hadley Hanson	I'll Be There	5 days ago
I'll Be Better Soon		Nachbi	I'll Be Better Soon	5 days ago
Searching for a Feeling		The Beltones	Searching	5 days ago
Rely On Me		James Smith	Rely On Me	5 days ago
Pill for This		Sam DeRosa	Pill for This	5 days ago
Everybody Wants You		Red Hearse	Red Hearse	5 days ago

Almost Heaven
Jeremiah Lloyd Harmon
0:23 3:49

Figure 1.1: Example of Spotify Discover Weekly.

- *To increase the number of items sold*: In general, from a business perspective, the primary goal for using recommendation systems is to increase the conversion rate - which is the number of users that accept the recommendation and consume an item, compared to the number of visitors that browse through the information.

- *To sell more diverse items*: In a music streaming service such as Spotify, the company is interested in displaying all of the music from all types of artists, not just the popular ones. This would be difficult to do without a recommendation system, given that Spotify cannot afford the risk of advertising music that is not likely to suit a particular user's taste. Therefore, a recommendation system can suggest unpopular music to the right users (Figure 1.1).

- *To increase user satisfaction*: A well-designed recommendation system can improve the user experience within the application. The user will find the recommendations to be interesting, relevant, and enjoyable. This leads to higher (overall) satisfaction.

- *To better capture user's needs*: A good recommendation system describes user preferences in detail, either explicitly or implicitly. The business may then decide to reuse this knowledge for many other goals.

1.1.2 Applications

Recommendation systems research is generally conducted with a strong emphasis on practical and commercial applications. The application domain significantly impacts the type of algorithmic approach that should be taken. [100] provides a taxonomy of recommendation systems and classifies their existing applications to specific domains:

- *Entertainment* - recommendations for movies (Netflix), music (Spotify, Pandora), and mobile apps (Apple Store, Android Store).
- *Content* - personalized newspapers (The New York Times, The Wall Street Journal), recommendation for images (Pinterest), recommendations of Web pages (Pocket), e-learning applications (Coursera), and e-mail filters (Gmail).
- *E-commerce* - recommendations for consumers of products to buy such as books (Amazon), beauty supplies (Birchbox), and clothes (Stitch Fix).
- *Services* - recommendations of travel services (Skyscanner), experts for consultation (StyleSeat, ClassPass), houses to rent (Zillow), or matchmaking services (Tinder, Bumble).

1.1.3 Current Challenges

While the task of automatic recommendation is not new, there remain many challenges that drive research in modern-day systems, especially for collaborative filtering ones that aim to find user-item interactions to facilitate recommendations. The three of interest to this thesis are:

- **Scalability**: Real-world recommendation systems are deployed in a dynamic, interactive environment - user and item data come in and out rapidly. It is crucial to address the computing requirements [131] and training/inference time [87] required to efficiently process this data.
- **Sparsity**: The sparsity problem is quite prominent in collaborative filtering systems since only a small number of users provide the ratings, and only a small number of items have the ratings. This is also known as the "cold-start problem" [17, 141], which prevents recommendation systems from generating meaningful predictions due to limited data.
- **Accuracy**: Recommendation systems need to provide a high prediction accuracy level to ensure the quality that users often demand. In research,

accuracy is usually evaluated/investigated by rating prediction and item ranking [131].

In this thesis, we will address the above challenges by reframing the problem of recommendation system training. Solutions to this problem typically follow a supervised learning format that requires extensive guidance from the system designer in providing necessary target behavior. In contrast, we argue that adopting an unconventional meta-learning ("learning how to learn") approach can be far more fruitful.

Specifically, we pay attention to the accuracy challenge, as it is the most well-defined one in existing recommendation systems literature. To this end, we develop a novel framework that can meta-learn across preferences for different users for a variety of base (collaborative filtering) models. We demonstrate that our models using this framework outperform models that do not possess meta-learning capability across several accuracy metrics.

1.2 Research Goals

This thesis studies the problem of learning (distributed) representations that can learn fast and efficiently with few training samples using both linear and neural-based models crafted for the recommendation task, specifically focusing on collaborative filtering. The central hypothesis of this thesis is the following:

Collaborative filtering methods that meta-learn information from related tasks generalize better than methods that do not use this information across a wide range of tasks in terms of accuracy.

As a result, the research goals of this work are two-fold. First, we aim to identify and analyze the current trends in several sub-domains relevant to this thesis. Specifically, we will consider the following:

- **Meta-Learning**: we will study the different families under which meta-learning algorithms are categorized. More specifically, we will provide a mathematical formulation of each and include examples of relevant different architectures. We will also identify existing challenges with each family.
- **Matrix Factorization for Recommendation**: we will thoroughly cover the inner workings of matrix factorization, the canonical and popular linear model for collaborative filtering. We will also examine ways to strengthen its capacity.

- **Multi-Layer Perceptron for Recommendation**: we dive deep into various state-of-the-art recommendation architectures that utilize a multi-layer perceptron as their backbone.
- **Autoencoders for Recommendation**: we will explore the various advances in the formalization of the autoencoder (and its variants) for the task of recommendation.

Second, this thesis will provide the initial steps towards unifying views and ideas in both neural-based automatic recommendation and meta-learning methodology, identifying the essential elements and principles upon which a deep meta-learning framework for recommendation systems can be built. Concretely, we present two original contributions:

- The adaptation of optimization-based meta-learning to the construction of recommendation systems by bridging the principles proposed in meta-learning with key concepts involved in collaborative filtering.
- The design of a neural-based, meta-learning framework for recommendation systems that provides a flexible configuration taking into account both linear and non-linear models.

1.3 Thesis Outline

The rest of the thesis is organized as follows:

- Chapter 2 provides an overview of background information that is relevant to understand the contents of this document. We formally define the recommendation task (Section 2.1), including content-based filtering and collaborative filtering, and discuss artificial neural network-based methods and their use-cases for the task (Section 2.2).
- Chapter 3 provides a comprehensive study of meta-learning. Three broad meta-learning paradigms are examined: black-box meta-learning (Section 3.3), optimization-based meta-learning (Section 3.4), and non-parametric meta-learning (Section 3.5).
- Chapter 4 presents an extensive look at state-of-the-art recommendation systems. Four general areas are identified: matrix-factorization-based systems (Section 4.1), multi-layer-perceptron-based systems (Section 4.2), autoencoder-based systems (Section 4.3), and meta-learning-based systems (Section 4.4).

- Chapter 5 details our proposed MetaRec framework, which is based on an optimization-based meta-learning algorithm and allows for different choices of base recommendation models.
- Chapter 6 presents the results of our evaluation regarding MetaRec's effectiveness. Three sets of experiments were conducted using different base models: a matrix-factorization variant (Section 6.2), a multi-layer-perceptron variant (Section 6.3), and an autoencoder variant (Section 6.4).
- Chapter 7 summarizes the work and contributions, discusses possible improvements, and outlines potential future work and directions.

Chapter 2

Recommendation Systems

2.1 Problem Formulation

In recommendation systems, an item's utility is usually represented by a rating (see Figure 2.1), which indicates how much a particular user liked a specific item. In general, utility can be an arbitrary function and depends on the application's goal. For example, an item will be more useful if it increases user satisfaction or better captures a user's needs. Depending on the application, the utility F can either be specified by the user (as is often done in the context of user-defined ratings) or is computed by the application (as is the case for a profit-based utility function).

Bearing utility in mind, formally, the recommendation problem can be formulated in the following manner:

- Let U be the set of all users and let I be the set of all items. Both of these spaces can be very large - up to potentially millions of items, depending on consumer service being provided.

- Let F be the utility function that measures the relevance of item i to user u as follows: $f : U \times I \rightarrow R$, where R is an ordered set of user preferences for the items.

- For each user $u \in U$, we want to choose an item $i \in I$ that maximizes the user's utility.

Mathematically, given the above notation and setting, we are interested in solving the following optimization problem:

$$\forall u \in U, i_s = \arg\max_{i \in I} F(u, i) \tag{2.1}$$

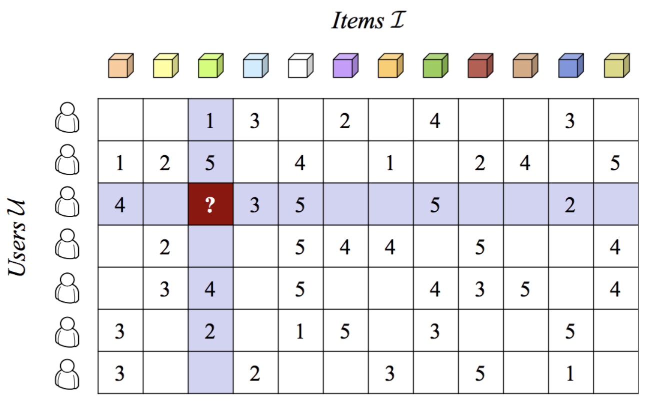


Figure 2.1: The recommendation problem as a rating prediction task [154].

where each element of the user space U can be defined with a profile that includes various user characteristics, such as user ID, age, gender, income etc. Similarly, each element of the item space I is defined with a (particular) set of characteristics. For example, in a music recommendation application, e.g., Spotify, Soundcloud, Pandora, where I is a collection of songs, each song can be represented not only by its ID but also by its title, genre, artist, year of release, etc. Alternatively, in a travel recommendation application, e.g., Expedia, Booking, Airbnb, where I is a collection of travel options, each option can be represented by its location, price, local activities, etc.

The central problem faced by any recommendation system is that the utility F is usually not defined over the whole $U \times I$ space – it is, at best, only defined on a subset of this space. This means that F needs to be extrapolated to the whole space $U \times I$. In recommendation systems, the utility is typically represented by (user) ratings and is initially defined only on the items that have been previously rated by (existing) users. For example, in a book recommendation application like Goodreads, users initially rate some subsets of books that they have already read. The application's goal is to predict the ratings of the non-rated book-user combinations and display the appropriate recommendations based on these predictions.

There are two ways to extrapolate from known to unknown ratings. The first specifies heuristics that define the utility function and empirically validates the function's performance. The second estimates the utility function that optimizes specific performance criteria using a proxy function such as mean squared error or cross-entropy loss.

Once we are able to estimate the unknown ratings, we can make the actual

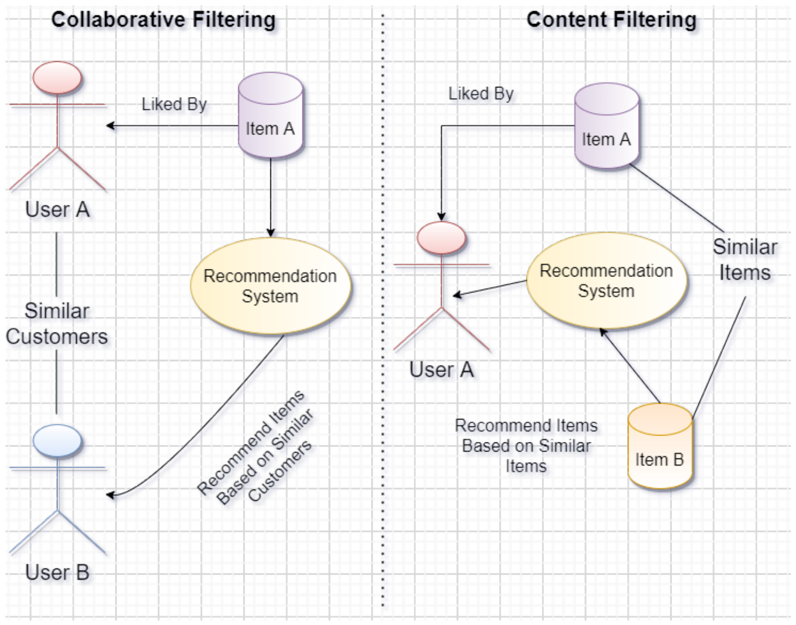


Figure 2.2: Collaborative filtering versus Content-Based filtering.

recommendations of an item to a user by choosing the highest rating among all the estimated ratings for that particular user. Otherwise, we can also recommend the K best items to a user or a set of K users to an item.

Generally speaking, recommendation systems are categorized into one of three categories [6] (the first two of which are depicted in Figure 2.2):

- **Content-Based Filtering**: The system recommends new items with similar characteristics to existing items that the user has preferred in the past. Hence, the quality of the recommendations depends entirely on the quality of the item content.

- **Collaborative Filtering**: The system recommend items to new users with similar tastes to existing users in the database. Hence, the quality of the recommendations depends on the quality of the user preferences.

- **Hybrid Approach**: A hybrid system simply combines both content-based and collaborative filtering to generate recommendations.

2.1.1 Content-Based Filtering

In content-based recommendation systems, the utility function $F(u, i)$ of item i for user u is estimated based on the utilities $F(u, i_k)$ assigned by user u for each item $i_k \in I$ that is similar to item i. For instance, in a music recommendation engine, in order to recommend new songs to user u, a content-based recommendation approach tries to understand the similarities among the songs that user u has listened to frequently in the past. Then, only the songs that have a high degree of similarity to whatever the user's preferences are would be subsequently recommended.

While useful, as was observed in [6] and [132], content-based approaches exhibit several important limitations:

- *Limited Analysis*: Content-based approaches are limited by the features that are explicitly associated with the items recommended. Hence, to have a sufficient set of features, the content must either be: 1) in a format that can be processed automatically, or 2) can be assigned to items manually. In the first scenario, it isn't easy to automatically extract features in unstructured data such as images and audio. In the second scenario, it is often impractical to assign attributes by hand because of limited computational and human resources.

- *Homogeneity*: When the system can only recommend items that score highly against a user's profile, the user is restricted to only items similar to those already rated. In other words, content-based approaches fail to provide diverse recommendations. Ideally, the user should be presented with a wide range of options and not just a homogeneous set of alternatives.

- *New User Recommendations*: A user has to rate a sufficient number of items before a content-based recommendation system can understand his/her preferences and present him/her with trustworthy items. Thus, a new user with no prior ratings will not get accurate recommendations.

2.1.2 Collaborative Filtering

In collaborative filtering recommendation systems, we attempt to predict the utility of items for a particular user based on the items previously rated by other users. More formally, the utility function $F(u, i)$ of item i for user u is estimated based on the utilities $F(u_j, i)$ assigned to item i by those users $u_j \in U$ who are similar to user u.

For instance, in the context of a book recommendation application, in order to recommend books to user u, the collaborative filtering system first finds the "peers" of user u, or other users with similar tastes in books (who rate the same books similarly). Then, only the books that are most liked by the peers of user u would be recommended.

According to [19], algorithms for collaborative filtering approaches can be divided into two main classes: *memory-based* and *model-based*. Memory-based algorithms are heuristics that predict ratings based on the entire collection of previously rated items by the system's users. In contrast to memory-based methods, model-based algorithms use a collection of ratings to learn a model from the underlying data using statistical and machine learning techniques, which, after fit to data, are then used to predict ratings. A method that combines both approaches was proposed in [11], where it was empirically demonstrated that the use of this mixed approach provided better recommendations than either pure memory-based and pure model-based techniques.

Pure collaborative recommendation systems also have particular limitations:

- *New User Recommendations*: This is the same problem as with content-based approaches. To make accurate recommendations, the system must first learn the user's preferences from the previous ratings.

- *New Item Recommendations*: New items are regularly put into the system. Since collaborative filtering approaches rely solely on users' preferences to make recommendations, a new item will not be recommended until it is rated by a sufficient number of users.

- *Sparsity*: In any system, the number of ratings already obtained is usually small compared to the number of ratings that need to be predicted. Effective prediction of ratings from a small number of examples is quite essential. Additionally, the success of collaborative filtering depends on the availability of a critical mass of users. For instance, there may be many products purchased by only a few people in a product recommendation system, which means that these would be recommended very rarely, even if those few users gave these items high ratings. Furthermore, for the user whose tastes are unusual compared to the rest of the population, there will not be any other significantly similar users, yielding poor recommendations [6].

2.1.3 Hybrid Approaches

A hybrid system combines content-based and collaborative filtering methods, side-stepping the limitations inherent to each. Among the many ways one could design a hybrid system, we highlight the following methods:

- [28, 107] implement collaborative filtering and content-based filtering separately, then combine their (final) predictions. The final combination can happen via a linear combination or a voting function.
- [6, 42, 96, 107] keep track of the item content during the design of a collaborative filtering system. This helps solve the cold-start issue, the biggest bottleneck of collaborative filtering.
- [137] reduces the dimension of the user data matrix during the design of a content-based filtering system. This helps solve the scalability issue, the biggest bottleneck of content-based filtering.
- [3,7,21,110,128] unify the characteristics of both content-based and collaborative filtering through rule-based, probabilistic-based, and knowledge-based techniques/methods.

2.2 Deep Learning Approaches

Recently, deep learning, or machine learning based on artificial neural networks (ANNs) that contain many layers of nonlinear processing, has seen increased adoption as a promising solution to a variety of problems in multiple computer science sub-fields (e.g., computer vision [72], language modeling [103], speech recognition [44], and data mining). In addition to the impressive results achieved by ANNs on several benchmarks, deep learning solutions have helped to form a common machine learning foundation across several disciplines as well as a shared vocabulary that bridge these fields together. Concerning the recommendation task, they have overcome conventional models' obstacles and achieved high recommendation quality.

2.2.1 Artificial Neural Network Architectures

While the survey [169] does a good job of summarizing the most prominent ANN architectures that are relevant in the context of recommendation systems, we provide a brief summary/explanation of the methods directly related to this thesis:

- A *multi-layer perceptron* (MLP) is an ANN with multiple hidden layers. It can be trained with back-propagation of errors [25,121] (or backprop), a well-known algorithm for calculating the gradients of a cost function with respect to model parameters. One heuristic that has emerged from recent work in deep learning is that adding more hidden layers combined with more efficient parameter initialization methods can lead to better model out-of-sample performance (generalization). This type of ANN will be discussed further in section 4.2.

- An *autoencoder* [60] is an unsupervised ANN, where the input layer and the output layer are the same – the bottleneck (hidden) layer of an autoencoder learns to extract a low-dimensional representation of the input data. This type of network will also be discussed further in section 4.3.

- A *Boltzmann machine* [57] is a neural network with symmetrically connected layers that works best with binary vectors. Recently, a particular design of a Boltzmann machine, also known as the restricted Boltzmann machine (RBM) [59], includes hidden units and visible units that are not (laterally) connected to each other yielding a bi-partite graph that facilitates efficient inference. Multiple RBMs can be stacked and trained greedily to form a deep belief network (DBN) [58] to learn complex, increasingly abstract representations of input data.

- A *convolutional neural network* (CNN) [79] is a specialized type of ANN that is meant to handle data with a spatial topology, such as images and videos. A typical CNN consists of multiple, stacked convolutional layers (which convolve their input with a set of filters), pooling layers (which decrease the size of their inputs with a fixed down-sampling transformation), and fully-connected layers (which compute the logits of different classes before a softmax transformation is applied to get predictive class probabilities).

- A *recurrent neural network* (RNN) [143] is another specific type of neural network that handles temporal data. A typical RNN utilizes a recurrence formula to process a sequence of vectors at the input, at the output, or either serially/in parallel.

2.2.2 Solutions For Existing Challenges

The first well-known study that uses deep learning in the context of recommendation systems dated back to 2007, which employed RBMs for rating prediction [125]. 2015 marked the signs of the deep learning "boom" for recommendations, where a couple of seminal papers laid the groundwork for current research directions. 2016 saw a steep increase: the creation of the "Deep Learning For RecSys" workshop series, the massive increase in deep learning papers at conferences such as RecSys, KDD, SIGIR, and the forming of distinct research directions in recommendation research by the end of the year (e.g., deep collaborative filtering, feature extraction from content, learning item embeddings, session-based recommendations, etc.). Since 2017, we have seen continued, promising progress made along these research directions, alongside more advanced ideas from deep learning that have yet to be applied.

We next review several relevant studies to see how they tackle the challenges (of Section 1.1.3) for the recommendation task.

Handling Scalability: In order to deal with the challenge of scalability, *ANN models have been used to extract low-dimensional vectors from high-dimensional user preferences for known items.* Truyen et al. [150] used RBMs to engineer latent features of users and items to generate final (recommendation) predictions. Elkahky et al. [34] experimented with different dimensionality reduction techniques applied to raw input features, which included using an MLP/feedforward network to reduce the feature dimensions. They finally selected only the most relevant features (in the reduced space) and finally applied a clustering procedure to group similar features together.

Another way to tackle scalability is *to look at it from a systems perspective.* Interestingly, Louppe [89] used parallel computing to speed up the model training and inference processes. Serra and Karatzoglou [130] applied compression techniques to transform high-dimensional data into embedding vectors before fitting recommendation models, which led to a drastic reduction in the space and computational time during model optimization. Tang and Wang [147] utilized knowledge distillation to train a small "student" model (which was extremely efficient in terms of storage and computation time) that could absorb knowledge from a large "teacher" model (which would be highly expressive and generalize well).

Handling Sparsity: In order to deal with the challenge of sparsity, *deep ANNs have been used in an automatic feature engineering process that integrates content-based features (user and item "side information") directly into a neural-based recommendation algorithm.* Strub et al. [139] utilized a denoising autoencoder architecture that takes into account side information as inputs. Gunawardana and Meek [47] used an RBM architecture that also uses side information as inputs. Kim et al. [64] employed a CNN architecture to collect image features for a tag-aware recommendation system. Tuan and Phuong [151] crafted a 3D-CNN model that collects users' browser activities to produce a session-based recommendation system.

Another way to tackle sparsity is *to use an ANN to extract high-level representations of users and items based on a preference matrix as well as side information, and then use them with matrix factorization to conduct collaborative filtering.* Wang et al. [162] presented the collaborative deep learning framework based on a Bayesian stacked denoising autoencoder that is capable of learning rich, high-level user and item representations. Li et al. [82] came up with a marginalized denoising autoencoder that learns from user's side information and subsequently combines the learning with a matrix factorization process. Oord et al. [153] designed a CNN that extracts high-level audio signal features to be used in a content-based recommendation framework. Wang and Wang [163] devised a hybrid method that combines probabilistic matrix factorization with features learned from a deep belief network. Wang et al. [161] developed a relational probabilistic stacked denoising autoencoder that can extract side information – this was also finally combined with matrix factorization. Shin et al. [134] adapted a hybrid of word2vec model (such as skip-gram or continuous bag-of-words) and a CNN – the model was used to extract text and image features which were then integrated into a matrix factorization process for blog recommendation. Shen et al. [133] showcased a CNN that extracts latent factors for educational-technology (ed-tech) recommendation. Ying et al. [167] birthed a stacked denoising autoencoder that extracts "deep features" from side information, but these representations were ultimately used in a Bayesian probabilistic framework for pair-wise ranking model.

Handling Accuracy: In order to deal with the challenge of accuracy, *deep ANNs have been used to extract hidden features / latent factors to improve recommendation prediction.* Sedhain et al. [129] employed autoencoders inside a framework called "AutoRec" to predict missing ratings. Zheng et al. [174] used the Neural Autoregressive Distribution Estimation (NADE) method in the collaborative filtering setting by sharing the model parameters across different ratings. Wu et al. [166] devised the collaborative denoising autoencoder to reconstruct the dense form of user preferences – this approach reached state-of-the-art performance for top-N recommendation task. Unger et al. [152] combined PCA and autoencoders to capture latent features from sensor data for a context-aware system. Deng et al. [30] used autoencoders to capture latent features from user and item data for a trust-aware system.

Another way to tackle accuracy is *to use ANNs to jointly combine information from different data sources/modalities.* Truyen et al. [150] developed an RBM-based scheme to jointly model user-based and item-based correlations in tandem with Singular Value Decomposition. Wang et al. [161] used a probabilistic stacked denoising autoencoder to capture the relation between items for a tag-aware system. For a group-based recommendation task, Hu et al. [62] constructed a deep architecture built using a collection of deep belief networks and dual-wing RBMs to extract collective features from users/members and combined these with group preferences.

2.2.3 Research Directions

Across several surveys written in recent years [8, 169], here are some of the key, outstanding open research directions:

- When working with large-scale datasets in the real world, optimizing the exponential growth of model parameters is challenging. A model that works well with small, offline datasets cannot be guaranteed to perform well on large online datasets. Although the solutions (discussed in Section 2.2.2) meant to address **scalability** are promising, further studies are crucially needed that focus on a large-scale evaluation of industry-level recommendation systems [40, 50, 106, 127].

- User preferences and tastes evolve. To account for these **temporal dynamics** in a session-based recommendation, more advanced sequential models are needed. RNNs have proven to be very effective [24, 55, 56, 112, 122, 146, 165] but can certainly be improved with regard to accuracy metrics, especially in addressing their difficulties in capturing long-term dependencies in sequential data.

- The quality of a recommendation system depends on the quality of the data. As **privacy** becomes more important in the technology world, there is a need to design solutions that will sensibly utilize user data. Therefore, building models on privately-collected, centrally-stored, or widely-distributed data is fundamental for further near-term progress to be made [5, 14, 22, 74, 95, 109, 115, 135, 164].

- In the early stage of a recommendation process, the users are likely to explore new and diverse directions. In these cases, the recommendation system can be utilized as a knowledge discovery tool. Many research studies [93, 94, 145] have attempted to characterize the nature of **diversity** - examining diversity among different recommendation sessions or within a single session – and have explored how to optimize the objectives of diversity and accuracy jointly.

- **Cross-domain** recommendation systems can mediate user data thanks to the knowledge acquired across different systems and application domains [12, 13, 35, 63, 68, 105]. One can consider the cross-domain recommendation to be very similar to "transfer learning" in deep ANNs - an agent must leverage the knowledge learned from one domain to improve the learning for another domain. This direction is promising but is largely an open problem for automatic recommendation to this day.

Chapter 3

Meta Learning

3.1 Meta Learning Overview

As a result of many advancements in algorithms and computing hardware in the past decade, combined with an ever increasingly deluge of data, deep artificial neural networks (ANNs) have allowed us to process unstructured data (such as images, text, audio, video, etc.) without the need to engineer features by hand. Empirical research has shown that ANNs generalize better (to unseen test examples) if we feed them large and diverse inputs. For example, Transformers [156] such as GPT-2 [113] and GPT-3 [20] have made massive headlines in the natural language processing research community in the last few years due to their broad applicability across various tasks. However, there are problems with using ANNs in real-world settings. Some of the central issues include:

- **Large datasets are unavailable**: This issue is common in many domains ranging from classification of rare diseases to translation of rare languages. In these scenarios, it is clearly impractical to learn from scratch for each task given the dearth of data.
- **Data has a long tail**: This issue can easily break the standard machine learning framework/pipeline. For example, in the self-driving car setting, an autonomous vehicle can be trained to handle common situations very well, but the agent often struggles with uncommon situations (such as people jay-walking, animals crossing, traffic lines not working).
- **Learning a new task without training from scratch is difficult**: Human agents are known to do this quite easily by leveraging prior experience/knowledge when faced with a new task. For example, if ones know

some Spanish, then it would be less difficult to learn Italian given that these two languages are quite similar linguistically (they share linguistic roots in Latin).

To combat the issues above, one promising direction to explore is that of **meta-learning**, or the process of "learning how to learn". The rest of this chapter looks into the evolution of meta-learning in more detail, which we also previously discussed in our published article [78]. In this setup/context, an ANN should learn a new task more proficiently, given knowledge acquired over previously seen tasks, assuming that it is given access to data on previous tasks. Historically, there have been various efforts made towards developing frameworks for and approaches to meta-learning in ANNs. In 1992, Samy Bengio et al. [10] looked at the possibility of a powerful (meta-)learning rule to solve new tasks. Sebastian Thrun [149], in 1995, explored the problem of lifelong learning, which is inspired by the ability of humans to exploit experiences that come from related learning tasks to generalize to new, unseen tasks. Two years later, Rich Caruana [23] wrote a survey about multi-task learning, a variant of meta-learning. He explained how tasks could be learned in parallel using a shared representation between models and presented a multi-task inductive transfer notion that uses back-propagation to handle/tune the model to additional tasks.

In recent times, meta-learning has experienced a vigorously renewed interest, given its importance and implication for building more general-purpose, adaptive agents. Many recent studies have leveraged meta-learning algorithms (and variations) to improve generalization for various tasks. For example, Aharoni et al. [1] expanded the number of languages used in a multi-lingual neural machine translation setting from 2 to 102. Their method learns a small number of languages and generalizes them to a vast set of languages previously unseen by their system. Yu et al. [168] presented domain-adaptive meta-learning, a system that allows robots to learn from a single video of a human given an inductive bias learned from prior meta-training data collected from related tasks. The engineering team at YouTube [173] showed how they used multi-task methods to make video recommendations and handle multiple competing ranking objectives. Looking forward, the development of meta-learning algorithms will help democratize deep learning and solve problems in domains with limited data.

3.2 Meta Learning Formulation

In the standard supervised learning setup, we want to maximize the likelihood of model parameters ϕ given the training data D:

$$\arg\max_{\phi} \log p(\phi|D). \tag{3.1}$$

Equation (3.1) can be redefined as maximizing the probability of the data given the parameters and maximizing the marginal probability of the parameters, where $p(D|\phi)$ corresponds to the data likelihood and $p(\phi)$ corresponds to a regularization term:

$$= \arg\max_{\phi} \log p(D|\phi) + \log p(\phi). \tag{3.2}$$

Assuming that the data D consists of (input, label) pairs of (x_i, y_i), Equation (3.2) can be further broken down as follows:

$$= \arg\max_{\phi} \sum_{i} \log p(y_i|x_i, \phi) + \log p(\phi) \tag{3.3}$$

However, if we dealing with a very large dataset D (as is the case with most complicated problems), our model will likely overfit. Even with the introduction of regularization (terms), there is still a risk of model overfitting/overparameterization unless careful, extensive tuning is conducted.

In contrast to the standard supervised setting defined above, the key problem that (supervised) meta-learning is concerned with solving is: *Is it feasible to get more data when dealing with supervised learning problem*? Ravi and Larochelle [116] is one of the first papers to provide a standard formulation of the meta-learning setup, as shown in Figure 3.1. They reframe Equation (3.1) to Equation (3.4) below, where $D_{meta-train}$ is the meta-training data that allows our model to learn more efficiently. Formally, the problem is:

$$\arg\max_{\phi} \log p(\phi|D, D_{meta-train}). \tag{3.4}$$

Here, $D_{meta-train}$ corresponds to a set of datasets for predefined tasks, i.e., $D_1, D_2, \cdots, D_n$. Furthermore, they design a set of **meta-parameters** $\theta = p(\theta|D_{meta-train})$, which includes the necessary information about $D_{meta-train}$ in order to solve the new tasks.

Mathematically speaking, with the introduction of this intermediate variable θ, the full likelihood of parameters for the original data given the meta-training data (in Equation (3.4)) can be expressed as an integral over the

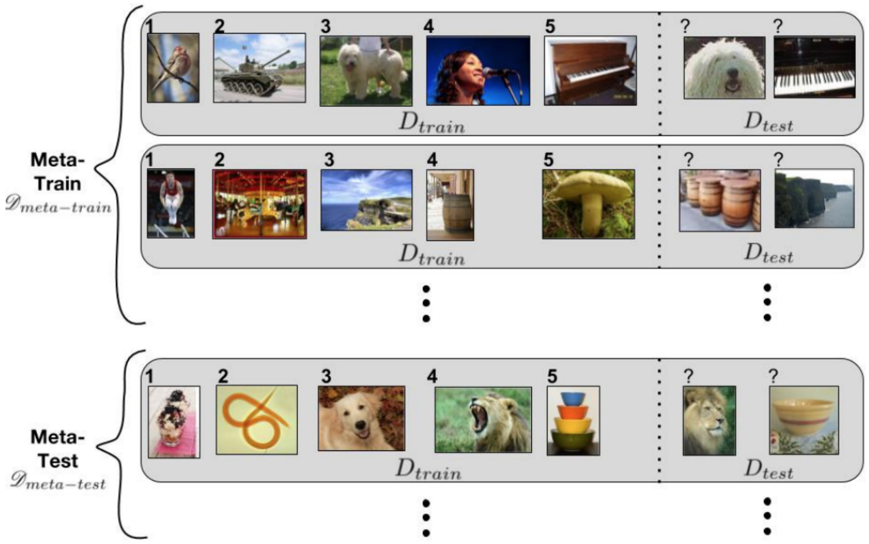


Figure 3.1: The meta-learning problem setup described in [116].

meta-parameters θ:

$$\log p(\phi|D, D_{meta-train}) = \log \int_{\Theta} p(\phi|D,\theta)p(\phi|D_{meta-train})d\theta \tag{3.5}$$

where we also note that the above equation can be approximated further with a point estimate of our parameters:

$$\approx \log p(\phi|D,\theta^*) + \log p(\theta^*|D_{meta-train}). \tag{3.6}$$

Note that $p(\phi|D,\theta^*)$ is the **adaptation** task that collects task-specific parameters ϕ for a new task - assuming that the agent has access to the data from that task D and meta-parameters θ. Furthermore, $p(\theta^*|D_{meta-train})$ is the **meta-training** task that collects meta-parameters θ - assuming that the agent has access to the meta-training data $D_{meta-train}$.

In sum, the meta-learning process can be broken down into two, key phases:

- **The adaptation phase**: $\phi^* = arg\max_{\phi} \log p(\phi|D,\theta^*)$ (first term in (3.6) that needs to be optimized).
- **The meta-training phase**: $\theta^* = arg\max_{\theta} \log p(\theta|D_{meta-train})$ (second term in (3.6) that needs to be optimized).

With respect to model parameter optimization, our meta-training data consists of pairs of training and test sets for each and every task:

$$D_{meta-train} = \{(D_1^{train}, D_1^{test}), \cdots, (D_n^{train}, D_n^{test})\} \tag{3.7}$$

where there are k feature-label pairs (x, y) in the training set D_i^{train} and l feature-label pairs (x, y) in the test set D_i^{test}, or:

$$D_i^{train} = \{(x_1^i, y_1^i), \cdots, (x_k^i, y_k^i)\}; D_i^{test} = \{(x_1^i, y_1^i), \cdots, (x_l^i, y_l^i)\} \tag{3.8}$$

During the adaptation phase, we infer a set of task-specific parameters ϕ^*, which is a function that takes as input the training set D^{train} and returns as output the task-specific parameters: $\phi^* = f_{\theta^*}(D^{train})$. Essentially, we want to learn a set of meta-parameters θ such that, the function $\phi_i = f_\theta(D_i^{train})$ is good enough for the test set D_i^{test}. Finally, during the meta-learning phase, to get meta-parameters θ^*, we want to maximize the probability of the task-specific parameters ϕ being effective on new data points in the test set D_i^{test}:

$$\theta^* = \max_\theta \sum_{i=1}^{n} \log p(\phi_i | D_i^{test}). \tag{3.9}$$

We next review key instantiations of the above meta-learning process.

3.3 Black-Box Meta-Learning

3.3.1 Formulation

The black-box meta-learning approach uses an ANN architecture to generate the distribution $p(\phi_i | D_i^{train}, \theta)$. More specifically, given task-specific parameters $\phi_i = f_\theta(D_i^{train})$, the process proceeds as follows:

- An ANN with meta-parameters θ (denoted as f_θ) takes in the training data D_i^{train} as input and returns task-specific parameters ϕ_i as output.
- Another ANN (denoted as $g(\phi_i)$) takes in the task-specific parameters ϕ_i as input and returns the predictions for test data points D_i^{test} as output.

To optimize model parameters, we maximize the log likelihood of the outputs from $g(\phi_i)$ for all the test data points. This is done for all of the tasks in the meta-training set:

$$\max_\theta \sum_{T_i} \sum_{(x,y) \sim D_i^{test}} \log g_{\phi_i}(y|x) \tag{3.10}$$

The above log likelihood of $g(\phi_i)$ is essentially the loss between a set of task-specific parameters ϕ_i and a test data point D_i^{test}, or:

$$\sum_{(x,y)\sim D_i^{test}} \log g_{\phi_i}(y|x) = L(\phi_i, D_i^{test}). \tag{3.11}$$

In Equation (3.10), we actually optimize the loss between the function $f_\theta(D_i^{train})$ and the evaluation on the test set D_i^{test}:

$$\max_\theta \sum_{T_i} L(f_\theta(D_i^{train}), D_i^{test}) \tag{3.12}$$

Building on the equations/ideas above, the black-box meta-learning process can then be fully summarized according to the following recipe:

1. We sample a task T_i, the training set D_i^{train}, and the test set D_i^{test} from the task dataset D_i.
2. We compute the task-specific parameters ϕ_i given the training set D_i^{train}: $\phi_i \leftarrow f_\theta(D_i^{train})$.
3. Then, we update the meta-parameters θ using the gradient of the objective with respect to the loss function between the computed task-specific parameters ϕ_i and D_i^{test}: $\nabla_\theta L(\phi_i, D_i^{test})$.
4. This process is repeated iteratively with gradient descent optimization.

3.3.2 Challenges

The main issue with the black-box approach depicted occurs when ϕ_i happens to be large/massive. If ϕ_i, for example, is a set of all the parameters of a very deep ANN, then **it is not scalable/feasible to output** ϕ_i. Santoro et al. [126] and Mishra et al. [97] attempt to tackle this problem. Instead of having an ANN that outputs all of the parameters ϕ_i, they output a low-dimensional vector h_i, which is then used alongside meta-parameters θ to make predictions. The new task-specific parameters ϕ_i take the form: $\phi_i = \{h_i, \theta_g\}$, where θ_g represents all of the parameters other than h. Overall, the general form of this variant of the black-box approach can be depicted formally as follows:

$$y^{ts} = f_\theta(D_i^{train}, x^{ts}) \tag{3.13}$$

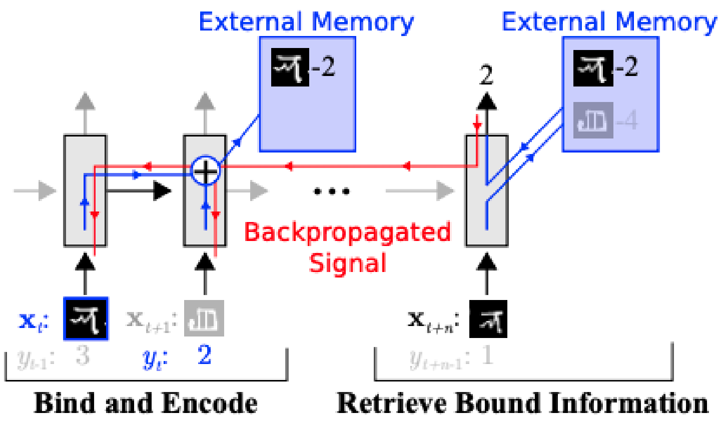


Figure 3.2: The memory-augmented network as described in [126].

where y^{ts} corresponds to the labels of test data, x^{ts} corresponds to the features of test data, and D_i^{train} corresponds to pairs of training data. This approach was shown to facilitate a more scalable black-box meta-learning process since h is of low complexity (in terms of parameters).

3.3.3 Architectures

Many different model architectures could be chosen to represent the function f. For example, memory-augmented networks, proposed by Santoro et al. [126], used Long Short-Term Memory (LSTM) and Neural Turing Machine (NTM) models to represent f. Both architectures have an external memory mechanism to store information about the training data and can access that information during inference in a differentiable way, as shown in Figure 3.2. Garnelo et al. proposed conditional neural processes [39] and chose to represent f in three steps: (1) using a feedforward network to compute the training data, (2) aggregating that data, and (3) passing that aggregation to another feedforward network for inference. In contrast, meta networks (proposed by Munkhdalai and Yu [101]) used other external memory mechanisms with slow and fast weights inspired by neuroscience to represent f. Specifically, the slow weights were designed for meta-parameters θ, and the fast weights were designed for task-specific parameters ϕ. The neural attentive meta-learner, proposed by Mishra et al. [97], used an attention mechanism to represent f. Such a mechanism allowed the network to "pick out" the most critical information from the data it gathers, thus making the optimization process far more efficient.

In general, black-box meta-learning algorithms have a powerful learning capacity because ANNs are universal function approximators. However, if we impose a particular structure into the function, there is no guarantee that black-box models will produce consistent results. Desirably, we can use black-box approaches with different problem settings, including reinforcement learning and self-supervised learning. However, despite their generality, since black-box models always learn from scratch, they are incredibly data-hungry.

3.4 Optimization-Based Meta-Learning

To represent the distribution $p(\phi_i|D_i^{train}, \theta)$ in the adaptation phase of meta-learning, optimization-based meta-learning approaches try to optimize the process of getting the task-specific parameters ϕ_i such that strong generalization performance is achieved on the test set.

3.4.1 Formulation

Recall that the meta-learning problem can be broken down into two terms (as shown below) – one that maximizes the likelihood of training data given the task-specific parameters and one that maximizes the likelihood of task-specific parameters given meta-parameters:

$$\max_{\phi_i} \log p(D_i^{train}|\phi_i) + \log p(\phi_i|\theta) \tag{3.14}$$

where the meta-parameters θ are "pre-trained" during training time and then fine-tuned at test time. The equation below is a typical optimization procedure that adapts parameters via gradient descent:

$$\phi \leftarrow \theta - \alpha \nabla_\theta L(\theta, D^{train}) \tag{3.15}$$

where α is the learning rate. To get the pre-trained parameters, we can use standard benchmark datasets such as ImageNet for computer vision, Wikipedia Text Corpus for language processing, or any other large and diverse dataset that we have access to and is relevant for the modality/type of inputs/outputs we want to model. As expected, this approach becomes less effective when only small quantities of training are available.

To combat the potential issue of limited/small datasets, **Model-Agnostic Meta-Learning** (MAML) from Finn et al. [36] is an algorithm was proposed as one potential solution. Specifically, it takes the optimization procedure in

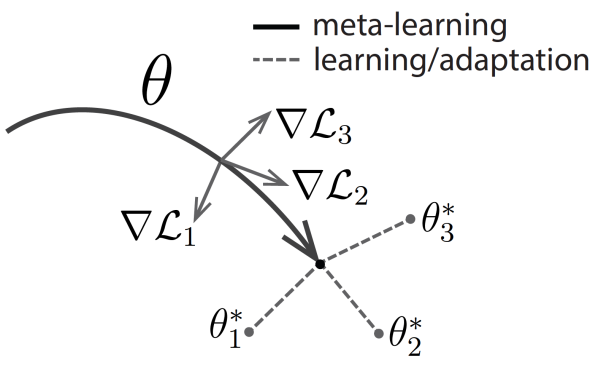


Figure 3.3: Diagram of the MAML method as described in [36].

Equation (3.15) and adjusts the loss so that only the best-performing task-specific parameters ϕ on test data points are considered. This happens across all of the tasks:

$$\min_{\theta} \sum_{task_i} L(\theta - \alpha\nabla_{\theta} L(\theta, D_i^{train}), D_i^{test}) \tag{3.16}$$

The key idea is to learn θ for all the assigned tasks such that θ can transfer effectively via the optimization process (see Figure 3.3). In sum, the optimization-based meta-learning algorithm proceeds as follows:

1. We sample a task T_i, the training set D_i^{train}, and the test set D_i^{test} from the task dataset D_i.

2. We compute the task-specific parameters ϕ_i given the training set D_i^{train} using the optimization procedure described above: $\phi_i \leftarrow \theta - \alpha\nabla_{\theta} L(\theta, D_i^{train})$.

3. Then, we update the meta-parameters θ using the gradient of the objective with respect to the loss function between the computed task-specific parameters ϕ_i and D_i^{test}: $\nabla_{\theta} L(\phi_i, D_i^{test})$

4. This process is repeated iteratively via gradient descent.

As explained in the last section, the black-box meta-learning approach takes the general form: $y^{test} = f_{\theta}(D_i^{train}, x^{test})$. The optimization-based MAML method described above has a similar form as shown below:

$$y^{test} = f_{MAML}(D_i^{train}, x^{test}) = f_{\phi_i}(x^{test}) \tag{3.17}$$

with the update: $\phi_i = \theta - \alpha\nabla_{\theta} L(\phi, D_i^{train})$.

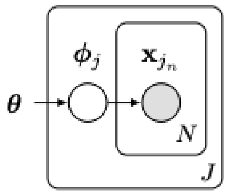


Figure 3.4: The probabilistic graphical model for which MAML provides an inference procedure, as described in [43].

To prove the effectiveness of the MAML algorithm, in [37], Finn and Levine demonstrated that the MAML algorithm can approximate any function of D_i^{train}, x^{test} for a complex function f. This finding demonstrates that the optimization-based MAML algorithm is as expressive as the other black-box algorithms mentioned so far.

3.4.2 Architectures

In [43], Grant et al. provided another MAML formulation as a method for **probabilistic inference via hierarchical Bayes**. Assume that we have a graphical model as illustrated in Figure 3.4, where J is the task, x_{j_n} is a data point in that task, ϕ_j are the task-specific parameters, and θ are the meta-parameters. To conduct inference with respect to this graphical model, we want to maximize the likelihood of the data given the meta-parameters:

$$\max_\theta \log \prod_j p(D_j|\theta) \tag{3.18}$$

The probability of the data given the meta-parameters can be expanded into the probability of the data given the task-specific parameters and the probability of the task-specific parameters given the meta-parameters. Thus, Equation (3.18) can be rewritten as follows:

$$= \log \prod_j \int p(D_j|\phi_j)p(\phi_j|\theta)d_{\phi_j}. \tag{3.19}$$

This integral in equation (3.19) can be approximated with a Maximum a Posteriori (MAP) estimate for ϕ_j:

$$\approx \log \prod_j p(D_i|\hat{\phi_j})p(\hat{\phi_j}|\theta). \tag{3.20}$$

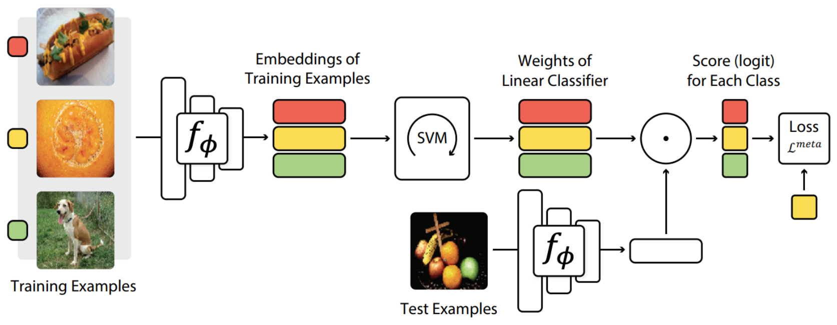

Embeddings of Training Examples
Weights of Linear Classifier
Score (logit) for Each Class
f_ϕ
SVM
Loss $\mathcal{L}^{meta}$
f_ϕ
Training Examples
Test Examples

Figure 3.5: The MetaOptNet approach as described in [81].

To compute the desired MAP estimate above, Grant et al. [43] proposes conducting the inference under an implicit Gaussian prior - with a mean that is determined by the initial parameters and variance that is determined by the number of gradient steps and the step size. Other proposals for computing the MAP estimate in Equation (3.20) include:

- Rajeswaran et al. [114] proposes an **implicit MAML algorithm** that uses gradient descent with an *explicit Gaussian prior*. More specifically, they regularize the inner optimization of the algorithm to be close to the meta-parameters θ: $\phi \leftarrow \min_{\phi'} L(\phi', D^{train}) + \frac{\lambda}{2}||\theta - \phi'||^2$. The mean and the variance of this explicit Gaussian prior is a function of λ regularizer.

- Harrison et al. [52] proposes the **ALPaCA algorithm** that uses an efficient Bayesian linear regression on top of the learned features from the inner optimization loop to represent the mean and variance of that regression as meta-parameters themselves. The inclusion of prior information here reduces computational complexity and adds more confidence to the final predictions.

- Bertinetto et al. [15] attempts to solve meta-learning with **differentiable closed-form solutions**. They apply a ridge regression as a base learner for the features in the inner optimization loop. The mean and variance predictions from the ridge regression are then used as meta-parameters in the outer optimization loop.

- Lee et al. [81] attempts to solve meta-learning with **differentiable convex optimization solutions**. The proposed method, called **MetaOptNet**, uses a support vector machine to learn the features from the inner optimization loop (see Figure 3.5).

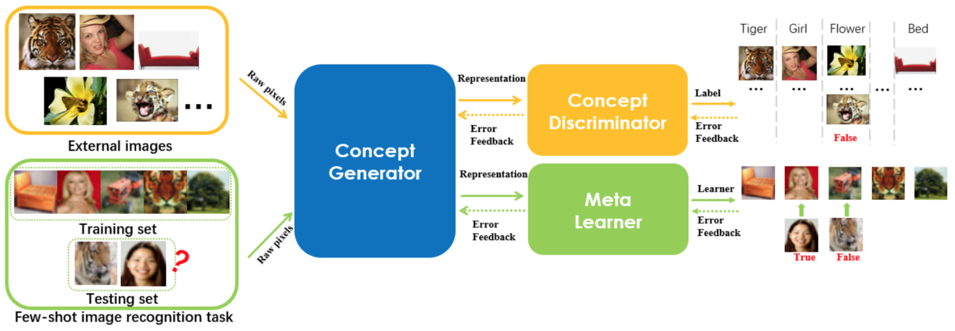


Figure 3.6: The Deep Meta Learning as described in [175].

3.4.3 Challenges

The MAML method requires very deep neural architecture in order to effectively obtain a good inner gradient update. Therefore, the first challenge lies in choosing that architecture. Kim et al. [65] proposed Auto-Meta, which searches for the (near optimal) MAML architecture. They found that highly non-standard architectures with deep and narrow layers tended to perform very well.

The second challenge relates to the unreliability of the two-degree optimization paradigm. There are many different optimization tricks that can be useful to combat this unreliability. For example, Li et al. [84] proposed Meta-SGD that learns the initialization parameters, the direction of the gradient updates, and the value of the inner learning rate in an end-to-end fashion. This method has proven to increase speed and accuracy of the meta-learner. Behl et al. [9] designed Alpha-MAML, which is an extension of the vanilla MAML. Alpha-MAML uses an online hyper-parameter adaptation scheme to automatically tune the learning rate, making the training process more robust. Zhou et al. [175] devised deep meta-learning, which performed meta-learning in a concept space. As illustrated in Figure 3.6, the concept generator generates the concept-level features from the inputs, while the concept discriminator discriminates the features generated from that first step. The final loss function includes both the loss from the discriminator and the loss from the meta-learner. Zintgraf et al. [176] designed CAVIA, which stands for fast context adaptation via meta-learning. To handle overfitting inherent to vanilla MAML, CAVIA optimizes only a subset of the input parameters in the inner loop at test time (deemed *context parameters*) instead of the whole ANN. By separating task-specific parameters from task-independent parameters, they showed that training CAVIA is very efficient. Finally, Antoniou et al. [4] proposed **MAML++**, a comprehensive guideline on reducing the hyper-parameter sensitivity, lowering the generalization error, and improving MAML stability. Furthermore, they suggested "disentangling" both the learning rate and the batch-norm statistics per step of the inner loop to speed up optimization.

The third challenge lies in the computational expense associated with back-propagation. The more inner gradient steps used, the more challenging the optimization process becomes. There are two approaches to deal with this. The first comes the studies of Finn et al. [36] and Nichol et al. [102], which proposed truncating the back-propagation by approximating the $\frac{d_{\phi_i}}{d\theta}$ matrix as an identity function. This proves to work for simple, few-shot learning problems (small number of tasks). The second comes from the work Rajeswaran et al. [114], which applies a theorem to compute the meta-gradient $\frac{d_{\phi_i}}{d\theta}$ implicitly. This algorithm's benefit is that the outcome only depends on the inner optimization's solution, but not the number of inner gradient steps required.

In general, optimization-based meta-learning algorithms can be reduced to gradient descent; thus, it is reasonable to expect consistent predictions. For deep enough ANNs, optimization-based models also demonstrate very high capacity – since the initialization is optimized internally, optimization-based models have a "head-start" over black-box models. Furthermore, we can try out different architectures without any real difficulty, as evidenced by the Model-Agnostic Meta-Learning (MAML) learning paradigm. However, the second-order optimization procedure makes optimization-based approaches computationally expensive.

3.5 Non-Parametric Meta-Learning

Non-parametric methods are very effective at learning with small amounts of data (k-Nearest Neighbor, decision trees, support vector machines). In **non-parametric meta-learning**, we compare the test data with the training data using a similarity metric. If we find the training samples that are most similar to the test samples, we assign the labels of those training data as the label of the test data. Meta-learning can also be designed to follow the non-parametric form of learning.

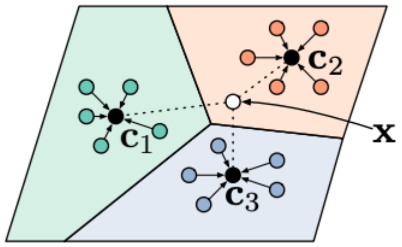


Figure 3.7: Prototpyical Networks as described in [136]

3.5.1 Formulation

Specifically, the non-parametric form of meta-learning can be compactly summarized as follows:

1. We sample a task T_i, the training set D_i^{train}, and the test set D_i^{test} from the task dataset D_i.
2. We predict the test label $\hat{y}^{test}$ via the similarity between training data and test data (represented by f_θ): $\hat{y}^{test} = \sum_{x_k, y_k \in D^{train}} f_\theta(x^{test}, x_k) y_k$.
3. Then we update the meta-parameters θ of this learned embedding function with respect to the loss function of how accurate our predictions are on the test set: $\nabla_\theta L(\hat{y}^{test}, y^{test})$
4. This process is repeated iteratively using gradient descent.

Unlike the black-box and optimization-based approaches, we no longer have the task-specific parameters ϕ (which is not required for the comparison between training and test data).

3.5.2 Architectures

Much as is the case for black-box based and optimization-based methods, there are many different architectures that can be used within the-parametric meta-learning process. Koch et al. [69] proposed a Siamese network that consists of two tasks: a verification task and a one-shot task. The network takes in pairs of images during training time and verifies whether they are of the same class or different classes. At test time, the network performs one-shot learning: comparing each image x_{test} in test set to the images in training set D_j^{train} for a respective task, and predicting the label of x_{test} that corresponds to the label of the closest image. Matching Networks were proposed by Vinyals et al. [158], which match the actions happening during training time at test time. The network takes the training data and the test data and embeds them into its respective embedding space. Then, the network compares each pair of train-test embeddings to make the final label predictions. The Matching Network architecture used in Matching Networks includes a convolutional encoder network to embed the images and a bi-directional Long-Short Term Memory network to produce such images' embeddings. Snell et al. [136] proposed the "prototypical network", which creates prototypical embeddings for all of the classes in a given dataset. The network then compares these embeddings to make the final label predictions for the corresponding class. Figure 3.7 provides a concrete illustrations of what Prototypical Networks look like in the few-shot scenario. c_1, c_2, and c_3 are the class prototypical embeddings. Using these embeddings, one can compute the distances from input data x to each of the prototypical class embeddings: $D(f_\theta(x), c_k)$. To get the final class prediction $p_\theta(y = k|x)$, we look at the probability of the negative distances after applying the softmax transformation.

3.5.3 Challenges

For non-parametric meta-learning, how can we learn deeper/more complex interactions between our inputs? Nearest neighbors will probably not work as well when the data is high-dimensional. Several efforts have tried to address this issue. The "RelationNet" was proposed by Sung et al. [142], which has two modules: the embedding module and the relation module. The embedding module embeds the training and test inputs to training and test embeddings. Then the relation module takes in the embeddings and learns a deep distance metric to compare those embeddings. Allen et al. [2] created a model called the "infinite mixture of prototypes". This is an extension of the Prototypical Networks, in the sense that it adaptively sets the model capacity based on the data complexity. By assigning each class its own cluster, this method allows the use of unsupervised clustering, making the method a bit more general. In contrast, Garcia and Bruna [38] designed a graph neural network in their meta-learning paradigm. By mapping the inputs into a graphical representation, the system can quickly learn the similarity between training and test data samples using the edge and node features.

In general, non-parametric meta-learning algorithms exhibit adequate learning capacity for most architectures and good learning consistency under the assumption that the learned embedding space is "effective enough". Furthermore, non-parametric approaches do not involve any back-propagation, so they are computationally fast and easy to optimize. The downside is that these approaches are hard to scale to large data batches since they are non-parametric.

Chapter 4

Literature Review

This section presents an extensive analysis of state-of-the-art recommendation models using linear models, artificial neural networks (ANN), and meta-learning-based approaches. There has been an increasing amount of interest in the past five to seven years of research in the recommendation systems community on how to incorporate deep artificial neural network (ANN) architectures into the automatic recommendation process. These advancements stand in contrast with the practical usage and deployment of models in the industry, where engineers and scientists rely on simple linear models due to engineering and business requirements. In light of this disparity between scientific and industry perspectives, we will try to provide a more holistic view of the field, considering both important advances and industry considerations. We will start by revisiting matrix factorization, the most popular linear recommendation model.

4.1 Matrix Factorization For Recommendation

Recall in Chapter 2, collaborative filtering algorithms analyze past user behavior in order to establish relationships between users and items. Users with similar preferences in the past will received similar preferences in the future. There are two common challenges with the implementation of these algorithms:

- The dataset size is huge in real-world applications.
- The rating matrix is often (very) sparse, where only a small number of users give ratings to a small number of items.

Matrix factorization [71, 119] has proven to be useful model when attempting to tackle the above challenges. Historically, it has been used for latent

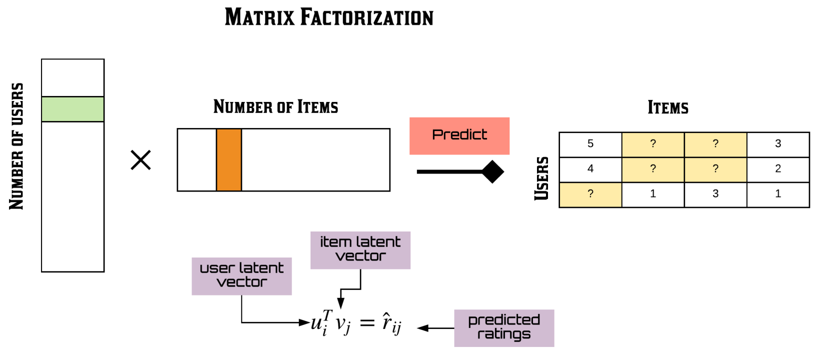


Figure 4.1: A typical matrix factorization process (diagram adapated from [45]).

variable decomposition and dimensionality reduction [71]. More specifically, matrix factorization (Figure 4.1) is a latent factor model that represents the rating matrix as the product of a user factor matrix and an item factor matrix.

We briefly review the four key matrix factorization models. Singular Value Decomposition (SVD) is a method that finds a lower-dimensional feature space for a given matrix by breaking it down into three separate orthogonal matrices [41,109,160]. Principal Component Analysis (PCA) is a classical statistical method (which usually builds on the mechanics of SVD) that finds patterns in data with high dimensions [159]. It obtains an ordered list of components that account for the largest variance from the data in terms of least square errors. Probabilistic Matrix Factorization adds a probabilistic spin to matrix factorization with Gaussian observation noise [124]. Non-Negative Matrix Factorization is the standard matrix factorization model, with a caveat that matrices cannot contain negative elements. It has proven to work well with non-negative data formats such as images and videos [46, 148].

4.2 Multilayer Perceptrons For Recommendation

Despite the effectiveness of matrix factorization for collaborative filtering, it is well-known that its performance can be hindered by the simple choice of the interaction function, i.e., the inner product. For example, for the task of rating prediction on explicit feedback, it is a common practice to improve the performance of the matrix factorization model by incorporating user and item bias terms into the interaction function. While this, at first glance, might

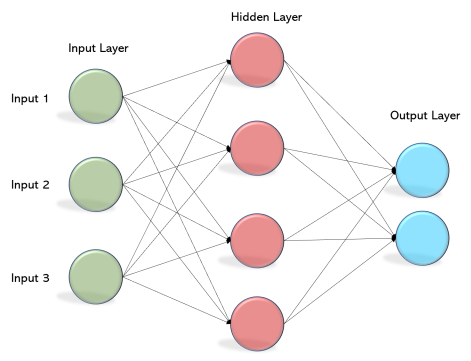


Figure 4.2: A typical multi-layer perceptron architecture (diagram taken from [99]).

appear to be just a trivial tweak for the inner product operator, it reveals that there is a positive effect if one designs a better, dedicated interaction function for modeling the latent feature interactions between users and items. The inner product, which combines the multiplication of latent features linearly, may be insufficient to capture the actual complex structure underlying user interaction data.

The **multilayer perceptron** (MLP) (Figure 4.2) is a feed-forward ANN with multiple hidden layers between the input layer and the output layer. Using this model means replacing the inner product of matrix factorization with a stack of interleaved linear and non-linear transformations. In between each layer is a (dense) matrix of scalar synaptic weight values, which represent the trainable parameters of the MLP model. To adjust these synaptic weight values, assuming we have a cost function for measuring the MLP's predictive performance on the task at hand, we can use back-propagation of errors to automatically calculate the gradients (or partial derivatives) of the cost with respect to each and every single synaptic weight. The weights themselves are adjusted with stochastic gradient descent (or a more outer optimization procedure).

There are several prevalent and powerful MLP-based methods for the recommendation task. Many of them take the approach of trying to construct a model that models both users and items. Neural Collaborative Filtering [54] and Neural Network Matrix Factorization [33] are two models that use complex ANN structures to represent the two-way interaction between user preferences and item features. On the other hand, the Deep Factorization Machine [49] is an end-to-end model that combines a matrix factorization process and an

MLP. It can model the high-order feature interactions of input variables using deep ANN and low-order interactions with the matrix factorization machine. The factorization machine captures the linear and pairwise interactions between features, while the MLP captures the nonlinear and hierarchical structure between features. As an extension of the Deep Factorization Machine, the Extreme Deep Factorization Machine [85] jointly models both the explicit and implicit feature representations. More specifically, it uses a compressed interaction network to capture explicit high-order feature interactions. The Neural Factorization Machine [53] is a closely related model. It uses an MLP to capture high-order feature interactions but regularizes the MLP with dropout and batch normalization (applied to each layer).

Other MLP-based approaches focus on learning representations of item features directly. Wide and Deep Learning [27] is a model framework introduced initially by Google to solve the app recommendation problem in Google Play. The "wide" learning component is a single layer perceptron, while the "deep" learning component is an MLP. The wide learning component can catch the direct features from historical data to capture memorization, while the deep learning component can produce more general and abstract representations to capture generalization. This unification improves both the accuracy and diversity of generated recommendations. Another work from Google applies the MLP to YouTube recommendations [29], in which they divide the recommendation task into two stages: a generating candidate and a ranking candidate. The candidate generation network receives a subset of samples from the video corpus, and then the ranking network builds a top-K list of videos based on the nearest neighbor scores from the selected candidates. [26] proposed a wide and deep learning framework tailored for large-scale industrial-level recommendation tasks. In particular, the deep learning component is a tailored network that has a significantly lower running time than a standard MLP. Moreoever, the model can determine which features are memorized or generalized and allocate them to the wide and deep components for further processing. Collaborative Metric Learning [61] uses an MLP to learn item feature representations in formats such as text, images, and tags. It captures the user and item latent embeddings via (1) maximization of the distance between users and their disliked items, and (2) minimization of the distance between users and their preferred items.

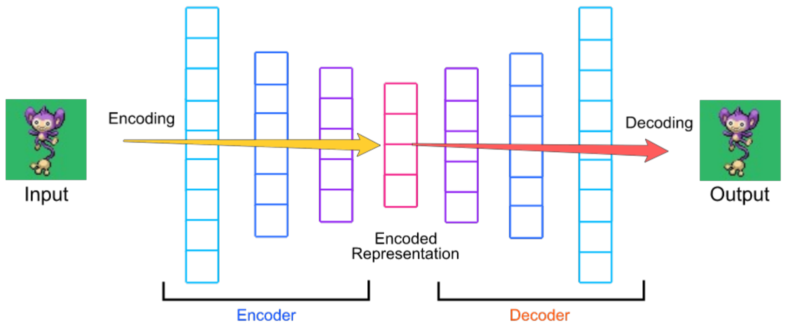


Figure 4.3: A typical Autoencoder architecture (diagram taken from [98]).

4.3 Autoencoders For Recommendation

Autoencoder (Figure 4.3) is suitable for unsupervised learning tasks and can acquire rich, latent feature representations in tasks ranging from image recognition [108] to speech recognition [90]. Inspired by autoencoders' expressive power, recent recommendation architectures have incorporated autoencoders to address the challenges of traditional recommendation systems. Notably, an autoencoder can learn the non-linear user-item interaction efficiently and encode complex abstractions into data representations. Additionally, we can construct autoencoders to discover useful knowledge from multi-modal sources to address the data sparsity issue.

There are currently two ways to use autoencoder in a recommendation system: either to learn the low-dimensional feature representations at the bottleneck layer or to fill in the missing values of the user-item interaction matrix directly in the reconstruction layer. For the first use, collaborative Deep Learning [162] is a hierarchical Bayesian deep learning framework that combines stacked denoising autoencoder (SDAE) and probabilistic matrix factorization (PMF). There are two components inside this Bayesian framework: (1) a perception component (SDAE) and (2) a task-specific component (PMF). This combination enables the model to balance the influences of side information and interaction history. [83] proposed the Collaborative Variational Autoencoder (CVAE), which is an extension of CDL that replaces the perception component with a variational autoencoder. This change enables efficient learning of probabilistic latent variables for content information and incorporation of multi-media data sources. The Deep Collaborative Filtering

Framework [82] is a marginalized denoising-autoencoder-based collaborative filtering model. This autoencoder marginalizes out the corrupted input and saves high computational costs, making it more scalable than other autoencoders' variants. Collaborative Deep Ranking [167] is a variant of CDL but was designed specifically for the task of item ranking. AutoSVD++ [170] utilizes a contractive autoencoder [120] to learn item feature representations and integrates it with SVD++ [70], a classic recommendation model. Compared to other autoencoders variants, the contractive autoencoder can capture even the tiniest of input variations, making it extremely effective to learn the implicit feedback.

For the second use, [104] proposed the Autoencoder-based Collaborative Filtering (ACF) method. It decomposes the partially observed vectors in the interaction matrix by integer ratings and uses a cost function to reduce the mean squared error between actual-predicted rating pairs. However, ACF fails to handle non-integer ratings. Moreover, its decomposition of partially observed vectors increases the sparseness of input data and reduces prediction accuracy. Unlike ACF, AutoRec [129] directly takes user or item rating vectors as input data and obtains the reconstructed rating at the output layer. The more layers it has, the better the prediction performance is. The Collaborative Filtering Network (CFN) [139, 140] is another extension of AutoRec that uses SDAE to make the model more robust. Moreover, CFN incorporates side information such as user profiles and item descriptions to mitigate the sparsity and speed up the training process. In contrast, the Collaborative Denoising Autoencoder (CDAE) [166] uses a denoising autoencoder to formulate the user-item feedback data and then learns the distributed representations of the users and items. A negative sampling technique is used to sample a small subset from the negative set, which reduces the time complexity substantially. Finally, the Multi-VAE and Multi-DAE [86] are other variants of Variational Autoencoder for recommendation with implicit data. They use a principled Bayesian inference approach known as multinomial likelihood to estimate the model parameters and show favorable results compared to other commonly used likelihood functions.

4.4 Meta Learning For Recommendation

Given that meta-learning can yield agents capable of training with few training samples (for each task) and adapt to new tasks, it is not surprising that meta-learning has been used recently to tackle the cold-start issues in recommendation systems. Many of these approaches adopted optimization-based

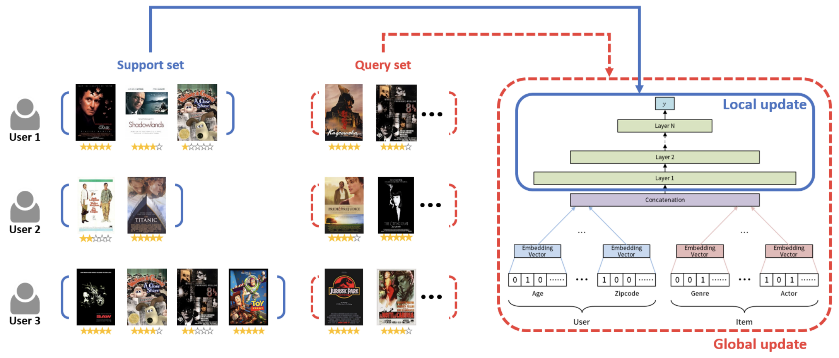


Figure 4.4: The MeLU architecture proposed in [80].

meta-learning and chose MAML for the model training process. Specifically, the approach taken is as follows:

- Given a recommendation model R_θ with parameter θ and a meta-learner $M(\theta)$, they treat each learning task as the act of capturing each user's preference.
- The initialization of parameter θ is defined by some function $\theta \leftarrow M(\theta)$, e.g. in the simplest case, $\theta \leftarrow \phi$.
- During training, θ will be locally updated by minimizing the recommendation loss $L(R_\theta(D^{train}))$ on the training set D^{train} for a single user. During inference, θ will be used to retrieve the recommendation loss $L(R_\theta(D^{test}))$ on the test set D^{test}.
- The parameter of the meta-learner ϕ will be globally updated by minimizing the sum of recommendation loss $\sum_{u \in U^{train}}(L(R_{\theta \leftarrow \phi}(D^{test})))$ for all users in the training dataset. Then, ϕ will be used to initialize θ, which can expedite the learning process of the recommendation models for test users.

The significant differences among these MAML-based recommendation systems are in designing the recommendation model R_θ and the meta-optimization approach $M(\theta)$. For example, MeLU [80] uses a feed-forward neural network as the recommendation model with multiple layers (figure 4.4). First, it takes user and item content as the input. Then, it performs embedding processes based on the input and concatenates the embedded vectors. Next, it feeds the concatenation of user and item embeddings into fully-connected layers to get predictions. For meta-optimization, MeLU locally updates the neural layers' parameters to get personalized recommendations and globally updates the embeddings' learning parameters and neural layers' global parameter. [32] proposed S^2Meta, which applies a feed-forward neural network as the recommendation model and consists of three modules: an embedding module to generate user and item embeddings, a hidden layer module that concatenates the user and item embeddings, and an output module that computes the recommendation score based on the mapped representation of user-item interaction from the last hidden layer of the neural network. For meta-optimization, s^2Meta can automatically learn to control the learning process from end-to-end, including parameter initialization, update strategy, and early-stop policy. MetaSelector [92] addresses the problem of user-level model selection to improve personalized recommendation quality. Given a recommendation request as input, it outputs a probability distribution over the recommendation models. In the meta-training phase, an initialization for MetaSelector is optimized through episodic learning with MAML. In the deployment phase, MetaSelector adapts to individual users using personalized historical data in the training set and aggregates results of recommendation models for new queries. On the other hand, SML [171] explores the central theme of model retraining in recommendations, a topic of high practical value in industry recommendation systems but receives relatively little scrutiny in research. It is a retraining method with two major attributes: (1) an expressive component that transfers the knowledge gained in previous training to the training on new interactions, and (2) an optimization scheme that moves the transfer component towards the recommendation performance in the near future. The whole architecture can be seen as an instance of meta-learning - where the retraining of each period is a task, which has the new interactions of the current period as the training set and the future interactions of the next period as the testing set.

Besides the above key papers, other recent efforts bring different meta-learning approaches into recommendation systems. [155] presents a metric-based approach to address the item cold-start problem. For conditioning the base model, two strategies are proposed: (1) a linear weight adaptation method that builds a linear classifier and adapts its weights based on the task information, and (2) a non-linear bias adaptation method that builds an ANN classifier using task information to adapt the biases of the ANN while sharing weights across all tasks. [172] tackles the challenge of making personalized recommendation for new users arriving sequentially. A two-stage meta-learning algorithm is proposed to learn the model parameters: (1) a fixed stage that

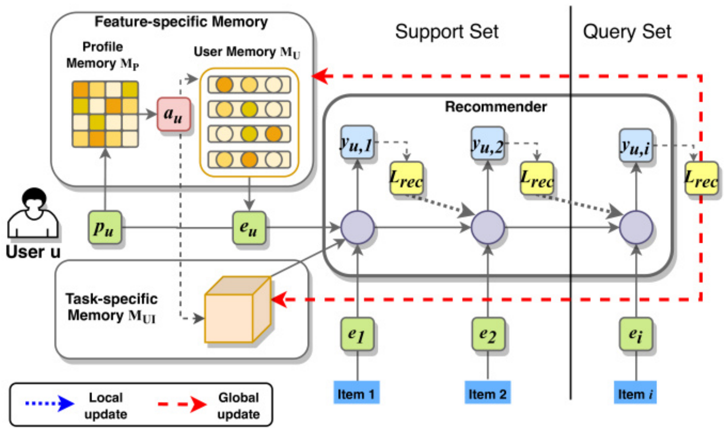


Figure 4.5: The MAMO architecture proposed in [31].

captures user-invariant features offline, and (2) an adaptive stage that captures user-specific features online. The proposed approach is more efficient and less bulky than previous methods by decoupling user-invariant parameters from user-dependent parameters. It also has the potential to deal with catastrophic forgetting while continually adapting for streaming coming users. [88] looks at the challenging problem in online recommendation systems of predicting users' current interests. The proposed framework, FLIP, can model user intent and user preference explicitly in two spaces. In the intent space, the relevant embeddings' learning is treated as a meta-learning problem instead of an online learning problem. In the preference space, the preference embedding space is learned through an online learning procedure based on samples with calibrated user intent. There are two major advantages of FLIP: (1) the user intent factors can be learned much faster in a meta-learning fashion (which is designed to learn from samples within the current session), and (2) the user preference factors could benefit from the stable continuous learning process and are updated smoothly. [91] proposes to address the cold-start recommendation problem at both the data and model levels. It uses a heterogeneous information network (HIN) to augment the data, where each meta-learning task is treated as learning the preference of each user. The proposed approach, MetaHIN, consists of two parts: (1) a semantic-enhanced task constructor to explore rich semantics on HINs in the meta-learning setting, and (2) a co-adaptation meta-learner with semantic- and task-wise adaptations to cope with different semantic facets within each task. [31] presents a black-box system that addresses the issues of an optimization-based system such as instability, slow convergence, and weak generalization. The proposed system, Memory-Augmented Meta-Optimization (MAMO) (Figure 4.5), leverages the power of memory-augmented neural networks - which can express, store, and manipulate the records explicitly, dynamically, and effectively. MAMO includes two memory matrices to provide personalized initialization for recommendation models: (1) a feature-specific memory that provides a personalized bias term when initializing the recommendation model, and (2) a task-specific memory that guides the recommendation process.

Chapter 5

MetaRec

Inspired by previous works [16,80,91] that use optimization-based meta-learning for the recommendation task, we discuss the inner workings of MetaRec in detail within this chapter. Following the typical problem formulations presented in mainstream recommendation systems research, we consider the collaborative filtering task for recommending items to each user, focusing on two important problem cases: (1) binary classification and (2) rating regression.

We denote all users' set as U and all items' set as I. In the binary classification scenario, each item is either implicitly liked (label 1) or implicitly not liked (label 0) by each user. The task is to predict the probability of whether the user p_u will like an item q_i: $P(r_{ui} = 1|p_u, q_i)$, where $r_{ui} \in \{0, 1\}$ is the rating that user p_u gives to item q_i. We want to recommend items that have a high probability of being liked by the user. In the rating regression scenario, each item is explicitly rated according to a numeric scale. For example, in the case of the MovieLens dataset (introduced in Section 6.1), the scale is $1 - 5$. The task is to predict the actual rating that user p_u will assign to an item q_i. We want to recommend movies that are highly rated by the user, i.e., 4 or 5 in the dataset.

In the context of meta-learning, we want to learn a recommendation model that generalizes to all users. With only minor fine-tuning (and limited data), the model can adapt to specific users. We define each task T as recommending items to one user, given examples of items already rated by the user. We denote the recommendation algorithm as $F_\theta(.)$ (which can be any function), where θ contains the model parameters:

$$\begin{aligned} \text{Binary Classification: } & P(r_{ui} = 1|p_u, q_i) = F_\theta(p_u, q_i, \theta) \\ \text{Rating Regression: } & \hat{r}_{ui} = F_\theta(p_u, q_i, \theta) \end{aligned} \tag{5.1}$$

5.1 The Automatic Recommendation Task

Next, we will present the notation associated with our recommendation task. Each task T_u generates a set of training items and a set of test items that user u has already rated. Every user-item pair has a rating r_{ui}. For each task T_u, we want to infer the ratings of each of the items in the test set for user u, using a model trained on the items in the training set.

Out of all the meta-learning algorithms introduced in Chapter 3, we decided to adopt the Model-Agnostic Meta-Learning [36] (MAML) framework, which uses stochastic gradient descent for rapid test-time predictions. When adapting the recommendation model to task T_u, the model's parameters θ become ϕ_u. Following the process of MAML, we compute ϕ_u using the gradient updates based on the loss function value for each task T_u.

5.2 The Local Updates

We want to train the model $F_\theta(p_u, q_i, \theta)$ to generalize across all tasks, such that it can perform well on a new task T_u for a new user using only a few gradient updates. We sample a batch of tasks B from the user set U and then update the recommendation model $F_\theta(p_u, q_i, \theta)$ for each task T_u as follows:

$$\phi_u = \theta - \alpha \nabla_\theta L_{T_u}(F_\theta) \tag{5.2}$$

where the (scalar) step size α is a hyper-parameter that is pre-defined. For binary classification, the loss function L_{T_u} is the binary cross-entropy (or negative Bernoulli log likelihood) which takes the following form:

$$L_{T_u}(F_\theta) = \sum_{T_u \sim B} [r_{ui} \log F_\theta + (1 - r_{ui}) \log(1 - F_\theta)] \tag{5.3}$$

For rating regression, the loss function L_{T_u} is a mean-squared error loss (or negative Gaussian log likelihood with fixed standard deviation of 1) with the following form:

$$L_{T_u}(F_\theta) = \frac{1}{2} \sum_{T_u \sim B} (r_{ui} - \hat{r}_{ui})^2. \tag{5.4}$$

5.3 Global Update

In the global update of our meta-learning process, we update the parameters θ based on the test loss for all tasks T_u. This meta-objective looks like this:

$$\min_\phi \sum_{T_u} L_{T_u}(F_{\phi_u}) = \sum_{T_u} L_{T_u}(F_{\theta - \alpha \nabla_\theta L_{T_u}(F_\theta)}). \tag{5.5}$$

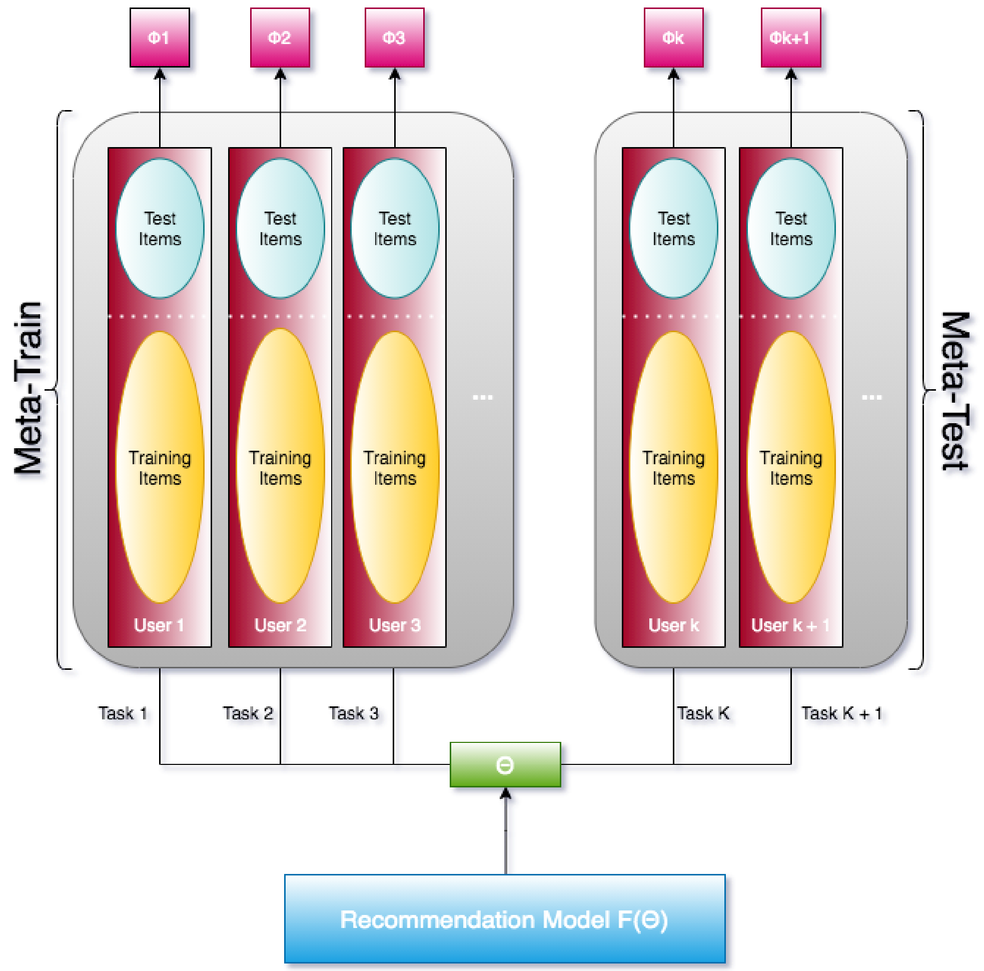


Figure 5.1: Meta-Learning scheme for MetaRec.

We also use stochastic gradient descent to globally update the parameters θ:

$$\theta = \theta - \beta \nabla_\theta \sum_{T_u \sim B} L_{T_u}(F_{\phi_u}) \tag{5.6}$$

where the (scalar) step size β is a hyper-parameter that is pre-defined.

In Figure 5.1, we graphically depict our proposed Meta-Rec learning process. In the next section, we will detail the dynamics of the concrete recommendation models used as baselines for (benchmarking) comparison for the Meta-Rec process itself.

Chapter 6

Experiments

6.1 The Dataset

In the experiments conducted for this thesis, we used the **MovieLens1M Dataset**, a famous dataset within the recommendation systems research community [51]. The database contains $1,000,209$ anonymous ratings of $3,883$ movies made by $6,040$ MovieLens users who joined MovieLens in 2000. There are three files in the dataset: ratings, users, and movies (see Table 6.1):

- **Ratings**: There are four columns in this file - UserID, MovieID, Rating, and Timestemp. UserIDs range between 1 and 6040. MovieIDs range between 1 and 3952. Ratings are made on a 5-star scale (whole-star ratings only). Timestamp is represented in seconds. Each user has at least 20 ratings.

- **Users**: There are five columns in this file - UserID, Gender, Age, Occupation, & Zipcodes. All demographic information is provided voluntarily by the users (not checked for accuracy). Only users who have provided some demographic information are included. Gender is denoted by “M” for male and “F” for female. Occupation is chosen from 21 different choices: ”other”, ”academic / educator”, ”artist”, ”clerical / admin”, ”college/grad student”, ”customer service”, ”doctor/health care”, ”executive / managerial”, ”farmer”, ”homemaker”, ”K-12 student”, ”lawyer”, ”programmer”, ”retired”, ”sales / marketing”, ”scientist”, ”self-employed”, ”technician / engineer”, ”tradesman / craftsman”, ”unemployed”, and ”writer.” Age is chosen from 7 different ranges: ”Under 18”, ”18-24”, ”25-34”, ”35-44”, ”45-49”, ”50-55”, & ”56+”.

Table 6.1: MovieLens1M Dataset Characteristics

Attributes	Users	Movies	Ratings
Total	6,040	3,883	1,000,209
Min Rating	20	1	1
Max Rating	2,314	3,428	5
Average Rating	165.6	269.9	3.58

- **Movies**: There are three columns in this file - MovieID, Title, and Genres. Titles are identical to those on IMDB (including year of release). Genres are pipe-separated and are selected from 18 different genres: "Action", "Adventure", "Animation", "Children's", "Comedy", "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror", "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", & "Western."

In the appendix, we explored and visualized several important attributes contained in the dataset (see the relevant figures in Section A.1), such as movie titles, movie genres, rating distribution, etc.

6.2 Matrix Factorization Experiments

In this section, we look at Matrix Factorization (MF) in detail. As explained briefly in Section 4.1, MF is essentially a linear, latent factor model. First, we examine MF's dynamics closer and brainstorm ways to strengthen its capacity by adding more factors: biases, side information, temporal information, mixture of tastes, etc. We also previously discussed these baseline models in our published article [77]. We then introduce our method, MetaRec-MF, that uses MF as the base (sub-)model $F_\theta(p_u, q_i)$ under the MetaRec framework introduced in Chapter 5. We will provide empirical results to evaluate our proposed approach and extract insights.

6.2.1 Standard Matrix Factorization

The dynamics of a standard matrix factorization for automatic recommendation can be posed as follows [71]:

$$r_{ui} = p_u \cdot q_i^T \tag{6.1}$$

where we observe that the model predicts the response of a user u to an item i with only a single term: the dot product between a k-dimensional

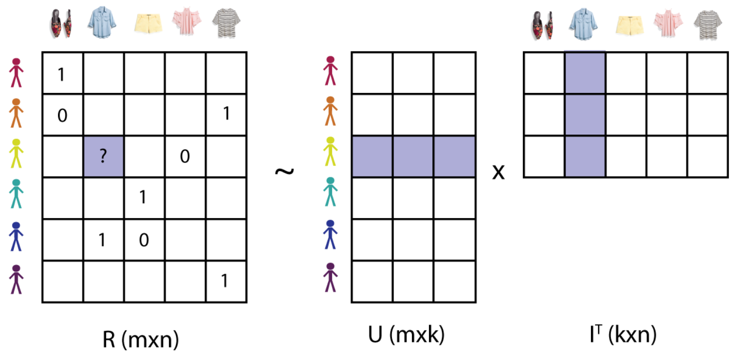


Figure 6.1: A graphical depiction of MF (diagram taken from [18]).

embedding vector of the user p_u and the item q_i (*how well-matched is this user to this particular item?*). The framing of Equation 6.1 is equivalent to a decomposition of a bias-corrected response matrix R (of shape $(m \times n)$) into two far lower-dimensional matrices U (of shape $(m \times k)$) and I^T (of shape $(k \times n)$). Unlike many machine learning algorithms, the MF process learns the implicit latent factors that compose the embeddings from raw responses instead of declaring them upfront as explicit features. Since we have not observed every user-item interaction a priori, R will be sparse, as shown in Figure 6.1.

MF assumes that there are only k latent factors that contribute to the preference of any given user for a particular item, where k is chosen to recover best the information in our observed matrix R. This assumption reduces the complexity of this problem from $O(m \times n)$ to the much more tractable $O(k \times (m+n))$ (though at the expense of convexity – meaning we will solving a non-convex optimization problem).

We learn the MF system defined in Equation 6.1 by fitting the model to previously observed ratings. However, we also want to generalize those previous ratings in order to predict unknown ratings. Thus, we mitigate overfitting to observed data by adding an L2 regularization penalty to each element (of R) and optimize the learned parameters simultaneously with stochastic gradient descent:

$$\min \sum_{(u,i)\in T} (r_{ui} - p_u \cdot q_i^T)^2 + \lambda(||p_u||^2 + ||q_i||^2) \tag{6.2}$$

where T is the set of user-item pairs with known ratings and λ is the regularization rate.

6.2.2 Matrix Factorization with Biases

One benefit of the matrix factorization approach to collaborative filtering is its flexibility when dealing with various data types and other application-specific requirements. Recall that Equation 6.1 attempts to capture the interactions between users and items that produce different rating values. However, much of the observed variation in the rating values are due to effects associated with either users or items, known as **biases**, independent of any interactions. The intuition behind this is that some users give higher ratings than others, and some items received higher ratings than others systematically.

To account for this intuition in the MF model dynamics, we modify Equation 6.1 as follows [71]:

$$r_{ui} = b + \omega_u + \omega_i + p_u \cdot q_i^T \tag{6.3}$$

where we add the global bias b (*the average rating for all items*), the bias of the user ω_u (*how much do this user like things generally?*), and the bias of the item ω_i (*how popular is this item generally?*). The loss function from equation 6.2 is updated as follows:

$$\min \sum_{(u,i)\in T} (r_{ui} - b - \omega_u - \omega_i - p_u \cdot q_i^T)^2 + \lambda(||p_u||^2 + ||q_i||^2 + \omega_u^2 + \omega_i^2) \tag{6.4}$$

6.2.3 Matrix Factorization with Side Features

A common challenge in collaborative filtering is the cold start problem due to its inability to address new items and new users. Furthermore, many users might supply very few ratings, making the user-item interaction matrix very sparse. A way to alleviate this problem is to utilize more user information, also called **side features**. Side features can be user demographics as well as implicit feedback.

We have two options for integrating potential side features: adding them as an additional bias (*artists like movies more than other occupations*) and adding them as a vector (*realtors love real estate shows*). The matrix factorization model should integrate all signal sources to create a richer user representation [71], the dynamics of which are:

$$r_{ui} = b + d_o + \omega_u + \omega_i + (p_u + t_o) \cdot q_i^T \tag{6.5}$$

For this model, we add the bias for occupation d_o (if occupation changes the "like rate") and the vector for occupation t_o (if occupation changes depending on the item). The loss function from equation 6.4 is updated as follows:

$$\begin{aligned} \min \sum_{(u,i)\in T} & (r_{ui} - b - \omega_u - \omega_i - d_o - (p_u + t_o) \cdot q_i^T)^2 \\ & + \lambda(||p_u||^2 + ||q_i||^2 + ||t_o||^2 + \omega_u^2 + \omega_i^2) \end{aligned} \tag{6.6}$$

6.2.4 Matrix Factorization with Temporal Features

In the real-world, item popularity and user preferences change frequently. Therefore, we should take into consideration the temporal effects reflecting the dynamic nature of user-item interactions. To accomplish this, we can add a **temporal term** that affects user preferences and, therefore, the interaction between users and items. Specifically, we experiment with a new dynamic prediction rule based on a rating at time t [71]. The dynamics of this system are:

$$r_{ui} = b + \omega_u + \omega_i + p_u \cdot q_i^T + m_u \cdot n_t \tag{6.7}$$

where we integrated the variables n_t (a vector of time series) and m_u (a vector of user with respect to time) to make the MF process evolve with time. Here, the latent temporal vector $m_u \cdot n_t$ encodes a user mixture of time series that captures user-time interactions. We also enforced that nearby time steps have similar components by minimizing $|n_t - n_{t-1}|$ in the regularization term. The loss function then looks like this:

$$\begin{aligned} \min \sum_{(u,i)\in T} & (r_{ui} - b - \omega_u - \omega_i - p_u \cdot q_i^T - m_u \cdot n_t) \\ + \lambda(||p_u||^2 + ||q_i||^2 & + ||m_u||^2 + |||n_t - n_{t-1}|||^2 + \omega_u^2 + \omega_i^2) \end{aligned} \tag{6.8}$$

6.2.5 Factorization Machines

One of the more powerful techniques for the recommendation system is called the **Factorization Machine** (FM) [117], which has a robust and expressive capacity to generalize matrix factorization methods. In many applications, we have plenty of item metadata that can be used to make better predictions. This is one of the benefits of using FMs with feature-rich datasets – the FM paradigm provides a natural way to include extra features during learning, using a dimensionality parameter k to model higher-order interactions. A second-order FM model suffices for sparse datasets since there is not enough information to estimate more complex interactions.

Formally, the dynamics of a second-order FM are as follows:

$$r_{ui} = b + \sum_{i=1}^{n} \omega_i \cdot x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < v_i^T, v_j > x_i \cdot x_j \tag{6.9}$$

where v_i and v_j represent two k-dimensional latent embeddings for feature i and feature j. $< v_i, v_j >$ represents their inner product that models the interaction between feature i and feature j. More specifically, v_i and v_j (within $V \in \mathbb{R}^{n \times k}$) denote the i-th variable and k-th variable with k factors (where k is the fixed dimensionality parameter). $\omega_i \in \mathbb{R}^n$ models the interaction of the i-th feature to the target. x_i and x_j denote the corresponding weights of feature i and feature j, so that only interactions of non-zero features are considered. $\sum_{i=1}^{n} \sum_{j=i+1}^{n}$ denote the sum over all pairs of features. The corresponding loss function is:

$$\begin{split} \min \sum_{(u,i) \in T} (r_{ui} - b - \sum_{i=1}^{n} \omega_i \cdot x_i - \sum_{i=1}^{n} \sum_{j=i+1}^{n} < v_i^T, v_j > x_i \cdot x_j) \\ + \lambda (\sum_{i=1}^{n} ||v_i||^2 + \sum_{i=1}^{n} \omega_i^2) \end{split} \tag{6.10}$$

As confirmed in [117], the FM introduces higher-order interactions in terms of latent vectors affected by categorical or tag data. This means that the FM model goes beyond co-occurrences to find more robust relationships between each feature's latent representations.

6.2.6 Matrix Factorization with a Mixture of Tastes

The techniques presented above implicitly treat user tastes as uni-modal, i.e., a single latent vector. This may lead to a lack of nuance when representing the user, where a dominant taste may overpower more niche ones. In addition, this may reduce the quality of item representations, decreasing the separation in the embedding space between groups of items belonging to multiple tastes/genres. [73] represented users as **mixtures of several distinct tastes** (similar in spirit to a Gaussian mixture model for data density estimation), represented by different taste vectors. Each taste vector is used alongside an attention vector, which describes the confidence level at describing an item. The dynamics of the mixture-of-taste model is formally:

$$r_{ui} = \sigma(A_u \cdot q_i^T) \cdot (U_u \cdot q_i^T) + b + \omega_u + \omega_i \tag{6.11}$$

where $U_u \in \mathbb{R}^{m \times k}$ matrix denotes the m tastes of user u, $A_u \in \mathbb{R}^{m \times k}$ matrix denotes the affinities of each taste from U_u for particular items, and $\sigma(x_i) = \frac{\exp^{x_i}}{\sum_j \exp^{x_j}}$ is the soft-max activation function. Then $\sigma(A_u \cdot q_i)$ gives the mixture probabilities and $(U_u \cdot q_i)$ gives the recommendation scores for each mixture component. The corresponding loss function is:

$$\begin{aligned} \min \sum_{(u,i) \in T} & (r_{ui} - b - \omega_u - \omega_i - \sigma(A_u \cdot q_i^T) \cdot (U_u \cdot q_i^T))^2 \\ & + \lambda(||A_u||^2 + ||U_u||^2 + ||q_i||^2 + \omega_u^2 + \omega_i^2) \end{aligned} \tag{6.12}$$

6.2.7 Variational Matrix Factorization

The last variant of matrix factorization that we experimented with is called variational matrix factorization [66, 111]. While all of the models above focus on optimizing a point estimate of the model parameters, **variational** inference focuses on optimizing a posterior (distribution). Loosely speaking, this expresses a spectrum of model configurations that are consistent with the data. Variational methods can provide a (principled) alternative form of regularization, measure what a model “does not know”, and reveal novel ways of grouping (input) data.

We generalize the MF dynamics in Equation 6.3 variational by replacing the point estimates with samples from a distribution. Formally, this means that the variational MF model is:

$$r_{ui} = b + \omega_u + \omega_i + GS(\mu_u, v_u) \cdot GS(\mu_i, v_i) \tag{6.13}$$

where μ_u and v_u denote the latent embeddings for the mean and variance values of users, and μ_i and v_i denote the latent embeddings for the mean and variance values of items. $GS(\mu, v)$ for mean μ and variance v denotes the Gaussian-Sampling Distribution, defined as follows (where $\mathcal{N}(0, 1)$ is the Gaussian noise, which is a normal distribution with mean 0 and standard deviation 1):

$$GS(\mu, v) = \mu + \mathcal{N}(0, 1) \cdot \sqrt{v} \tag{6.14}$$

6.2.8 MetaRec Matrix Factorization

For MetaRec-MF, our base model is a matrix factorization model with biases defined as follows:

$$\hat{r}_{ui} = F(u, i | p_u, q_i, \theta) = p_u \cdot q_i + b + w_u + w_i \tag{6.15}$$

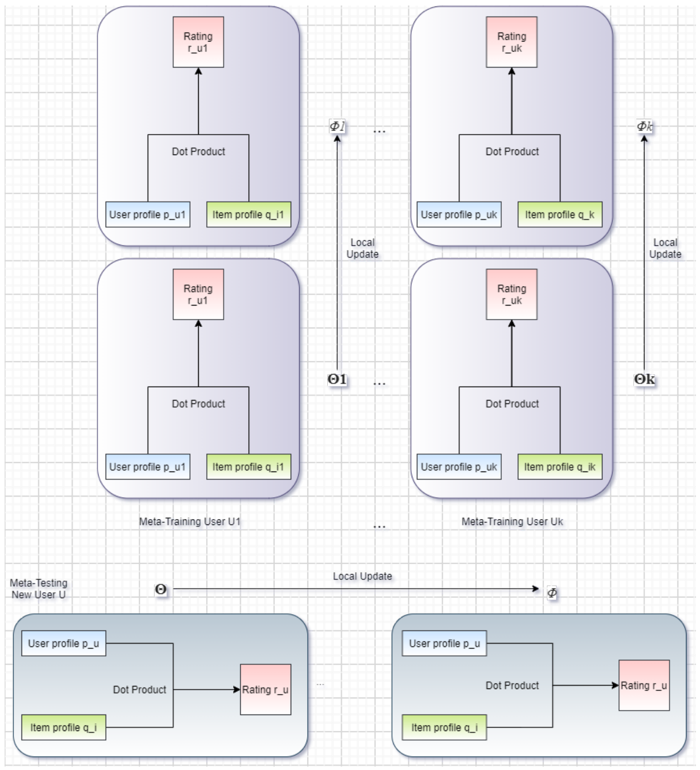


Figure 6.2: MetaRec-MF uses a matrix factorization base model and trains it according to our MAML process defined in Chapter 5.

where the k-dimensional user embedding vector is denoted by p_u, the k-dimensional item embedding vector is denoted by q_i, and the bias involved in the overall average rating is denoted by b. The parameters w_i and w_u indicate the observed deviations of item i (item bias) and user u (user bias) from the average, respectively. $\theta = \{w_u, w_i, b\}$, therefore, denotes the model parameters.

The loss function for training our MF process is defined as follows:

$$L(\theta) = \frac{1}{|D_{train}|} \sum_{r_{u,i} \in D_{train}} (r_{u,i} - \hat{r}_{u,i})^2 + \lambda \frac{1}{2} ||\theta||_2^2 \tag{6.16}$$

where $|\circ|$ denotes cardinality (how many data points are in D_{train}), θ denotes the model parameters, D_{train} denotes the training data, and λ denotes the weight of the L2 regularization term.

Formally, our proposed MetaRec-MF consists of two training phases:

Algorithm 1: MAML Algorithm for MetaRec-MF

```
Input  : Trainable global parameters $\theta$, user set $U$, local and global
         hyper-parameters $\alpha, \beta$
Output: Learned global parameters $\theta$
1 Initialize $\theta$ randomly;
2 while not converged do
3     Sample a batch of users $B$ from $U$;
4     for user $u \in B$ do
5         Evaluate the gradient $\nabla_\theta L_u(F_\theta)$;
6         Perform the local update $\phi_u \leftarrow \theta - \alpha \nabla_\theta L_u(F_\theta)$;
7     end
8     Perform global update $\theta \leftarrow \theta - \beta \sum_u L_u(F_{\phi_u})$
9 end
```

- **Local updates**: We randomly initialize the matrix factorization's model parameter vector θ. Then, we train it and calculate the loss $L_{train}(\theta)$ from Equation 6.16 for the training test set D_{train} for each batch of users. Finally, we locally update the parameter θ according to the loss.
- **Global update**: We want to identify a parameter vector θ that quickly generalizes to new tasks, which is to recommend items to new users. Each batch of users have their task-specific parameters ϕ_u, so we traverse all users and calculate the loss $L_{test}(\phi)$ from Equation 6.16 on the test set D_{test}. Finally, we sum up the losses and update the global parameter space θ via stochastic gradient descent.

This training procedure is summarized for reference in Algorithm 1. An illustration of the MetaRec-MF process is also included in Figure 6.2.

6.2.9 Experimental Results

For our experiments on MovieLens1M, we randomly separate user data into 75% training and 25% test sets. We train the models using ratings of the training sample users and evaluate the models on the ratings of the test set users. To evaluate the performance of the MF methods described earlier, we use two popular metrics – the mean absolute error (MAE) and the mean squared error (MSE). Given the true rating $r_{u,i}$ and the predicted rating $\hat{r}_{u,i}$

Table 6.2: Matrix factorization experiments (lower MAE & MSE values are better).

Models	MAE	MSE	Training Time
MF	0.703	0.817	**6m5s**
MF-Bias	0.694	0.790	11m38s
MF-Side	0.691	**0.784**	13m34s
MF-Temporal	0.689	0.793	18m51s
FM	0.712	0.823	**3m40s**
MF-Mixture	**0.687**	0.788	13m44s
Variational-MF	0.707	0.839	16m51s
MetaRec-MF (Ours)	**0.687**	**0.760**	12m45s

on the test set D_{test}. Formally, we define these metrics as follows:

$$\begin{aligned} MAE &= \frac{1}{|D_{test}|} \sum_{r_{u,i} \in D_{test}} |r_{u,i} - \hat{r}_{u,i}| \\ MSE &= \frac{1}{2|D_{test}|} \sum_{r_{u,i} \in D_{test}} (r_{u,i} - \hat{r}_{u,i})^2 \end{aligned} \tag{6.17}$$

Table 6.2 lists the MAE and MSE performance of all the Matrix Factorization methods in the test set after 50 training epochs. The implementation details are provided in the appendix (Section A.2). Our main observations are as follows:

- MetaRec-MF slightly outperforms the other models on both MAE and MSE metrics (the lower the values, the better the performance). This proves the advantage of meta-optimization in this learning setting for collaborative filtering systems.
- The training time for MetaRec-MF is slightly higher than MF-Bias, its non-MAML equivalence. This hints at a potential tradeoff between performance and compute cost.
- The FM model had the fastest training time, showcasing the benefits of learning higher-order interactions.
- Adding more features to the MF equation improved performance but led to a longer training time. For example, MF with side features took

longer to train than vanilla MF, and MF with both side and temporal features took longer to train than MF with just side features.

Considering that the base model for MetaRec-MF is a just matrix factorization one with only biases, it would be interesting to see if adding more features, replace the MF with a factorization machine, or using variational learning could potentially lead to better performance or faster training time. We leave this for future work.

6.3 Multi-Layer Perceptron Experiments

In this section, we explore the adaptation of the multi-layer perceptron (MLP) to the recommendation task. As explained in Section 4.2, the MLP can handle non-linear interactions between users and items to predict the user preferences for the items. Particularly, we introduce five cutting-edge models and rigorously walk through their mathematical formulations: WideDeep, DeepFM, xDeepFM, NeuralFM, and NeuralCF. We also previously discussed these baseline models in our published article [75]. We then introduce our method, MetaRec-MLP, which uses an MLP as its base model $F_\theta(p_u, q_i)$ to drive the MetaRec process introduced in Chapter 5. Empirical results support the proposed approach and provide further insights.

6.3.1 Wide and Deep Learning (WideDeep)

[27] proposed a framework that combines the strength of wide linear models and deep network models to address memorization and generalization issues in recommendation systems. This framework has been put into production as the back-end system for Google Play, a massive-scale commercial application store. As shown in Figure 6.3, the wide learning component is a single-layer perceptron that memorizes sparse feature interactions through cross-product feature transformations. The deep component is a multi-layer perceptron that generalizes to previously unseen feature interactions via low-dimensional embeddings.

Formally, the wide learning component of the WideDeep system is defined as follows:

$$y = W_{wide}^T \cdot x + b \tag{6.18}$$

where y is the prediction, x is a vector of features, W is a vector of model parameters, and b is the bias. The feature set includes both raw inputs and transformed inputs created with a cross-product transformation to capture the correlation(s) between features. The deep learning component can be

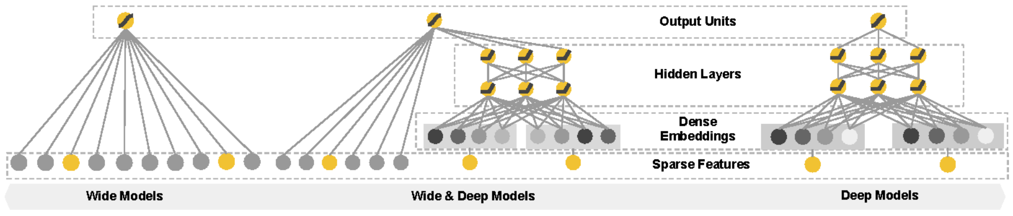


Figure 6.3: The spectrum of “wide and deep models” proposed in [27].

decomposed into a set of hidden layers, where each hidden layer performs the following computation:

$$a_{l+1} = f(W_{deep}^{T} \cdot a_l + b_l) \tag{6.19}$$

where l is the layer number, f is the activation function, a_l is the vector of (input) activation values, b_l is the vector of biases, and W_l is the vector of model weights at the l-th layer.

The full WideDeep model is obtained by fusing together the two component models described above:

$$P(Y = 1|X) = \sigma(W_{wide}^{T} \cdot x + W_{deep}^{T} \cdot a_{last} + b) \tag{6.20}$$

where Y is the binary class label, $\sigma(.)$ is the sigmoid activation function, W_{wide} is the vector of weights for the wide learning component, W_{deep} is the vector of weights applied on the final activation a_{last}, and b is the bias term.

6.3.2 The Deep Factorization Machine (DeepFM)

As an extension of the wide and deep approach is the so-called deep factorization machine [48], which is an end-to-end model that seamlessly integrates an FM (the wide component) and an MLP (the deep component). Compared to the WideDeep model, DeepFM does not require tedious feature engineering. As depicted in Figure 6.4, the FM component utilizes addition and inner product operations to capture the linear and pairwise interactions between features. The MLP leverages non-linear activations and a deep hierarchical neural structure to model high-order interactions.

Formally, the inputs to the DeepFM are m-dimensional data records consisting of pairs (u, i) which are the identity and features of user and item, as well as a binary label y, which indicates user click behaviors ($y = 1$ means the user clicked the item, and $y = 0$ otherwise). The task here is to build a prediction model to estimate the probability that a user will click a specific app in a given context.

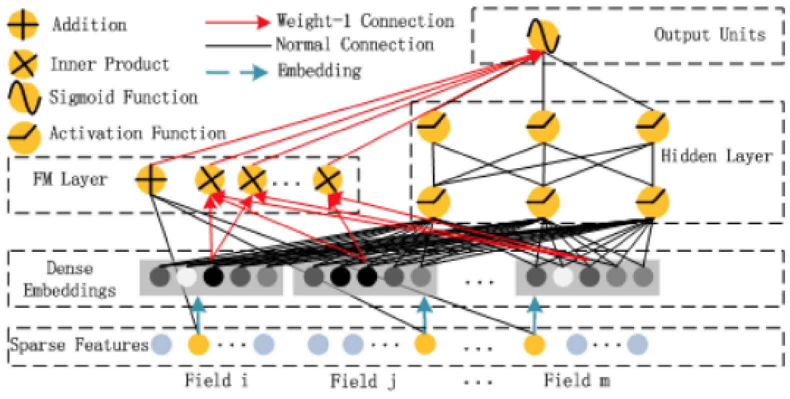


Figure 6.4: The wide and deep architecture of DeepFM in [48].

For any particular feature i, DeepFM uses a scalar w_i to weigh i's first-order importance, and a latent vector V_i to measure i's impact/interaction with other features. DeepFM then feeds V_i into the wide component in order to model the second-order feature interactions and next into the deep component to model even higher-order feature interactions. All parameters (w_i, V_i, as well as the network parameters) are trained jointly under the full prediction model:

$$\hat{y} = \sigma(y_{FM} + y_{DNN}) \tag{6.21}$$

where $\hat{y}$ is the predicted click-through rate between 0 and 1, y_{FM} is the output of the wide FM component, y_{DNN} is the output of the MLP component, and $\sigma(.)$ is the sigmoid activation function.

In the wide component, the FM measures both the linear (first-order) interactions among features and the pairwise (second-order) feature interactions (the inner product of latent feature vectors). This process can capture the second-order feature interactions quite effectively when dealing with sparse datasets. The output of the FM module is the sum of an addition unit and several inner product units:

$$y_{FM} =< w, x > + \sum_{j_1=1} \sum_{j_2=j_1+1} < V_i, V_j > x_{j_1} \cdot x_{j_2} \tag{6.22}$$

with given features i and j. The addition unit (the first term) captures the importance of first-order features, and the inner product units (the second term) capture the impact of second-order feature interactions. In the deep component, the output of the MLP is simply:

$$y_{DNN} = \sigma(W^{|H|+1} \cdot a^H + b^{|H|+1}) \tag{6.23}$$

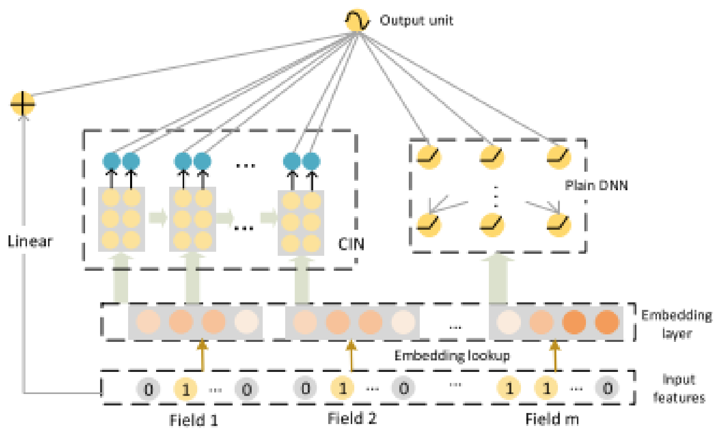


Figure 6.5: The architecture of xDeepFM in [85].

where σ is the sigmoid activation function, $|H|$ is the number of hidden layers, a is the vector output of the embedding layer, W is the vector of model weights, and b is the bias vector.

6.3.3 The Extreme Deep Factorization Machine (xDeepFM)

As an extension of the DeepFM, the extreme deep factorization machine (xDeepFM) [85] can jointly model explicit and implicit feature interactions. The explicit high-order feature interactions are learned with a "compressed interaction network" (CIN), while the implicit high-order feature infractions are learned with, again, an MLP. This model also requires no manual feature engineering and releases the experimenter from tedious feature engineering work. The CIN module applies interactions at a vector-wise level. Therefore, its complexity will not grow exponentially with the degree of interactions. Its structure is similar to that of a recurrent neural network, in which the output of the next hidden layer depends on the output of the previous hidden layer (of the last time step) and additional inputs.

xDeepFM combines the CIN with a simple MLP under the wide and deep learning framework (described earlier). This model captures both low-order and high-order feature interactions as well as both implicit and explicit feature interactions. The xDeepFM architecture is shown in Figure 6.5. Formally, the resulting output unit of xDeepFM system is:

$$\hat{y} = \sigma(W_{linear}^T \cdot a + W_{MLP}^T \cdot x_{MLP} + W_{CIN}^T \cdot p_+ + b) \tag{6.24}$$

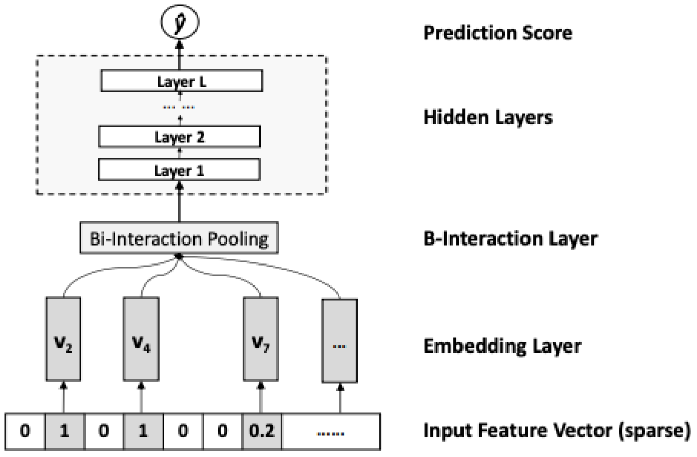


Figure 6.6: The architecture of the NeuralFM in [53].

where σ is the sigmoid activation function, a is the vector of raw features, x_{MLP} is the vector of outputs from the plain MLP, p_+ is the vector of outputs from the CIN module. W and b are the (trainable) weights and biases parameters, respectively.

The loss function for xDeepFM is the log loss with L2 regularization as seen below:

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i) + \lambda||\Theta||^2 \tag{6.25}$$

where N is the size of the training data and Θ is the set of parameters from different components of xDeepFM.

6.3.4 The Neural Factorization Machine (NeuralFM)

Another parallel work that seamlessly integrates FMs and MLPs is the neural factorization machine (or NeuralFM) [53], which brings together the effectiveness of linear FMs with the strong representation ability of deep ANNs for sparse prediction problems. As shown in Figure 6.6, the NeuralFM architecture's key is an operation called bilinear-interaction pooling (BIP), which enables an ANN to learn feature interactions at a low level. The non-linear layers are stacked on top of the BIP layer, enabling effective modeling of non-linear feature interactions. In contrast to traditional deep learning methods that simply concatenate or average embedding vectors in the low level, thanks to the BIP layer, the NeuralFM can encode higher-order feature interactions, strongly encouraging the "deeper" layers (those closer to the top) to learn meaningful representations.

Formally, given a sparse input vector x, the model estimates a target as:

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i \cdot x_i + f(x) \tag{6.26}$$

where the first term denotes the global bias of features, the second term denotes the global bias of feature weights, and the third term $f(x)$ denotes an MLP that models feature interactions. The design of $f(x)$ consists of 4 layer components: an embedding layer, a bilinear-interaction layer, (several) hidden layers, and a final prediction layer. The embedding layer is a fully-connected layer that projects each feature to a dense vector representation:

$$V_x = x_1 \cdot v_1, x_2 \cdot v_2, \cdots, x_n \cdot v_n \tag{6.27}$$

where x_i is the input feature vector and v_i is the embedding vector for the i-th feature.

Then NeuralFM feeds the embedding set V_x into a BIP layer, which includes a pooling operation that converts a set of embedding vectors to one single vector:

$$f_{BI}(V_x) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (x_i \cdot v_i) \cdot (x_j \cdot v_j) \tag{6.28}$$

where $v_i \cdot x_j$ denotes the element-wise product of two vectors v_i and x_j. The output of this pooling is a k-dimension vector that encodes the second-order interactions between features in the embedding space. Above the BIP layer is a stack of fully-connected hidden layers, which learn the higher-order interactions between features:

$$\begin{aligned} z_1 &= \sigma_1(W_1 \cdot f_{BI}(V_x) + b_1), \\ z_2 &= \sigma_2(W_2 \cdot z_1 + b_2), \\ &\cdots \\ z_L &= \sigma_L(W_L \cdot z_{L-1} + b_L) \end{aligned} \tag{6.29}$$

where L is the number of hidden layers; W_L, b_L, and σ_L corresponds to the weight matrix, bias vector, and activation function for the l-th layer, respectively. The choice of activation functions could be the logistic link (sigmoid),

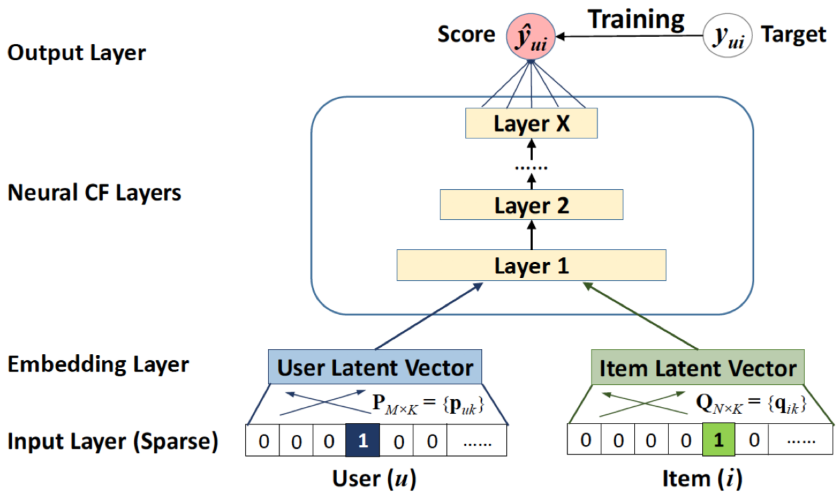


Figure 6.7: The architecture of the NeuralCF in [54].

the hyperbolic tangent (tanh), or the linear rectifier (ReLU) to learn higher-order, non-linear feature interactions. Lastly, the output vector of the last hidden layer z_L is transformed into the final prediction score (h^T denotes the neuron weights of the prediction layer):

$$f(x) = h^T \cdot z_L. \tag{6.30}$$

6.3.5 Neural Collaborative Filtering (NeuralCF)

The study [54] proposed using an MLP to model the user-item interaction r_{ui}, as shown in Figure 6.7, where the output of one layer serves as the input to the next one. Specifically, this entails the following:

- The bottom *input layer* consists of two feature vectors that describe user u and item i, which can be customized to support a wide range of ways of modeling users and items. In particular, the NeuralCF uses only the identity of a user and an item as the input feature, transforming it into a binarized sparse vector (through one-hot encoding). With such a generic feature representation for inputs, this framework can be easily adjusted to address the cold-start problem by using content features to represent users and items.

- Above the input layer is the *embedding layer*, a fully connected layer that projects the sparse representation to a dense vector. The obtained user/item embedding can be seen as the latent vector for user/item in the context of the latent factor model.

- The user embedding and item embedding are then fed into a multi-layer neural structure (termed *Neural Collaborative Filtering* layers) to map the latent vectors to prediction scores. Each layer of neural collaborative filtering layers can be customized to discover a specific type of latent structure of user-item interactions. The last hidden layer's dimension X controls the model's top-most representation complexity (smaller dimensions yield more compact latent codes).
- The final output layer returns the predicted rating r_{ui}.

The above framework can be summed up with the scoring function below:

$$r_{ui} = F(P^T \cdot v_u^U, Q^T \cdot v_i^I | P, Q, \theta_F) \tag{6.31}$$

where $P \in R^{U \times K}$ and $Q \in R^{I \times K}$ denote the latent factor matrix for users and items, respectively. v_u^U and v_i^I denote the side information associated with user and item features, respectively. θ_F denotes the model parameters of the interaction function F. Since the function F is defined as an MLP, it can be formulated in the following way:

$$F(P^T \cdot v_u^U, Q^T \cdot v_i^I) = \phi_{out}(\phi_X(...\phi_2(\phi_1(P^T \cdot v_u^U, Q^T \cdot v_i^I))...)) \tag{6.32}$$

where ϕ_{out} and ϕ_x denote the mapping function for the output layer and the x-th neural collaborative filtering layer, respectively.

6.3.6 The MetaRec Multi-Layer Perceptron (MetaRec-MLP)

Inspired by the NeuralCF framework presented above, the MetaRec-MLP employs a similar architecture to learn the user-item interaction r_{ui}, training synaptic parameters according to the MetaRec MAML process (Figure 6.8). Given user profile p_u and item profile q_i, we first construct an MLP (denoted as F) to learn the user embedding e_u and the item embedding e_i:

$$\begin{aligned} e_u &= F_{\theta_u}(p_u) \\ e_i &= F_{\theta_i}(q_i) \end{aligned} \tag{6.33}$$

where θ_u contains the user parameters and θ_i contains the item parameters that make up the MLP meant to learn embeddings.

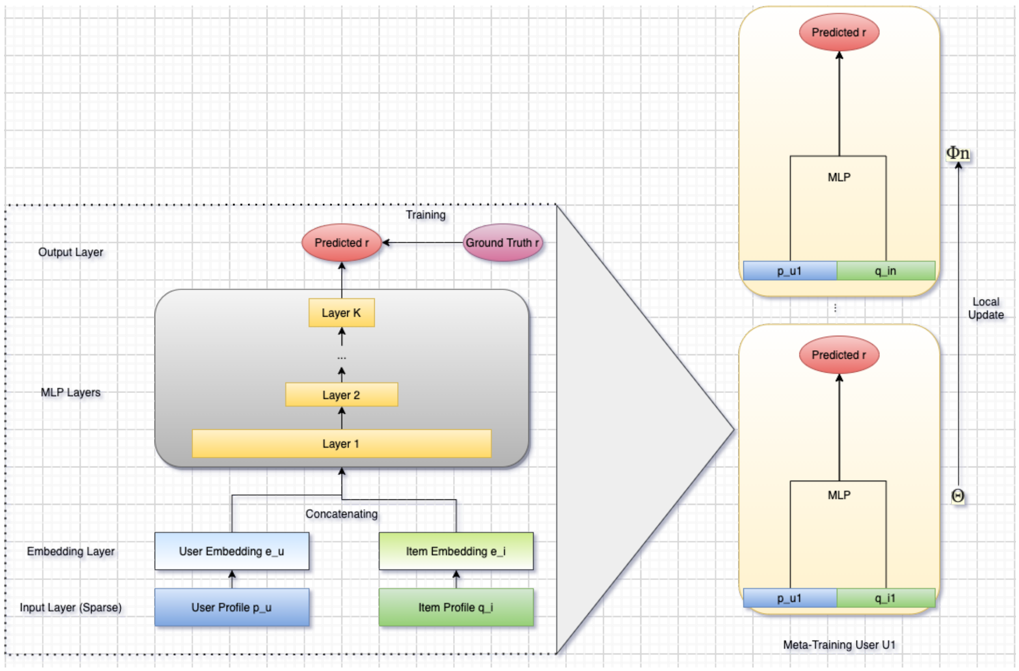


Figure 6.8: MetaRec-MLP uses an MLP base model, inspired by NeuralCF [54], and trains the full system acccording to the MAML process.

Next, we concatenate the two embeddings and feed the concatenation vector into another MLP to map the latent embeddings to prediction scores $\hat{r}_{ui}$:

$$\begin{aligned} x_0 &= [e_u, e_i], \\ x_1 &= \sigma(W_1 x_0 + b_1), \\ &\cdots \\ x_m &= \sigma(W_m x_{m-1} + b_m), \\ \hat{r}_{ui} &= \sigma(W_o x_m + b_o) \end{aligned} \tag{6.34}$$

where $[e_u, e_i]$ is the concatenation of user and item embeddings. W and b are the weight matrices and bias vectors for the layers (σ is the activation).

Equation 6.34 can be condensed into this form (for a particular user u):

$$\hat{r}_{ui} = F_{\phi_u}(e_u, e_i) \tag{6.35}$$

where ϕ_u contains the task-specific parameters of the (second) MLP that predicts the ratings given by user u.

Since we formulated the recommendation task as a binary-classification problem (explained below in Section 6.3.7), the loss function will be the binary cross-entropy (or Bernoulli log likelihood) loss, defined as follows:

$$L_u(F_{\theta,\phi_u}) = -\frac{1}{|B|}\sum_{i\in B}(r_{ui}\log \hat{r}_{ui}) + (1-r_{ui})\log(1-\hat{r}_{ui}) \tag{6.36}$$

for user u and item i, where B is a batch of items consumed by user u. The full training process is formally presented in Algorithm 2, which, note, is very similar to Algorithm 1. In essence, the process entails:

- **Local updates**: We randomly initialize the model parameters θ (equation 6.33) and the task-specific parameters ϕ (equation 6.35). Then, we randomly sample a batch of users B. Next, we back-propagate the loss from Equation 6.36 to locally update ϕ_u for each user $u \in B$.
- **Global update**: We globally update θ and ϕ based on the loss function from Equation 6.36 on the test set.

Algorithm 2: MAML Algorithm for MetaRec-MLP

```
Input : Trainable global parameters $\theta$, user set $U$, local and global
        hyper-parameters $\alpha, \beta$
Output: Learned global parameters $\theta$
1  Initialize $\theta$ randomly;
2  Initialize $\phi$ randomly;
3  while not converged do
4      Sample a batch of users $B$ from $U$;
5      for user $u \in B$ do
6          Evaluate the gradient $\nabla_{\phi_u} L_u(F_{\theta,\phi_u})$;
7          Perform the local update $\phi_u \leftarrow \phi_u - \alpha\nabla_{\phi_u} L_u(F_{\theta,\phi_u})$;
8      end
9      Perform global update $\theta \leftarrow \theta - \beta\sum_u \nabla_\theta L_u(F_{\theta,\phi_u})$ ;
10     Perform global update $\phi \leftarrow \phi - \beta\sum_u \nabla_\phi L_u(F_{\theta,\phi_u})$
11 end
```

6.3.7 Experimental Results

We evaluate our proposed MetaRec-MLP on the MovieLens1M benchmark by comparing it to the WideDeep, DeepFM, xDeepFM, NeuralFM, and NeuralCF baselines. Following the experiment setups of [27, 48, 53, 54, 85], we

Table 6.3: Results For Our Multi-Layer Perceptron Experiments (higher AUC values are better).

Models	Test AUC	Validation AUC	Training Time
WideDeep	**0.7991**	**0.7995**	1h12m15s
DeepFM	0.7915	0.7918	1h10m50s
xDeepFM	0.7429	0.7408	2h15m17s
NeuralFM	0.7589	0.7560	1h36m0s
NeuralCF	0.7668	0.7673	**54m15s**
MetaRec-MLP (Ours)	**0.8135**	**0.8127**	**1h05m3s**

pre-processed the dataset to work with implicit feedback for the task of top-N recommendation across various metrics. Specifically, we transformed the explicit ratings into implicit ratings, where each entry is labeled as 0 (less than 3) and 1 (greater than or equal to 3) indicating whether the user likes or does not like the item.

For the users, we randomly separate them into 80% training, 10% validation, and 10% test sets. We train the models using ratings of the training users and evaluate the models on the ratings of the validation users. To evaluate the performance of these models on the unseen ratings of the test users, we use a popular metric, Area Under the Receiver Operating Characteristics Curve (AUC), which is equivalent to measuring the probability that the system will be able to choose correctly between two items – one randomly selected from the set of bad items, and one randomly selected from the set of good items.

Table 6.3 reports the AUC measurements of all of the MLP methods on the validation and test sets after 100 training epochs. The implementation details are provided in the Appendix (see Section A.3). Our main observations are as follows:

- We can see that our proposed MetaRec-MLP outperforms other methods in terms of AUC on both test and validation sets. This can be attributed to the stronger capacity of meta-optimization approach in capturing user preferences. For example, compared with NeuralCF which uses a similar base recommendation algorithm, MeteRec-MLP shows a 6% lift in test AUC performance.

- There is a trade-off in performance and training time for MetaRec-MLP. Compared to NeuralCF, the training took longer. However, compared to the rest of the baseline methods, MetaRec-MLP took less time to run.

- Across the remaining methods, both WideDeep and DeepFM perform better than other baselines, justifying the power of the wide and deep architectural design.
- Surprisingly, xDeepFM, an extension of DeepFM, performed the worst (6% worse than DeepFM in terms of test AUC), even though it is designed to learn both explicit and implicit feature interactions between user and items. We reckon that this is a case of over-fitting, considering that it took the longest to train as well.

For possible improvements to our MetaRec-MLP, we believe that adding more hidden layers to the base MLP model and tuning hyper-parameters more carefully would yield further improvements in generalization ability (this is also left to future work).

6.4 Autoencoder Experiments

In this section, we explore the adaptation of the autoencoder (AE) to the recommendation task. As explained in Section 4.3, an AE can learn non-linear user-item interactions efficiently and encode complex abstractions into its hidden layers/representations. Particularly, we introduce four state-of-the-art models and examine their architectural design: CDAE, MultVAE, SVAE, and ESAE. We also previously discussed these baseline models in our published article [76]. We then introduce our method, MetaRec-AE, which uses AE as its base model $F_\theta(p_u, q_i)$ to instantiate our MetaRec framework (Chapter 5). Finally, we close by presenting empirical results.

6.4.1 Collaborative Denoising Auto-encoder

Collaborative Denoising Auto-encoder [166] (CDAE) is a one-hidden-layer ANN that applies a denoising autoencoder to the recommendation context. A denoising autoencoder (DAE) [157] is an extension of the standard autoencoder, where it is trained to reconstruct the input data from its partially corrupted version. Forcing the model to reconstruct corrupted inputs enables the DAE to learn more robust features. Figure 6.9 displays a sample structure of CDAE, which consists of 3 layers: the input, the hidden, and the output:

- There are a total of $I + 1$ nodes in the input layer. The first I nodes represent user preferences, and each node of these I nodes corresponds to an item. The last node is a user-specific node denoted by the red

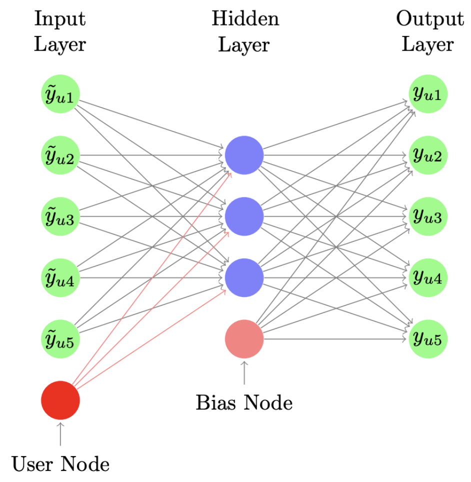


Figure 6.9: The basic CDAE architecture in [166].

node, which means different users have different nodes and associated weights.

- Here y_u is the I-dimensional feedback vector of user u on all the items in I. y_u is a sparse binary vector that only has non-zero values: $y_{ui} = 1$ if i has been rated by user u and $y_{ui} = 0$ otherwise.

- There are $K + 1$ nodes in the hidden layer. The blue K nodes are fully connected to the nodes of the input layer. The additional pink node in the hidden layer captures the bias effects.

- There are I nodes in the output layer, which are the reconstructed output of the input y_u. They are fully connected to the nodes in the hidden layer.

The corrupted input r_{corr} of CDAE is drawn from a conditional Gaussian distribution $p(r_{corr}|r)$. The reconstruction of r_{corr} is formulated as follows:

$$h(r_{corr}) = f(W_2 \cdot g(W_1 \cdot r_{corr} + V_u + b_1) + b_2) \tag{6.37}$$

where W_1 is the weight matrix corresponding to the encoder (going from the input layer to the hidden layer). W_2 is the weight matrix corresponding to the decoder (going from the hidden layer to the output layer). V_u is the weight matrix for the red user node. Both b_1 and b_2 are the bias vectors. We learn the parameters of CDAE by minimizing the average reconstruction error as follows:

$$\min_{W_1,W_2,V_u,b_1,b_2} = \frac{1}{M}\sum_{u=1}^{M} E_{p(r_{corr}|r)}[L(r_{corr},h(r_{corr}))] + \lambda \cdot \Omega \tag{6.38}$$

The loss function $L(r_{corr}, h(r_{corr}))$ in the Equation 6.38 can be square loss or logistic loss (and Ω is the regularization penalty – the CDAE uses the square L2 norm to control the model complexity). The system finally (1) applies stochastic gradient descent to learn the models parameters, and (2) adopts AdaGrad to automatically adapt the training step size (learning rate) during the learning procedure.

At inference time, CDAE takes a users existing preference set (without corruption) as input and recommends the items with the largest prediction values on the output layer to that user.

6.4.2 Multinomial Variational Auto-encoder

MultVAE [86] is a variational autoencoder [67, 118] for collaborative filtering based on implicit feedback (Figure 6.10). Additionally, the model uses the multinomial likelihood to estimate its parameters, which was better well-suited for modeling implicit feedback data than more popular likelihood functions such as the logistic and Gaussian.

In particular, the user-by-item interaction matrix is the rating/preference matrix $R \in \mathbb{R}^{U\times I}$. The lower case r_u is a bag-of-words vector with each item's preferences from user u. The generative process can be decomposed into the following probabilistic graphical model:

$$\begin{aligned} z_u &\sim \mathbb{N}(0, I_K), \\ \pi(z_u) &\sim \sigma\{F_\theta(z_u)\}, \\ r_u &\sim Mult(\mathbb{N}_u, \pi(z_u)). \end{aligned} \tag{6.39}$$

For each user u, the model samples a K-dimensional latent representation z_u from a standard Gaussian prior. Then it transforms z_u via a non-linear function F_θ to produce a probability distribution over I items $\pi(z_u)$. Here, F_θ is a multi-layer perceptron with parameters θ and σ is a softmax activation function. Given the matrix R, the bag-of-words vector r_u is sampled from a multinomial distribution with probability $\pi(z_u)$.

The log-likelihood for user u (conditioned on the latent representation) is:

$$\log p_\theta(r_u|z_u) = \sum_i r_{ui} \log \pi_i(z_u). \tag{6.40}$$

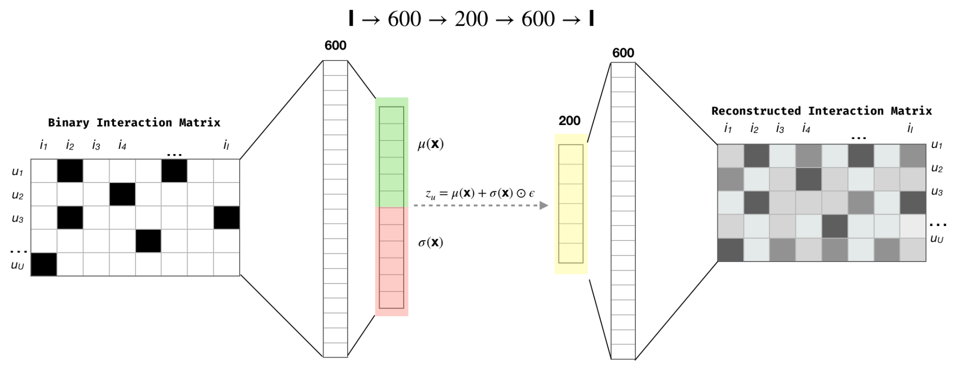

I → 600 → 200 → 600 → I
600
200
600
Binary Interaction Matrix
Reconstructed Interaction Matrix
$\mu(\mathbf{x})$
$\sigma(\mathbf{x})$
$z_u = \mu(\mathbf{x}) + \sigma(\mathbf{x}) \odot \epsilon$
i_1 i_2 i_3 i_4 ... i_I
u_1 u_2 u_3 ... u_U

Figure 6.10: The MultVAE architecture in [86].

This multinomial likelihood rewards the model for putting probability mass on the non-zero entries in r_u. However, considering that $\pi(z_u)$ must sum to 1, the items must compete for a limited budget of probability mass. Therefore, the model should instead assign more probability mass to items with a higher probability of being liked, making it suitable to achieve reliable performance in the top-N ranking evaluation metric of recommendation systems.

To learn the generative model in Equation 6.39, we want to estimate θ by approximating the intractable posterior distribution $p(z_u|r_u)$ using variational inference. This technique approximates the true intractable posterior with a simpler variatonal distribution $q(z_u)$, which, in our case, is a fully diagonal Gaussian distribution. The objective of variational inference is to optimize the free variational parameters $\{\mu_u, \sigma_u^2\}$ so that the Kullback-Leiber divergence $KL(q(z_u)||p(z_u|r_u))$ is minimized. The issue with variational inference is that the number of parameters to optimize $\{\mu_u, \sigma_u^2\}$ grows with the number of users and items in the dataset. The VAE resolves this issue by replacing the individual variational parameters with a data-dependent function (commonly called the "inference model") as follows:

$$g_\phi(r_u) = [\mu_\phi(r_u), \sigma_\phi(r_u)] \tag{6.41}$$

This function is parametrized by ϕ in which both $\mu_\phi(r_u)$ and $\sigma_\phi(r_u)$ are vectors with K dimensions. The variational distribution is then set to be:

$$q_\phi(z_u|r_u) = \mathbb{N}(\mu_\phi(r_u), diag\{\sigma_\phi^2(r_u)\}). \tag{6.42}$$

Using the input r_u, the inference model returns the corresponding variational parameters of variational distribution $q_\phi(z_u|r_u)$. When being optimized, this variational distribution approximates the intractable posterior $p_\phi(z_u|r_u)$.

To learn latent-variable models with variational inference, we need to optimize a lower-bound on the data's log marginal likelihood. The objective function to maximize for user u now becomes:

$$\log p(r_u;\theta) \geq \mathbb{E}_{q_\phi(z_u|r_u)}[\log p_\theta(r_u|z_u)] - KL(q_\phi(z_u|r_u)||p(z_u)) = \mathbb{L}(r_u;\theta,\phi) \tag{6.43}$$

The above objective is also called the **evidence lower bound** (ELBO). We can intuitively obtain an estimate of ELBO by sampling $z_u q_\phi$ and optimizing it with stochastic gradient ascent. However, we cannot easily differentiate ELBO to get the gradients with respect to ϕ. The reparametrization trick allows us to reformulate the model such that we can back-prop "through" the noise component of the VAE:

$$z_u = \mu_\phi(r_u) + \epsilon \cdot \sigma_\phi(r_u) \tag{6.44}$$

where we have isolated the stochasticity in the sampling process. Thus the gradient with respect to ϕ can be back-propagated through the sampled z_u.

From a different perspective, the first term of Equation 6.43 can be interpreted as a reconstruction error, while the second term of Equation 6.43 can be interpreted as regularization. Thus, Equation 6.43 can be extended with an additional parameter β to control the strength of the regularization:

$$\mathbb{L}_\beta(r_u;\theta,\phi) = \mathbb{E}_{q_\phi(z_u|r_u)}[\log p_\theta(r_u|z_u)] - \beta \cdot KL(q_\phi(z_u|r_u)||p(z_u)) \tag{6.45}$$

where the parameter β engages a tradeoff between how well the model fits the data and how close the approximate posterior stays to the prior during learning. We can tune β via KL annealing, a common heuristic used for training VAEs when there is concern that the model is under-utilized.

Given a users preference history r_u, the model ranks all items based on the un-normalized predicted multinomial probability $F_\theta(z)$. The latent representation z for x is simply the mean of the variational distribution $z = \mu_\phi(x)$. Note that, with autoencoders, we can effectively make predictions for users by evaluating two functions - the inference model (encoder) $g_\phi(.)$ and the generative model (decoder) $f_\theta(.)$. For latent factor collaborative filtering models such as matrix factorization and multi-layer perceptron, when giving the preference/rating history of a user not present in the training data, we can only obtain the latent factor for this user via some optimization process. This makes autoencoders extra useful when predictions need to be made cheaply with low latency and limited resources.

6.4.3 Sequential Variational Auto-encoder

Sequential Variational Auto-encoder (SDAE) [123] is an extension of Mult-VAE by exploring the rich information present in the past preference history. SDAE is a recurrent version of the MultVAE, where instead of processing a subset of the whole history regardless of temporal dependencies, the model processes/consumes a sequence subset through a recurrent neural network (RNN). The promise here is that handling temporal information can greatly improve the accuracy of the VAE recommender.

Given a set of users U, a set of items I, and the user-item preference matrix X with dimension $U \times I$. The principal difference between the MultVAE and the SVAE is that the SVAE considers precedence and temporal relationships within the matrix X. Specifically, the SVAE embodies the assumptions that:

- X induces a natural ordering relationship between items: $i <_u j$ has the meaning that $x_{u,i} > x_{u,j}$ in the rating matrix.
- It assumes the existence of timing information T, where the term $t_{u,i}$ represents the time when i was chosen by u. Then $i <_u j$ denotes that $t_{u,i} > t_{u,j}$.
- It also introduces a temporal mark in the elements of $x_u : x_{u(t)}$ represents the t-th item in I_u in the sorting induced by $<_u$, whereas $x_{u(1:t)}$ represents the sequence from $x_{u(1)}$ to $x_{u(t)}$.

Looking at the SVAE architecture, we can observe the recurrence relation occurs in the layer upon which $z_{u(t)}$ depends. The basic idea behind SVAE is that its latent variable should be able to express temporal dynamics, hence capturing causal dependencies among preferences in a users history.

The SVAE framework models temporal dependencies by conditioning each event on previous events. Given a sequence $x_{(1:T}$, its probability is:

$$P(x_{(1:T)}) = \prod_{t=0}^{T-1} P(x_{t+1}|x_{(1:)}). \tag{6.46}$$

This probability represents a recurrent relationship between $x_{(t+1}$ and $x_{(1:t)}$. Thus, the model can process each time-step separately.

Recall the generative process in Equation 6.39. We can now integrate the timestamp t and alter the process as follows:

$$\begin{aligned} z_{u(t)} &\sim \mathbb{N}(0, I_K), \\ \pi(z_{u(t)}) &\sim \sigma\{F_\theta(z_{u(t)})\}, \\ x_{u(t)} &\sim Mult(\mathbb{N}_u, \pi(z_{u(t)})). \end{aligned} \tag{6.47}$$

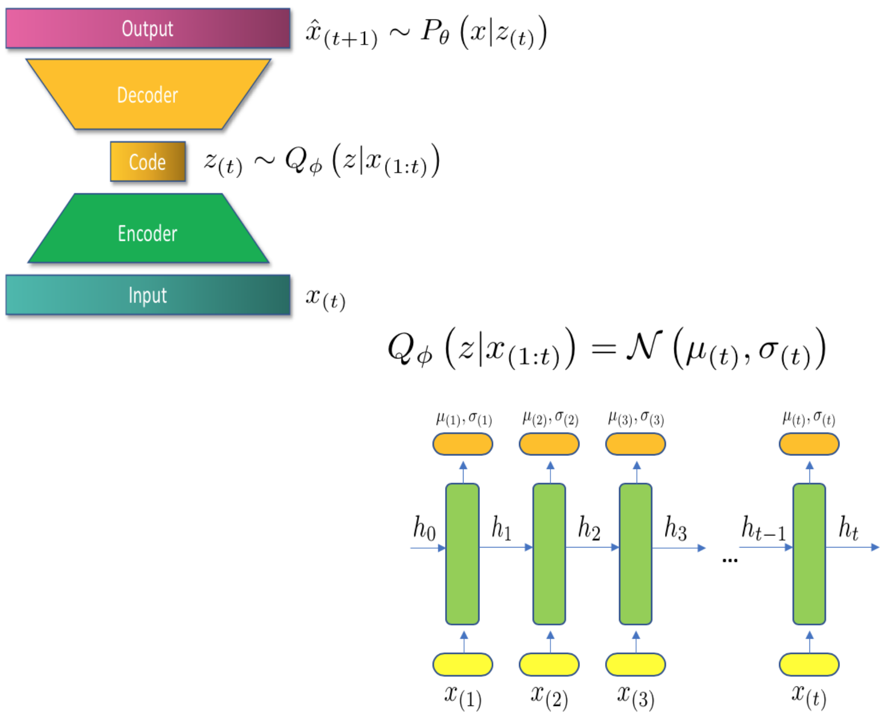

Output
$\hat{x}_{(t+1)} \sim P_\theta\left(x|z_{(t)}\right)$
Decoder
Code
$z_{(t)} \sim Q_\phi\left(z|x_{(1:t)}\right)$
Encoder
Input
$x_{(t)}$
$Q_\phi\left(z|x_{(1:t)}\right) = \mathcal{N}\left(\mu_{(t)}, \sigma_{(t)}\right)$
$\mu_{(1)}, \sigma_{(1)}$
$\mu_{(2)}, \sigma_{(2)}$
$\mu_{(3)}, \sigma_{(3)}$
$\mu_{(t)}, \sigma_{(t)}$
h_0
h_1
h_2
h_3
h_{t-1}
h_t
$x_{(1)}$
$x_{(2)}$
$x_{(3)}$
$x_{(t)}$

Figure 6.11: The sample SVAE architecture in [123].

The full, joint likelihood of the model is:

$$P(x_{u(1:T)}, z_{u(1:T)}) = \prod_t P(x_{u(t)}|z_{u(t)})P(z_{u(t)})) \tag{6.48}$$

while the posterior likelihood can be approximated with a (completely) factorized proposal distribution:

$$Q_\lambda(z_{u(1:T)}|x_{u(1:T)}) = \prod_t q_\lambda(z_{u(t)}|x_{(1:t-1)}) \tag{6.49}$$

where the right-hand side is a Gaussian distribution whose parameters μ and σ depend upon the current history $x_{u(1:t-1)}$, by means of a recurrent layer h_t:

$$\begin{aligned} \mu_\lambda(t), \sigma_\lambda(t) &= \phi_\lambda(h_t) \\ (h_t) &= RNN_\lambda(h_{t-1}, x_{u(t-1)}). \end{aligned} \tag{6.50}$$

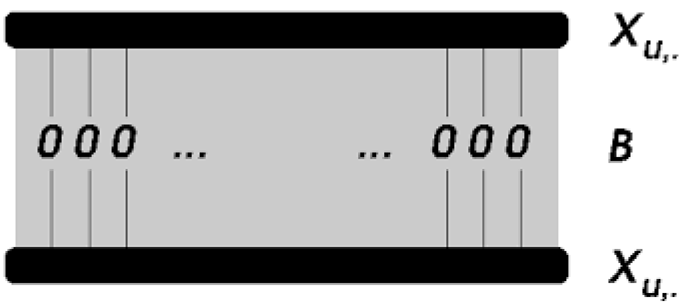


Figure 6.12: The item self-similarity layer in the ESAE architecture in [138].

Finally, the complete loss function that the SVAE optimizes is:

$$L(\phi, \lambda, X) = \sum_u \sum_{t=1}^{N_u} \{ \frac{1}{2} \sum_k (\sigma_{\lambda,k}(t) - 1 - \log \sigma_{\lambda,k}(t) + \mu_{\lambda,k}(t)^2) \\ - E_{\epsilon \sim N(0,1)}[\log P_\phi(x_{u(t)} | z_\lambda(\epsilon, t))] \}. \tag{6.51}$$

6.4.4 Embarrassingly Shallow Auto-encoder

The Embarrassingly Shallow Auto-encoder (ESAE) [138] is a fascinating model that we also want to bring into this discussion as it is quite relevant for the recommendation task. The motivation here is that, according to the author, *deep models with a large number of hidden layers typically do not obtain a notable improvement in ranking accuracy in collaborative filtering, compared to deep models with only one, two, or three hidden layers.* This is in stark contrast to other problem domains such as computer vision.

ESAE is a linear model **without** a hidden layer. The (binary) input vector X vector indicates which items a user has interacted with, and ESAEs objective is to predict the best items to recommend to that user in the output layer (as seen in the figure above). For implicit feedback, a value of 1 in X indicates that the user interacted with an item, while a value of 0 in X indicates that there is no observed interaction.

The item-item weight matrix B represents the parameters of ESAE. As observed in Figure 6.12, we omit an input item's self-similarity in the output layer so that the ESAE can generalize effectively during the reconstruction step. Thus, the diagonal of this weight-matrix B is constrained to 0 ($diag(B) = 0$). For an item j and a user u, we want to predict $S_{u,j}$, where $X_{u,\cdot}$ refers to row u and $B_{\cdot,j}$ refers to column j:

$$S_{u,j} = X_{u,\cdot} \cdot B_{\cdot,j}. \tag{6.52}$$

With respect to $diag(B) = 0$, ESAE has the following convex objective for learning the weights B:

$$\min_B ||X - X \cdot B||_F^2 + \lambda \cdot ||B||_F^2. \tag{6.53}$$

Note, for this convex objective, the following:

- $||.||$ denotes the Frobenius norm. This squared loss between the data X and the predicted scores $X \cdot B$ allows for a closed-form solution.
- The hyper-parameter λ is the L2-norm regularization of the weights B.
- The constraint of a zero diagonal helps avoid the trivial solution $B = I$ where I is the identity matrix.

Notably, ESAEs similarity matrix is based on the inverse of the given data matrix. As a result, the learned weights can also be negative, and thus the model can learn the **dissimilarities** between items (in addition to similarities). This proves to be essential when attempting to obtain good ranking accuracy. Furthermore, the data sparsity problem (there possibly exists only a small amount of data available for each user) does not affect the uncertainty in estimating weight matrix B, assuming that the number of users in the data matrix X is sufficiently large.

6.4.5 MetaRec Autoencoder

As observed throughout Section 6.4, we can train autoencoders in an unsupervised way to reproduce the input, here the ground-truth rating matrix. If we can draw samples of user-item interaction outputs given the inputs, we can use stochastic gradient descent (SGD) to train both the encoder and the decoder simultaneously. One notable limitation of the autoencoder-based approach (in an end-to-end training pipeline) is that training needs to happen from scratch for each new user. With MetaRec-AE, we will eliminate this limitation by incorporating the AE into our MetaRec's MAML process. Rather than training a static model for all users, our meta-learning process will find a common initialization vector that enables fast training on any user.

Let us quickly recall the rating-based collaborative filtering setting: we have m users, n items, and a partially-observed user-item rating matrix $R \in \mathbb{R}^{m \times n}$. For each user $u \in U = \{1 \cdots m\}$, his/her rating can be represented by a partially observed vector $r_{ui} = (R_{u1} \cdots R_{un}) \in \mathbb{R}^n$. The goal is to design an autoencoder that takes as input each partially observed r_{ui}, transforms it into a low-dimensional latent space, and reconstructs r_{ui} in the output space to predict missing ratings.

Formally, the autoencoder minimizes the objective function below for a set R of vectors in $\mathbb{R}^n$:

$$\frac{1}{m}\sum_{u=1}^{m} L(r_{ui} - recon(r_{ui};\theta)) \tag{6.54}$$

where $recon(r_{ui};\theta)$ is the reconstructed rating $r_{ui} \in \mathbb{R}^n$ for activation functions $f(.), g(.)$:

$$recon(r_{ui};\theta) = f(W \cdot g(Vr_{ui} + \mu) + b) \tag{6.55}$$

where $\theta = \{W, V, \mu, b\}$ denotes the transformations $W \in \mathbb{R}^{n\times k}, V \in \mathbb{R}^{k\times n}$ and the biases $\mu \in \mathbb{R}^k, b \in \mathbb{R}^n$. n is the dimension of the input (and output) layer (also the total number of items in the data), while k is the dimension of the hidden layer. Given the learned parameters $\hat{\theta}$, our predicted rating is:

$$\hat{r}_{ui} = (recon(r_{ui}, \hat{\theta})) \tag{6.56}$$

Algorithm 3: Autoencoder-Based Collaborative Filtering

```
Input  : user-based rating $r_{ui}$, training data $D_{train}$, step size
         hyper-parameter $\alpha$
Output: learned parameter vector $\theta$
1 initialize $\theta^{(0)}$;
2 $i = 0$;
3 while not converged do
4     draw a sample of users from $D_{train}$
5     update autoencoder with respect to that sample
6     $\theta^{(i+1)} \leftarrow \theta^{(i)} - \alpha\nabla L_u(\theta^{(i)})$
7     $i \leftarrow i + 1$
8 end
9 $\theta \leftarrow \theta(i)$
```

Since we formulated the recommendation task as a binary-classification problem (explained below in Section 6.4.6), MetaRec-AE minimizes the following (binary) cross-entropy loss :

$$L_u(\theta) = -\frac{1}{m}\sum_{u=1}^{m} ||(r_{ui}\log\hat{r}_{ui}) + (1 - r_{ui})\log(1 - \hat{r}_{ui})||_O^2 + \frac{\lambda}{2}\cdot(||W||_2^2 + ||V||_2^2) \tag{6.57}$$

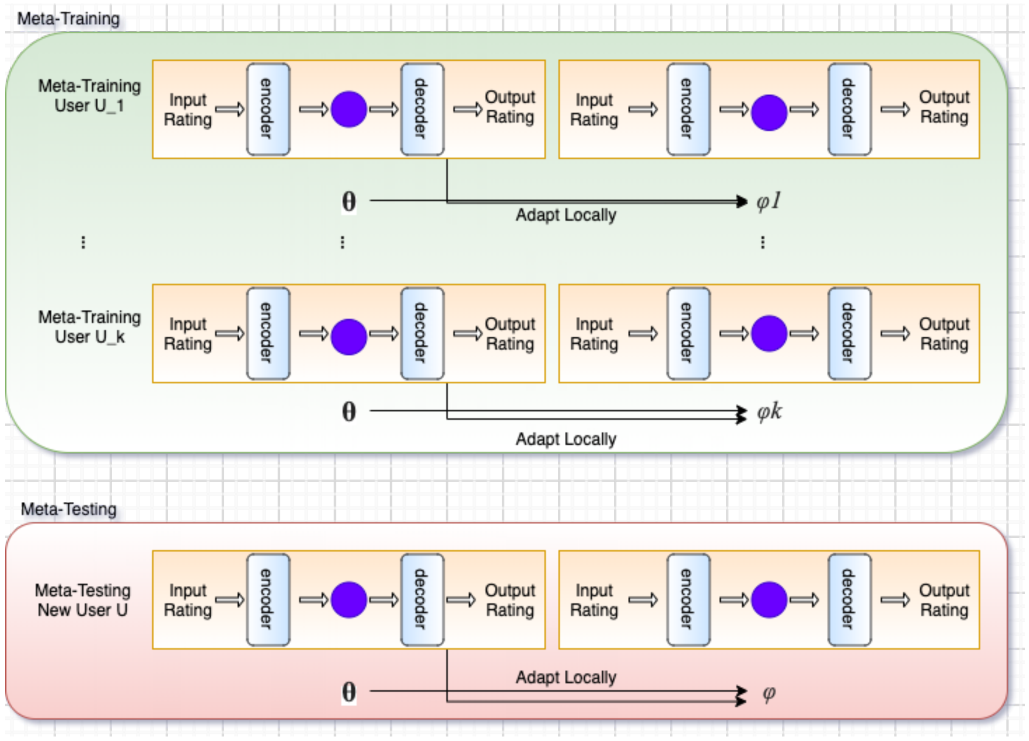


Figure 6.13: The MAML setup in MetaRec-AE. We gradually update the encoder and decoder parameters θ using ϕ so that they can be adapted to any recommendation task with a few SGD steps.

where $||\cdot||_O^2$ confirms that only the weights associated with observed inputs are updated, $||\cdot||_2^2$ denotes the squared L2 norm during regularization, and λ is a regularization parameter that prevents potential for over-fitting. The full training procedure is summarized for reference in Algorithm 3.

With MAML, we want MetaRec-AE to have faster convergence to an effective solution for any rating r_{ui} based on fewer iterations in algorithm 3. The workflow is illustrated in figure 6.13.

Consider the recommendation task $T^u = (D^u_{train}, D^u_{test})$ for user u that consists of a training set D^u_{train} and a test set D^u_{test}.

- **Local updates**: MetaRec-AE learns the autoencoder's global parameters θ shared across a set of meta-training tasks T_{train} and adapts them to local task-specific parameters ϕ^u (with respect to the loss on D^u_{train}).
- **Global update**: Next, on the test set D^u_{test}, the loss under ϕ^u is calcu-

lated and backward propagated to update the global θ.

Formally, MAML seeks to optimize this objective function:

$$\begin{aligned}\min_{\theta} \sum_{T^u \in T_{train}} & L(\theta^{(0)} - \alpha \nabla_\theta L(\theta^{(0)}, D^u_{train}), D^u_{test}) \\ &= \min \sum_{T^u \in T_{train}} L(\phi^u, D^u_{test})\end{aligned} \tag{6.58}$$

where L is the loss function, ∇ is the gradient, and α is the local learning rate. The task-specific parameter $\phi^u = \theta - \alpha \nabla_\theta L(\theta, D^u_{train})$ is adapted to T^u after every gradient step from θ. The full training procedure is summarized in Algorithm 4.

Algorithm 4: MAML Algorithm for MetaRec-AE

```
Input  : Trainable global parameters $\theta$, user set $U$, local and global
         hyper-parameters $\alpha, \beta$
Output: Learned global parameters $\theta$
1  initialize $\theta^{(0)}$;
2  $i = 0$;
3  while not converged do
4      for $u \in U$ do
5          update autoencoder
6          adapt locally: $\phi^u = \theta^{(0,i)} - \alpha \nabla_\theta L(\theta^{(0,i)})$
7      end
8      update globally: $\theta^{(0,i+1)} \leftarrow \theta^{(0,i)} - \beta \nabla_\phi L(\phi^u)$
9      $i \leftarrow i + 1$
10 end
```

For each task $T^u \in T_{test}$ during meta-testing, MetaRec-AE adapts θ learned during meta-training via a few gradient steps with respect to its training set D^u_{train}. We then use the adapted parameters to predict the ratings in the test set r_{ui} ($i \in D^u_{test}$).

6.4.6 Experimental Results

We evaluate MetaRec-AE on MovieLens1M by comparing it with CDAE, MultVAE, SVAE, and ESAE and assessing generalization ability. Following the experimental setup of competing works [86,123,138,166], we pre-processed the dataset to work with implicit feedback for the task of top-k recommendation across various metrics. In particular, we binarized the explicit ratings data by considering only the user-item pairs where the ratings provided by the user was greater than 3 on a range from 1 to 5. Additionally, we only kept users who have interacted with at least 5 items.

For the top-k recommendation task, we present each user a set of k items with highest probability of being preferred by that user, but are not interacted by the user in the training data. We use the following three ranking-based metrics: Precision@k, Recall@k, and NDCG@k. In our experiments, we set $k = 100$ for all the models. For each user, these metrics compare the predicted rank of the items in the test set with their ground-truth rank.

Precision@k for user u is the percentage of recommended items in the top-k recommendation list that are actually preferred by user u:

$$Precision@k = \frac{Hits@k}{k} \tag{6.59}$$

Recall@k for user u is the percentage of items actually preferred by user u that exists in the top-k recommendation list:

$$Recall@k = \frac{Hits@k}{|R|} \tag{6.60}$$

where $Hits@k = \sum_i r_i$ is the number of items present in the top-k recommendation list that were actually preferred by user u, r_i is the implicit rating (0 or 1) that user u gives to item i in the list, and R is the set of all items with $r_i = 1$.

NDCG@k (Normalized Discounted Cumulative Gain) gives additional weight to the relevance of items on top of the top-k recommendation list:

$$NDCG@k = \frac{DCGs@k}{IDCG@k} \tag{6.61}$$

where:

$$\begin{aligned} DCG@k &= \sum_{i=1}^{k} \frac{r_i}{\log(i+1)} \\ IDCG@k &= \sum_{i=1}^{|R|} \frac{1}{\log(i+1)} \end{aligned} \tag{6.62}$$

We randomly separate the ratings data into 80% training set, 10% validation set, and 10% test set. We train the models using ratings of the training

Table 6.4: Results For Our Autoencoder Experiments (higher Precision, Recall, and NDCG values are better).

Models	Precision@100	Recall@100	NDCG@100	Training Time
CDAE	**8.94**	41.37	25.28	17m29s
MultVAE	**8.86**	41.15	25.08	**6m31s**
SVAE	8.18	**58.49**	**38.07**	6h37m19s
ESAE	7.57	41.81	25.61	**10m12s**
MetaRec-AE (Ours)	8.34	**55.12**	**33.06**	18m24s

users and evaluate the models on the ratings of the validation users. We then compute the metrics above by looking at how well the models ranks the unseen ratings from the test users.

Table 6.4 lists the Precision@100, Recall@100, and NDCG@100 performance of all the Autoencoder methods for the test set after 50 training epochs. The implementation details are provided in the appendix (Section A.4). Our main observations are the following:

- MetaRec-AE has an average performance across three metrics: third highest for *Precision*@100 (7% lower than the highest baseline CDAE) and second highest for *Recall*@100 and *NDCG*@100 (6% and 15% lower than the highest baseline SVAE, respectively). The training time for MetaRec-AE is also average among competing methods (roughly 3x longer than the fastest baseline MultVAE).

- SVAE is the best performing model in *Recall*@100 and *NDCG*@100 metrics. This is due to the recurrent nature of the model, which handles temporal information very effectively. However, it also took many order of magnitude longer to train on a single CPU device (22x longer than MetaRec-AE).

- MultVAE took the shortest time to train, which illustrates the efficiency of learning with variational inference.

- ESAE achieved higher *Recall*@100 and *NDCG*@100 metrics than both CDAE and MultVAE, which is surprising given the fact that it is a linear model. This brought up an interesting point that deep architectures are often times not the best options for sparse datasets.

For possible improvements to our proposed MetaRec-AE, we believe that adding more hidden layers between the encoder and the decoder of the base

AE model and tuning hyper-parameters more carefully will improve the performance. Furthermore, it would be interesting (in future work) to see if using either a denoising AE, a variational AE, or a simple shallow AE as the base model would yield better performance or faster training time.

Chapter 7

Conclusions

7.1 Summary and Contributions

In this thesis, we considered the problem of building collaborative filtering systems, aiming to find an effective way to recommend items for a particular user based on the items previously rated by other users. In Chapter 2, we formally defined the recommendation task and outlined a few modeling approaches to consider when tackling this task. In Chapter 3, we presented a taxonomy of meta-learning techniques and provided a comprehensive overview of each area. In Chapter 4, we examined related work in the literature that was relevant to our research.

After exploring the pertinent work in detail, we proposed the MetaRec framework in Chapter 5, which embeds the Model-Agnostic Meta-Learning algorithm (MAML for short) into the learning process of collaborative filtering. The aim is to find a good set of model parameters such that one or a few gradient steps can lead to effective generalization.

After presenting the MetaRec framework, we considered an array of different experimental scenarios in Chapter 6. First, in the context of matrix factorization models, we showed how adding meta-optimization could lead to better performance while working with explicit feedback. Second, in the context of multi-layer perceptron models, we showed how adding meta-optimization could lead to competitive performance while working with implicit feedback. Finally, in the context of autoencoder models, we showed how meta-learning could adapt to a fairly complicated generative model quite well and achieve a reasonable performance while working with implicit feedback.

Our contributions provide a new way of stating, formalizing, and addressing the challenge of obtaining strong accuracy in recommendations - whether with explicit feedback or implicit feedback. This thesis aimed to elaborate a more comprehensive perspective that leverages learning from one recommendation task to another, instead of learning from scratch as has been traditionally the case in the field.

7.2 Discussion and Future Work

Similar to many academic endeavors, our research presented here results in many more questions than answers. Below, we discuss several open questions and future directions that we found compelling and worth pursuing.

7.2.1 More Complex Base Models and Better MAML Training for MetaRec

As addressed in Section 3.4, MAML requires very deep neural architectures to obtain good inner gradient updates effectively. We believe that using more complex base models for MetaRec would lead to better performance (such as using a multi-layer perceptron with multiple hidden layers for MetaRec-MLP or an autoencoder with multiple layers between the encoding and decoding phases for MetaRec-AE).

Furthermore, the original MAML algorithm used in MetaRec has several issues. We consulted suggestions from [4] and proposed the following ideas to improve MAML training:

- **Training Instability**: MAML training is sensitive to the choice of the model architecture and hyper-parameter setup. For example, the instability might be due to vanishing or exploding gradients, especially in the case of a very deep network. A fix for this issue would be to design a multi-step loss optimization procedure (a similar idea is in [144]), which computes the global loss after every local update.

- **Second-Order Derivative Cost**: MAML ignores the second-order derivative cost, which could impair generalization. A fix for this issue would be to apply derivative-order annealing. For example, if we train MAML for 100 epochs, we use first-order gradients for the first 50 epochs and second-order gradients for the remaining 50 epochs.

- **Fixed Learning Rates During Local and Global Updates**: MAML uses a fixed learning rate for all parameters and update steps, which could damage generalization and convergence speed. Using a fixed learning rate requires multiple hyper-parameter searches to find the optimal choice, which is computationally expensive. A fix for this issue is to learn the learning rate for each parameter in the base model and to learn different learning rates for each adaptation that the base model takes [84].

7.2.2 Other Meta-Learning Schemes For Recommendations

We focused on integrating MAML, an optimization-based meta-learning approach, for MetaRec. As extensively discussed in Chapter 3, black-box and non-parametric schemes are two other forms of meta-learning. We think that applying them to the context of collaborative filtering would be very interesting. Specifically, we think that:

- **Black-box meta-learning** approaches can represent any function of our training data. MAMO [31] is a successful attempt that uses memory-augmented networks [126], a black-box architecture, for the recommendation task. We believe other black-box architectures such as conditional neural processes [39], Meta Network [101], and SNAIL [97] would serve as promising candidates for automatic recommendation.
- **Non-parametric meta-learning** approaches do not involve any back-propagation, so they are computationally fast and easy to optimize. We have not seen any work that uses these approaches for the recommendation task, so future work that applies non-parametric models like the Siamese Network [69], the Matching Network [158], or the Prototypical Network [136] to the collaborative filtering problem would potentially be very promising.

7.2.3 Addressing The Scalability and Sparsity Challenges

Finally, as mentioned in Section 1.1.3, scalability and sparsity are two other major challenges for collaborative filtering systems. In future work, we would like to design evaluation settings that can quantitatively address these issues:

- For **scalability**,we view training time as a reliable proxy metric in the sense that short training time correlates to better scalability. Throughout our experiments with MetaRec, we observed a computational trade-off between accuracy performance and computational cost. Thus, we believe that better MAML training, as proposed above, can reduce computational cost. Testing MetaRec on bigger datasets would justify this hypothesis.

- For **sparsity**, we would like to consider the experimental setup proposed in [31, 80, 91]. More specifically, they considered the recommendation performance on four scenarios: (1) warm users for warm items, (2) warm users for cold items, (3) cold users for warm items, and (4) cold users for cold items. According to their first comment time, the users in the MovieLens dataset were classified into warm or cold categories. The model would then be evaluated on these different scenarios - measuring the recommendation performance for warm users and cold users.

***

Using algorithmic frameworks capable of leveraging other users' preferences instead of learning from scratch for each new, incoming user, recommendation systems will be better prepared to handle the diversity of real-world data in production robustly. Ultimately, they will be better equipped to improve themselves. More importantly, as recommendation systems improve and become more prominent, remaining questions about societal issues such as fairness, interpretability, and value alignment will need to be answered in the decades to come.

Bibliography

[1] Roee Aharoni, Melvin Johnson, and Orhan Firat. Massively multilingual neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1, pages 3874-3884.*, 2019.

[2] Kelsey Allen, Evan Shelhamer, Hanul Shin, and Joshua Tenenbaum. Infinite mixture prototypes for few-shot learning. In *Proceedings of the 36th International Conference on Machine Learning, PMLR 97:232-241*, 2019.

[3] Asim Ansari, Skander Essegaier, and Rajeev Kohli. Internet recommendations systems. In *J. Marketing Research, pages 363-375*, 2000.

[4] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. In *International Conference on Learning Conference*, 2019.

[5] Esma Ameur, Gilles Brassard, Jos Fernandez, and Flavien Serge Mani Onana. Alambic: a privacy-preserving recommender system for electronic commerce. In *International Journal of Information Security, volume 7, issue 5, pages 307-334*, 2008.

[6] Marko Balabanovic and Yoav Shoham. Fab: Content-based, collaborative recommendation. In *ACM Comm., volume 40, no. 3, pages 66-72*, 1994.

[7] Chumki Basu, Haym Hirsh, and William Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *Recommender Systems. Papers from 1998 Workshop, Technical Report WS-98-08, AAAI Press*, 1999.

[8] Zeynep Batmaz, Ali Yurekli, Alper Bilge, and Cihan Kaleli. A review on deep learning for recommender systems: challenges and remedies. In *Artificial Intelligence Review, volume 52, page 137*, 2019.

[9] Harkirat Singh Behl, Atlm Gne Baydin, and Philip H.S. Torr. Alpha maml: Adaptive model-agnostic meta-learning. In *The 6th ICML Workshop on Automated Machine Learning*, 2019.

[10] Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. In *Conference on Optimality in Biological and Artificial Networks*, 1992.

[11] David Bennock, Eric Horvitz, Steve Lawrence, and Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory and model based approach. In *Proc. International Joint Conf. Artificial Intelligence Workshop: Machine Learning for Information Filtering*, 1999.

[12] Shlomo Berkovsky, Tsvi Kuflik, and Francesco Ricci. Cross-technique mediation of user models. In *Proceedings of International Conference on Adaptive Hypermedia and AdaptiveWeb-Based Systems, pages 21-30. Dublin*, 2006.

[13] Shlomo Berkovsky, Tsvi Kuflik, and Francesco Ricci. Cross-representation mediation of user models. In *User Modeling and User-Adapted Interaction 19(1-2), pages 35-63*, 2009.

[14] Shlomo Berkvosky, Yaniv Eytani, Tsvi Kuflik, and Francesco Ricci. Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In *RecSys 07: Proceedings of the 2007 ACM conference on Recommender systems, pages 9-16. ACM Press, New York, NY, USA*, 2007.

[15] Luca Bertinetto, Joo F. Henriques, Philip H.S. Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2019.

[16] Homanga Bharadhwaj. Meta-learning for user cold-start recommendation. In *2019 International Joint Conference on Neural Networks (IJCNN), pages 1-8*, 2019.

[17] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46:109–132, 2013.

[18] Erin Boyle and Jana Beck. Understanding latent style. In *https://multithreaded.stitchfix.com/blog/2018/06/28/latent-style/*, 2018.

[19] John Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. 14th Conf. Uncertainty in Artificial Intelligence*, 1998.

[20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *arXiv preprint arXiv:2005.14165*, 2020.

[21] Robin Burke. Knowledge-based recommender systems. In *Encyclopedia of Library and Information Systems, volume 69, supplement 32, Marcel Dekker*, 2000.

[22] John Canny. Collaborative filtering with privacy. In *IEEE Symposium on Security and Privacy, pages 45-57*, 2002.

[23] Rich Caruana. Multitask learning. In *Machine Learning 28, 4175. https://doi.org/10.1023/A:1007379606734*, 1997.

[24] Sotirios P Chatzis, Panayiotis Christodoulou, and Andreas S Andreou. Recurrent latent variable networks for session-based recommendation. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*, pages 38–45, 2017.

[25] Yves Chauvin and David E Rumelhart. *Backpropagation: theory, architectures, and applications.* Psychology press, 1995.

[26] Cen Chen, Peilin Zhao, Longfei Li, Jun Zhou, Xiaolong Li, and Minghui Qiu. Locally connected deep learning framework for industrial-scale recommender systems. In *WWW*, 2017.

[27] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, and Mustafa Ispir. Wide and deep learning for recommender systems. In *RecSys, pages 7-10*, 2016.

[28] Mark Claypool, Anuja Gokhale, Tim Miranda, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. Combining content-based and collaborative

filters in an online newspaper. In *Proc. ACM SIGIR '99 Workshop Recommender System: Algorithms and Evaluation*, 1999.

[29] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *RecSys, pages 191-198*, 2016.

[30] Shuiguang Deng, Longtao Huang, Guandong Xu, Xindong Wu, and Zhaohui Wu. On deep learning for trust-aware recommendations in social networks. In *IEEE Transactions on Neural Networks and Learning Systems, Volume 28, number 5, page 1164-1177*, 2017.

[31] Manqing Dong, Feng Yuan, Lina Yao, Xiwei Xu, and Liming Zhu. Mamo: Memory-augmented meta-optimization for cold-start recommendation. In *arXiv:2007.03183*, 2020.

[32] Zhengxiao Du, Xiaowei Wang, Hongxia Yang, Jingren Zhou, and Jie Tang. Sequential scenario-specific meta-learner for online recommendation. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2019.

[33] Gintare Karolina Dziugaite and Daniel Roy. Neural network matrix factorization. In *arXiv preprint arXiv:1511.06443*, 2015.

[34] Ali Elkahky, Yang Song, and Xiaodong He. A multi-view deep learning approach for cross-domain user modeling in recommendation systems. In *Proceedings of the 24th International Conference on World Wide Web, Florence, Italy, pages 278-288*, 2015.

[35] Ignacio Fernández-Tobías, Iván Cantador, Marius Kaminskas, and Francesco Ricci. Cross-domain recommender systems: A survey of the state of the art.

[36] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

[37] Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. In *International Conference on Learning Conference*, 2018.

[38] Victor Garcia and Joan Bruna. Few-shot learning with graph neural networks. In *International Conference on Learning Conference*, 2018.

[39] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and S. M. Ali Eslami. Conditional neural processes. In *Proceedings of the 35th International Conference on Machine Learning, PMLR 80:1704-1713*, 2018.

[40] Thomas George and Srujana Merugu. A scalable collaborative filtering framework based on co-clustering. In *Proceedings of the 5th IEEE Conference on Data Mining (ICDM), pages 625-628*, 2005.

[41] Song Jie Gong, Hong Wu Ye, and Ya Dai. Combining svd and item-based recommender in collaborative filtering. In *Second International Workshop on Knowledge Discovery and Data Mining, pages 769-772*, 2009.

[42] Nathaniel Good, Ben Schafer, Joseph Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Prof. Conf. Am. Assoc. Artificial Intelligence, pages 439-446*, 1999.

[43] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. In *Proceedings of the International Conference on Learning Representations*, 2018.

[44] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[45] Prince Grover. Various implementations of collaborative filtering. In *https://towardsdatascience.com/various-implementations-of-collaborative-filtering*, 2017.

[46] Naiyang Guan, Dacheng Tao, Zhigang Luo, and Bo Yuan. Nenmf: An optimal gradient method for non-negative matrix factorization. In *IEEE Transactions on Signal Processing, Volume 60, Number 6, pages 2882-2898*, 2012.

[47] Asela Gunawardana and Christopher Meek. Tied boltzmann machines for cold start recommendations. In *RecSys, pages 19-26*, 2008.

[48] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. 2017.

[49] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for ctr prediction. In *IJCAI, pages 2782-2788*, 2017.

[50] Peng Hang, Bo Xie, Fan Yang, and Ruimin Shen. A scalable p2p recommender system based on distributed collaborative filtering. In *Expert Systems with Applications, 27(2):203-210*, 2004.

[51] Maxwell Harper and Joseph Konstan. The movielens datasets: History and context. In *ACM Transactions on Interactive Intelligent Systems, Article No. 19*, 2015.

[52] James Harrison, Apoorva Sharma, and Marco Pavone. Meta-learning priors for efficient online bayesian regression. In *Workshop on the Algorithmic Foundations of Robotics*, 2018.

[53] Xiangnan He and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. 2017.

[54] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *WWW, pages 173-182*, 2017.

[55] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[56] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. Parallel recurrent neural network architectures for feature-rich session-based recommendations. *Proceedings of the 10th ACM Conference on Recommender Systems, page 241248*, 2016.

[57] Geoffrey E Hinton. Boltzmann machine. *Scholarpedia*, 2(5):1668, 2007.

[58] Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.

[59] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

[60] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[61] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. Collaborative metric learning. In *WWW, pages 193-201*, 2017.

[62] Liang Hu, Jian Cao, Guandong Xu, Longbing Cao, Zhiping Gu, and Wei Cao. Deep modeling of group preferences for group-based recommendation. In *AAAI*, 2014.

[63] Markus Jessenitschnig and Markus Zanker. A generic user modeling component for hybrid recommendation strategies. In *E-Commerce Technology, IEEE International Conference, pages 337-344*, 2009.

[64] Donghyun Kim, Chanyoung Park, Jinoh Oh, and Hwanjo Yu. Deep hybrid recommender systems via exploiting document context and statistics of items. In *Inf Sci 417:72-87, https://doi.org/10.1016/j.ins.2017.06.026*, 2017.

[65] Jaehong Kim, Sangyeul Lee, Sungwan Kim, Moonsu Cha, Jung Kwon Lee, Youngduck Choi, Yongseok Choi, Dong-Yeon Cho, and Jiwon Kim. Auto-meta: Automated gradient based meta learner search. In *Workshop on Meta-Learning at NeurIPS*, 2018.

[66] Yong-Deok Kim and Seungjin Choi. Scalable variational bayesian matrix factorization with side information. In *Artificial Intelligence and Statistics*, pages 493–502, 2014.

[67] Diederik Kingma and Max Welling. Auto-encoding variational bayes. In *arXiv preprint arXiv:1312.6114*, 2013.

[68] Alfred Kobsa. Generic user modeling systems. In *P. Brusilovsky, A. Kobsa,W. Nejdl (eds.) The Adaptive Web, Lecture Notes in Computer Science, volume 4321, pages 136-154. Springer*, 2007.

[69] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[70] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *SIGKDD, pages 426-434*, 2008.

[71] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. In *IEEE Computer Society 0018-9162/09, pages 42-49*, 2009.

[72] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[73] Maciej Kula. Mixture-of-tastes models for representing users with diverse interests. 2017.

[74] Shyong Lam, Dan Frankowski, and John Riedl. Do you trust your recommendations? an exploration of security and privacy issues in recommender systems. In *G. Muller (ed.) ETRICS, Lecture Notes in Computer Science, volume 3995, pages 14-29. Springer*, 2006.

[75] James Le. The 5 variants of multi-layer perceptron for collaborative filtering. In *https://jameskle.com/writes/rec-sys-part-5*, 2020.

[76] James Le. The 6 variants of autoencoders for collaborative filtering. In *https://jameskle.com/writes/rec-sys-part-6*, 2020.

[77] James Le. The 7 variants of matrix factorization for collaborative filtering. In *https://jameskle.com/writes/rec-sys-part-4*, 2020.

[78] James Le. Meta-learning is all you need. In *https://jameskle.com/writes/meta-learning-is-all-you-need*, 2020.

[79] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[80] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. Melu: Meta-learned user preference estimator for cold-start recommendation. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2019.

[81] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceeedings of the conference on Computer Vision and Pattern Recognition*, 2019.

[82] Sheng Li, Jaya Kawale, and Yun Fu. Deep collaborative filtering via marginalized denoising autoencoder. In *CIKM, pages 811-820*, 2015.

[83] Xiaopeng Li and James She. Collaborative variational autoencoder for recommender systems. In *SIGKDD*, 2017.

[84] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. In *arXiv ePrint 1707.09835v2*, 2017.

[85] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *arXiv preprint arXiv:1803.05170*, 2018.

[86] Dawen Liang, Rahul Krishman, Matthew Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *arXiv preprint arXiv:1802.05814*, 2018.

[87] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[88] Zhaoyang Liu, Haokun Chen, Fei Sun, Xu Xie, Jinyang Gao, Bolin Ding, and Yanyan Shen. Intent preference decoupling for user representation on online recommender system. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, 2020.

[89] Gilles Louppe. Collaborative filtering: Scalable approaches using restricted boltzmann machine. In *Master's Thesis, University of Liege*, 2010.

[90] Xugang Lu, Yu Tsao, Shigeki Matsuda, and Chiori Hori. Speech enhancement based on deep denoising autoencoder. In *Interspeech, pages 436-440*, 2013.

[91] Yuanfu Lu, Yuan Fang, and Chuan Shi. Meta-learning on heterogeneous information networks for cold-start recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020.

[92] Mi Luo, Fei Chen, Pengxiang Cheng, Zhenhua Dong, Xiuqiang He, Jiashi Feng, and Zhenguo Li. Metaselector: Meta-learning for recommendation with user-level adaptive model selection. In *Proceedings of The Web Conference 2020*, 2020.

[93] Lorraine McGinty and Barry Smyth. On the role of diversity in conversational recommender systems. In *A. Aamodt, D. Bridge, K. Ashley (eds.) ICCBR 2003, the 5th International Conference on Case-Based Reasoning, pages 276-290. Trondheim, Norway*, 2003.

[94] David McSherry. Diversity-conscious retrieval. In *S. Craw, A. Preece (eds.) Advances in Case-Based Reasoning, Proceedings of the 6th European Conference on Case Based Reasoning, ECCBR 2002, pages 219-233. Springer Verlag, Aberdeen, Scotland*, 2002.

[95] Frank McSherry and Ilya Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *KDD 09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 627-636. ACM, New York, NY, USA*, 2009.

[96] Prem Melville, Raymond Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Proc. 18th Nat'l Conf. Artificial Intelligence*, 2002.

[97] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *Proceedings of the International Conference on Learning Representations*, 2018.

[98] Niyas Mohammed. How to autoencode your pokmon. In `https://medium.com/hackernoon/how-to-autoencode-your-pokemon`, 2017.

[99] Awhan Mohanty. Multi layer perceptron (mlp) models on real world banking data. In `https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data`, 2019.

[100] Miquel Montaner, Beatriz Lpez, and Josep Llus de la Rosa. A taxonomy of recommender agents on the internet. In *Artificial Intelligence Review 19(4), pages 285-330*, 2003.

[101] Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70*, 2017.

[102] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. In *arXiv ePrint 1803.02999*, 2018.

[103] Alexander G Ororbia II, Tomas Mikolov, and David Reitter. Learning simpler language models with the differential state framework. *Neural computation*, 29(12):3327–3352, 2017.

[104] Yuanxin Ouyang, Wenqi Liu, Wenge Rong, and Zhang Xiong. Autoencoder-based collaborative filtering. In *International Conference on Neural Information Processing, pages 284-291*, 2014.

[105] Weike Pan, Evan Wei Xiang, Nathan Nan Liu, and Qiang Yang. Transfer learning in collaborative filtering for sparsity reduction. 2010.

[106] Manos Papagelis, Ioannis Rousidis, Dimitris Plexousakis, and Elias Theoharopoulos. Incremental collaborative filtering for highly-scalable recommendation algorithms. In *M.S. Hacid, N.V. Murray, Z.W. Ras, S. Tsumoto (eds.) ISMIS, Lecture Notes in Computer Science, vol. 3488, pages 553-561. Springer*, 2005.

[107] Michael Pazzani. A framework for collaborative, content-based, and demographic filtering. In *Artificial Intelligence Rev., pages 393-408*, 1999.

[108] Xishuai Peng, Yuan xiang Li, Xian Wei, Jianhua Luo, and Yi Lu Murphey. Traffic sign recognition with transfer learning. In *Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence (SSCI), pages 1-7*, 2017.

[109] Huseyin Polat and Wenliang Du. Svd-based collaborative filtering with privacy. In *ACM Symposium on Applied Computing*, 2005.

[110] Alexandrin Popescul, Lyle Ungar, David Pennock, and Steve Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proc. 17th Conf. Uncertainty in Artificial Intelligence*, 2001.

[111] Ian Porteous, Arthur U Asuncion, and Max Welling. Bayesian matrix factorization with side information and dirichlet process mixtures. 2010.

[112] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 130–137, 2017.

[113] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. In *Technical Report, OpenAI*, 2019.

[114] Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. In *Proceedings of the 33rd Conference on Neural Information Processing Systems*, 2019.

[115] Naren Ramakrishnan, Benjamin Keller, Batul Mirza, Ananth Grama, and George Karypis. When being weak is brave: Privacy in recommender systems. In *IEEE Internet Computing*, 2001.

[116] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Conference*, 2017.

[117] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.

[118] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceeedings of the 31st International Conference on Machine Learning, pages 1278-1286*, 2014.

[119] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul Kantor. Recommender systems handbook. In *Springer, ISBN: 978-0-387-85819-7*, 2011.

[120] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML, pages 833-840*, 2011.

[121] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[122] Massimiliano Ruocco, Ole Steinar Lillestøl Skrede, and Helge Langseth. Inter-session modeling for session-based recommendation. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*, pages 24–31, 2017.

[123] Noveen Sachdeva, Giuseppe Manco, Ettore Ritacco, and Vikram Pudi. Sequential variational autoencoders for collaborative filtering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining - WSDM 19*. ACM Press, 2019.

[124] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Proceedings of Neural Information Processing Systems Foundation*, 2017.

[125] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML, pages 791-798*, 2007.

[126] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. In *Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA*, 2016.

[127] Badrul Sarwar, Joseph Konstan, and John Riedl. Distributed recommender systems for internet commerce. In *M. Khosrow-Pour (ed.) Encyclopedia of Information Science and Technology (II), pages 907-911. Idea Group*, 2005.

[128] Andrew Schein, Alexandrin Popescu, David Pennock, and Lyle Ungar. Methods and metrics for cold-start recommendations. In *Proc. 25th Ann. Int'l ACM SIGIR Conf.*, 2002.

[129] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *WWW, pages 111-112*, 2015.

[130] Joan Serra and Alexandros Karatzoglou. Getting deep recommenders fit: Bloom embeddings for sparse binary input/output networks. In *RecSys, pages 279-287*, 2017.

[131] Guy Shani and Asela Gunawardana. Evaluating recommendation systems. In *Recommender systems handbook*, pages 257–297. Springer, 2011.

[132] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating 'word of mouth'. In *Prof. Conf. Human Factors in Computing Systems*, 1995.

[133] Xiaoxuan Shen, Baolin Yi, Zhaoli Zhang, Jiangbo Shu, and Hai Liu. Automatic recommendation technology for learning resources with convolutional neural network. In *Proceedings of the International Symposium on Educational Technology, page 30-34*, 2016.

[134] Donghyuk Shin, Suleyman Cetintas, Kuang-Chih Lee, and Inderjit Dhillon. Tumblr blog recommendation with boosted inductive matrix completion. In *Proceedings of the 24th ACM Conference on Information and Knowledge Management, page 203-212*, 2015.

[135] Reza Shokri, Pedram Pedarsani, George Theodorakopoulos, and Jean-Pierre Hubaux. Preserving privacy in collaborative filtering through distributed aggregation of offline profiles. In *RecSys 09: Proceedings of the third ACM conference on Recommender systems, pages 157-164. ACM, New York, NY, USA*, 2009.

[136] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, 2017.

[137] Ian Soboroff. Combining content and collaboration in text filtering. In *Proc. Int'l Joint Conf. Artificial Intelligence Workshop: Machine Learning for Information Filtering*, 1999.

[138] Harald Steck. Embarrassingly shallow autoencoders for sparse data. In *The World Wide Web Conference 2019*. ACM Press, 2019.

[139] Florian Strub, Romaric Gaudel, and Jeremie Mary. Hybrid recommender system based on autoencoders. In *Proceedings of the 1st Workshop on Deep Learning for Recommender System, pages 11-16*, 2016.

[140] Florian Strub and Jeremie Mary. Collaborative filtering with stacked denoising autoencoders and sparse inputs. In *NIPS Workshop*, 2015.

[141] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009, 2009.

[142] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H.S. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceeedings of the conference on Computer Vision and Pattern Recognition*, 2018.

[143] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

[144] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[145] Nima Taghipour and Ahmad Kardan. A hybrid web recommender system based on q-learning. In *Proceedings of the 2008 ACM symposium on applied computing, pages 1164-1168. ACM, New York, NY, USA*, 2008.

[146] Yong Kiam Tan, Xinxing Xu, and Yong Liu. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 17–22, 2016.

[147] Jiaxi Tang and Ke Wang. Ranking distillation: Learning compact ranking models with high performance for recommender systems. In *SIGKDD*, 2018.

[148] Nguyen Thai-Nghe, Lucas Drumond, Tomas Horvath, Alexandros Nanopoulos, and Lars Schmidt-Thieme. Matrix and tensor factorization for predicting student performance. In *Proceedings of the 3rd International Conference on Computer Supported Education*, 2011.

[149] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *NIPS'95: Proceedings of the 8th International Conference on Neural Information Processing Systems, pages 640-646*, 1995.

[150] Tran The Truyen, Dinh Quoc Phung, and Svetha Venkatesh. Ordinal boltzmann machines for collaborative filtering. In *Proceedings of the 25th Conference on Uncertainty in AI, pages 548-556*, 2009.

[151] Trinh Xuan Tuan and Tu Minh Phuong. 3d convolutional networks for session-based recommendation with content features. In *Proceedings of the 11th ACM Conference on Recommender Systems, page 138-146*, 2017.

[152] Moshe Unger, Ariel Bar, Bracha Shapira, and LiorRokach. Towards latent context-aware recommendation systems. In *Knowledge-Based Systems, Volume 104, page 165-178*, 2016.

[153] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Proceedings of the 26th International Conference on NIPS, page 2643-2651*, 2013.

[154] Saúl Vargas. Novelty and diversity enhancement and evaluation in recommender systems. 2015.

[155] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. A meta-learning perspective on cold-start recommendations for items. In *Advances in Neural Information Processing Systems 30*, 2017.

[156] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *arXiv ePrint 1706.03762*, 2017.

[157] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[158] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Proceedings of the 30th Conference on Neural Information Processing Systems*, 2016.

[159] Manolis Vozalis and Konstantinos Margaritis. A recommender system using principal component analysis. In *11th Panhellenic Conference inn Informatics, pages 271-283*, 2007.

[160] Manolis Vozalis and Konstantinos Margaritis. Using svd and demographic data for the enhancement of generalized collaborative filtering. In *An International Journal of Information Sciences, Volume 177, pages 3017-3037*, 2007.

[161] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. Relational stacked denoising autoencoder for tag recommendation. In *Proceedings of the 29th AAAI conference on Artificial Intelligence, page 3052-3058*, 2015.

[162] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *SIGKDD, pages 1235-1244*, 2015.

[163] Xinxi Wang and Ye Wang. Improving content-based and hybrid music recommendation using deep learning. In *Proceedings of the 22nd ACM international conference on Multimedia, pages 627636*, 2014.

[164] Yang Wang and Alfred Kobsa. Performance evaluation of a privacy-enhancing framework for personalized websites. In *G.J. Houben, G.I. McCalla, F. Pianesi, M. Zancanaro (eds.) UMAP, Lecture Notes in Computer Science, volume 5535, pages 78-89. Springer*, 2009.

[165] Sai Wu, Weichao Ren, Chengchao Yu, Gang Chen, Dongxiang Zhang, and Jingbo Zhu. Personal recommendation using deep recurrent neural networks in netease. In *2016 IEEE 32nd international conference on data engineering (ICDE)*, pages 1218–1229. IEEE, 2016.

[166] Yao Wu, Christopher DuBois, Alice Zheng, and Martin Ester. Collaborative denoising autoencoders for top-n recommender systems. In *WSDM, pages 153-162*, 2016.

[167] Haochao Ying, Liang Chen, Yuwen Xiong, and Jian Wu. Collaborative deep ranking: a hybrid pair-wise recommendation algorithm with implicit feedback. In *PAKDD, pages 555-567*, 2016.

[168] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. In *Robotics: Science and Systems 2018*, 2018.

[169] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

[170] Shuai Zhang, Lina Yao, and Xiwei Xu. Autosvd++: An efficient hybrid collaborative filtering model via contractive autoencoders. 2017.

[171] Yang Zhang, Fuli Feng, Chenxu Wang, Xiangnan He, Meng Wang, Yan Li, and Yongdong Zhang. How to retrain recommender system? a sequential meta-learning method. In *SIGIR*, 2020.

[172] Liang Zhao, Yang Wang, Daxiang Dong, and Hao Tian. Learning to recommend via meta parameter partition. In *arXiv:1912.04108*, 2019.

[173] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. Recommending what video to watch next: A multitask ranking system. In *RecSys '19: Proceedings of the 13th ACM Conference on Recommender Systems, pages 43-51*, 2019.

[174] Yin Zheng, Cailiang Liu, Bangsheng Tang, and Hanning Zhoi. Neural autoregressive collaborative filtering for implicit feedback. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, page 2-6*, 2016.

[175] Fengwei Zhou, Bin Wu, and Zhenguo Li. Deep meta-learning: Learning to learn in the concept space. In *arXiv ePrint 1802.03596*, 2018.

[176] Luisa M Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, 2019.

Appendices

Appendix A

Appendix

A.1 Dataset Details

This includes the figures for section 6.1.

Figure A.1 shows a word-cloud visualization of the movie titles. We can recognize that there are a lot of movie franchises in this dataset, as evidenced by words like *II* and *III*... In addition to that, *Day*, *Love*, *Life*, *Time*, *Night*, *Man*, *Dead*, and *American* are among the most commonly occuring words.

Figure A.2 displays the summary statistics and distribution of the ratings data. It appears that users are quite generous in their ratings. The mean

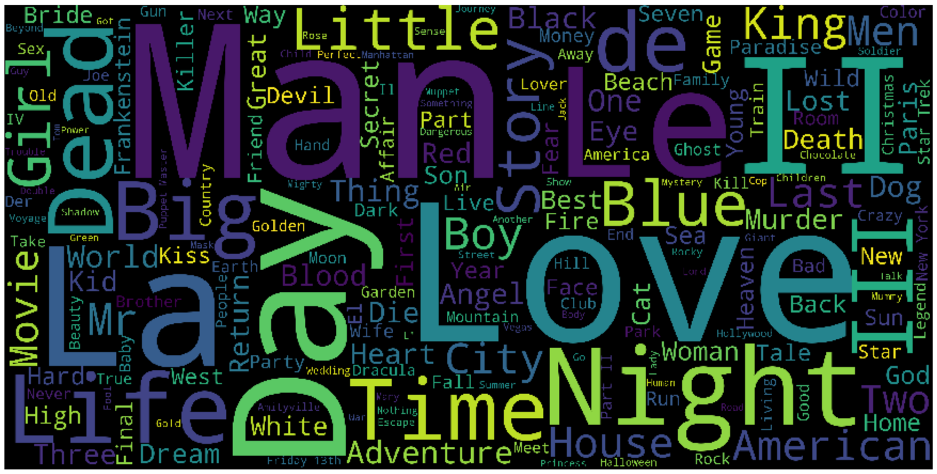


Figure A.1: MovieLens1M movie titles in a word cloud

rating is 3.58 on a scale of 5. Half the movies have a rating of 4 and 5. We personally think that a 5-level rating skill was not a good indicator as people could have different rating styles (i.e. person A could always use 4 for an average movie, whereas person B only gives 4 out for their favorites). Each user rated at least 20 movies, so we doubt the distribution could be caused just by chance variance in the quality of movies.

Figure A.3 shows a word-cloud visualization of the movie genres. The top 5 genres are, in that respect order: Drama, Comedy, Action, Thriller, and Romance.

Figure A.4 shows a subset of 20 movies with the highest rating.

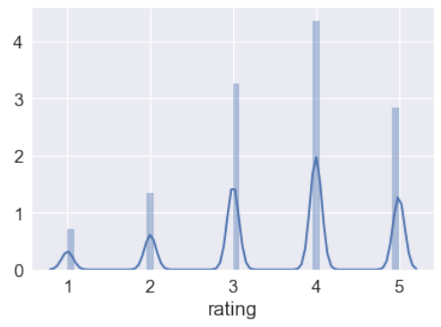


Figure A.2: MovieLens1M rating distribution

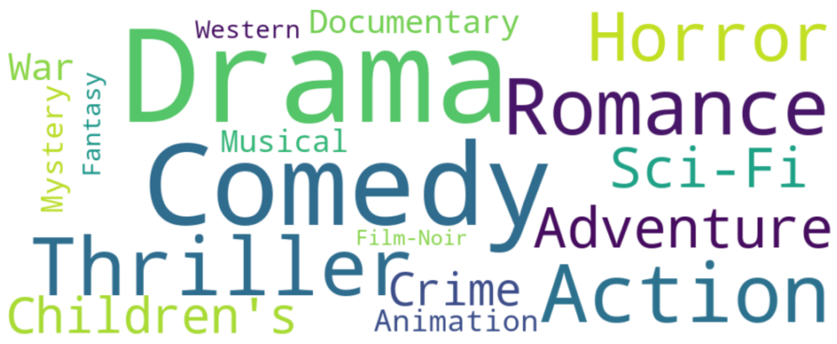


Figure A.3: MovieLens1M movie genres in a word cloud

A.2 Matrix Factorization Methods

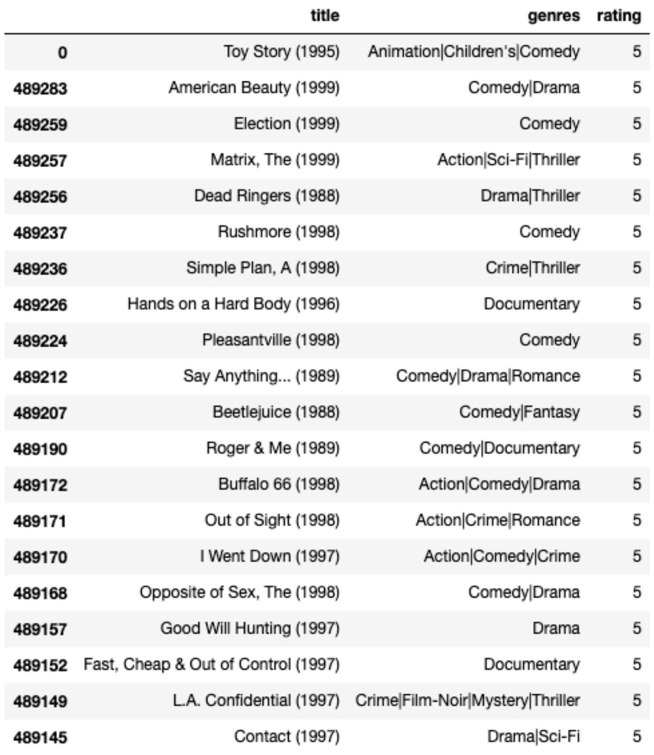
A.2.1 Code

All the matrix factorization models are implemented with PyTorch 1.3.0 in Python 3.6 and runs on a Macbook Pro with CPU computing. We used PyTorch Ignite for model development and evaluation, NumPy and Pandas for data manipulation, and TensorBoard for result visualization.

A.2.2 Parameter Configuration

We use a batch size of 1024 and adopt Adam with a learning rate of 0.01 to optimize our baseline Matrix Factorization models for MovieLens. Here are other configuration settings:

- For MF: the regularization rate is 0.000001 and the dimension of user



	title	genres	rating
0	Toy Story (1995)	Animation|Children's|Comedy	5
489283	American Beauty (1999)	Comedy|Drama	5
489259	Election (1999)	Comedy	5
489257	Matrix, The (1999)	Action|Sci-Fi|Thriller	5
489256	Dead Ringers (1988)	Drama|Thriller	5
489237	Rushmore (1998)	Comedy	5
489236	Simple Plan, A (1998)	Crime|Thriller	5
489226	Hands on a Hard Body (1996)	Documentary	5
489224	Pleasantville (1998)	Comedy	5
489212	Say Anything... (1989)	Comedy|Drama|Romance	5
489207	Beetlejuice (1988)	Comedy|Fantasy	5
489190	Roger & Me (1989)	Comedy|Documentary	5
489172	Buffalo 66 (1998)	Action|Comedy|Drama	5
489171	Out of Sight (1998)	Action|Crime|Romance	5
489170	I Went Down (1997)	Action|Comedy|Crime	5
489168	Opposite of Sex, The (1998)	Comedy|Drama	5
489157	Good Will Hunting (1997)	Drama	5
489152	Fast, Cheap & Out of Control (1997)	Documentary	5
489149	L.A. Confidential (1997)	Crime|Film-Noir|Mystery|Thriller	5
489145	Contact (1997)	Drama|Sci-Fi	5

Figure A.4: MovieLens1M subset of movies with 5-star ratings

and item embeddings is set to 10.

- For MF-Bias: the global regularization rate is 0.000001, the bias regularization rate is 0.000001, and the dimension of user and item embeddings is set to 10.

- For MF-Side: the global regularization rate is 0.000001, the bias regularization rate is 0.000001, and the dimension of user and item embeddings is set to 10.

- For MF-Temporal: the global regularization rate is 0.000001, the bias regularization rate is 0.000001, the regularization rate for temporal features is 0.000001, the regularization rate for user-temporal interaction is 0.000001, the dimension of user and item embeddings is set to 10, and the dimension of temporal embedding is set to 2.

- For FM: the global regularization rate is 0.000001, the bias regularization rate is 0.000001, and the dimension of user and item embeddings is set to 10.

- For MF-Mixture: the global regularization rate is 0.000001, the bias regularization rate is 0.000001, the dimension of user and item embeddings is set to 10, and the dimension of the taste and attention embeddings is set to 4.

- For Variational-MF: the bias regularization rate is 0.000001, the KL-divergence constant rate is 0.0000000001, and the dimension of user and item embeddings is set to 10.

For MetaRec-MF: the global regularization rate is 0.000001, the bias regularization rate is 0.000001, the dimension of user and item embeddings is set to 10, the local learning rate α is 0.0005, and the global learning rate β is 0.005. MetaRec-MF was trained on batches of size 1024.

A.3 Multi-Layer Perceptron Methods

A.3.1 Code

All the multi-layer perceptron models are implemented with PyTorch 1.3.0 in Python 3.6 and runs on a Macbook Pro with CPU computing. We used PyTorch for model development and evaluation, NumPy and Pandas for data manipulation, and Weights and Biases for result visualization.

A.3.2 Parameter Configuration

We use a batch size of 512 and adopt Adam with a learning rate of 0.001 and weight decay of 0.000001 to optimize our baseline Multi-Layer Perceptron models for MovieLens. Here are other configuration settings:

- For WideDeep: the dimension of user and item embeddings is set to 16, the number of hidden layers is 16, the choice of activation function is sigmoid, and the dropout rate is 0.5.
- For DeepFM: the dimension of user and item embeddings is set to 16, the number of hidden layers is 16, the choice of activation function is sigmoid, and the dropout rate is 0.5.
- For xDeepFM: the dimension of user and item embeddings is set to 16, the number of hidden layers is 16, the input of the compressed interaction module is $(16, 16)$, the choice of activation function is sigmoid, and the dropout rate is 0.5.
- For NeuralFM: the dimension of user and item embeddings is set to 64, the number of hidden layers is 64, the choice of activation function is sigmoid, and the dropout rate is 0.2.
- For NeuralCF: the dimension of user and item embeddings is set to 16, the number of hidden layers is 16, the choice of activation function is sigmoid, and the dropout rate is 0.5.

For MetaRec-MLP: the dimension of user and item embeddings is set to 16, the number of hidden layers is 16, the choice of activation function is sigmoid, the dropout rate is 0.5, the local learning rate α is 0.000005, and the global learning rate β is 0.00005. MetaRec-MLP was trained on batches of size 512.

A.4 Autoencoders Methods

A.4.1 Code

All the autoencoder models are implemented with PyTorch 1.3.0 in Python 3.6 and runs on a Macbook Pro with CPU computing. We used PyTorch for model development and evaluation, NumPy and Pandas for data manipulation, and CometML for result visualization.

A.4.2 Parameter Configuration

We adopt Adam with a learning rate of 0.001 to optimize our baseline Autoencoder models for MovieLens. Here are other configuration settings:

- For CDAE: the number of hidden layers is 50, the corruption ratio for the data to the input layer is set to be 0.5, and the choice of activation function is TanH. CDAE was trained on batches of size 512.
- For MultVAE: the dimension of the encoder module is 200, the dropout rate is 0.5, and the KL-divergence annealing rate is 0.2. MultVAE was trained on batches of size 512.
- For SVAE: the dimensions of two encoding layers are 150 and 64, the recurrent layer is initialized as a Gated Recurrent Unit with 200 cells, the size of the embedding layer is 256, and the number of latent factors is 64. SVAE was trained on batches of size 1 (because we don't pack multiple sequences in the same batch).
- For ESAE: This is a linear model without a hidden layer. The closed-form solution is derived from the model's convex training objective. ESAE was trained on batches of size 512.

For MetaRec-AE: the dimension of the encoder module is 500, the global regularization rate is 0.001, the local learning rate α is 0.000005, and the global learning rate β is 0.00005. MetaRec-AE was trained on batches of size 512.