Neural architectures for faster deep learning-based collaborative filtering

Chen, Yu

2021

Chen, Y. (2021). Neural architectures for faster deep learning-based collaborative filtering. Master's thesis, Nanyang Technological University, Singapore.
<https://hdl.handle.net/10356/152759>

<https://hdl.handle.net/10356/152759>

<https://doi.org/10.32657/10356/152759>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 11:18:03 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**NEURAL ARCHITECTURES FOR FASTER DEEP
LEARNING-BASED COLLABORATIVE
FILTERING**

CHEN YU

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

2021

NEURAL ARCHITECTURES FOR FASTER DEEP LEARNING-BASED COLLABORATIVE FILTERING

CHEN YU

School of Computer Science and Engineering

A Thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Master of Engineering

2021

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

28/09/2021

.....

Date

NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
.....

Chen Yu

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

09-28-2021

.....
Date

R

Sinno Jialin Pan

Authorship Attribution Statement

*(B) This thesis contains material from [1] paper(s) published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 3-5 is published as [Chen, Y., & Pan, S. J. \(2021\). Towards Faster Deep Collaborative Filtering via Hierarchical Decision Networks. Proceedings of the AAAI Conference on Artificial Intelligence, 35\(5\), 4001-4009.](#)

The contributions of the co-authors are as follows:

- I initiated the project proposal, developed the algorithm, and performed the experiments of the project.
- Prof Pan Provided supervision and guidance and supervision throughout the process of my research.
- I prepared the manuscript drafts. The manuscript was revised by Prof Pan.

28/09/2021

• • • • •

Date

[Handwritten signature]

Chen Yu

Acknowledgement

I would like to start to thank to my supervisor Dr Sinno Jianlin Pan for both his professional supervision as a research advisor, and wholehearted help as a life mentor.

I would like to thank my family for providing support and encouragement throughout the time, as well as Ms Dana Lim and Dr Tan for mental coaching to help me through rough times.

I would also like to thank all my colleagues, especially Mr Tianze Luo and Dr Longkai Huang for their great help and support throughout the years.

Contents

Acknowledgement	4
Abstract	iv
Acronyms	v
Symbols	vi
Lists of Figures	vii
Lists of Tables	viii
1 Introduction	1
1.1 Background	1
1.1.1 Recommendation System	1
1.1.2 The recommendation problem	1
1.1.3 Collaborative filtering	2
1.2 Deep learning for Collaborative Filtering	5
1.3 Major contribution of the Dissertation	7
1.3.1 Limitations of current deep learning based CF	7
1.3.2 Hierarchical Decision Network for faster deep learning based CF .	7
2 Literature Review	9
2.1 Neural Collaborative Filtering Framework	9
2.1.1 Outer Product-based Neural Collaborative Filtering	10

2.2	Approaches to Reduce Computation	11
2.2.1	Multiple Staged Recommendation	11
2.2.2	Re-ranking Based Method	11
2.2.3	Ranked Distillation	11
2.2.4	Quantization for Deep CF	12
3	Methodology	13
3.1	Hierarchical Decisions Network	14
3.1.1	Decision Module	14
3.1.2	Scoring Module	15
3.2	Training HDN: Decision Distillation	15
3.2.1	Distillation of the relevance score	16
3.2.2	Item Rejection as Binary Classification	17
3.3	Computational Cost for HDN Inference	18
4	Experiments, Results and Discussion	21
4.1	Experiment Data	21
4.1.1	Dataset descriptiton	21
4.1.2	Dataset preprocessing and splitting	22
4.2	Experiment Design	22
4.2.1	Test of design concept	23
4.2.2	Test on Inference Computational Cost Reduction	23
4.2.3	Test on Recommendation Quality	24
4.2.4	Test of distillation method	25
4.2.5	Experiment on number of decision modules	25
4.3	Implementation	26
4.4	Results and Discussion	27
4.4.1	Results on Item Rejection Rate	27
4.4.2	Results on Inference Time	28
4.4.3	Results on Recommendation Quality	30

4.4.4	Experiment on Distillation-based Training	30
4.4.5	Experiment on Number of Decision Modules N	32
5	Conclusion	34
5.1	Future work	34
5.1.1	Alternative Decision Module	34
5.1.2	Content Based Recommendation	35
5.1.3	Direct Training of HDN	35
5.1.4	Integration with other model compression methods	35
5.1.5	Explore model acceration methods with de-biased recommenda- tion systems	35
5.2	Author's list of publication	37

Summary

For a recommendation system where an algorithm recommend items to users, data is collected when the user interacts with the website/mobile application. The data collected could be the user information, including their demographic information, item information, the interaction data of what items the user interacted with, and the context information when the user interacts with a certain item. Collaborative filtering is a common method for recommendation systems, where only the past interaction data is considered, without extra information of the users, items, and interaction context.

Deep learning has been successful applied in multiple fields such as computer vision and natural language processing. It has also been utilized for collaborative filtering, and indicated promising results. However, some of the deep learning-based collaborative filtering methods require a lot of computational resources to perform recommendations, which would cost more time to recommend and increase the energy consumption.

In this thesis, a new deep learning architecture is developed for potential faster recommendation via collaborative filtering without sacrificing the recommendation quality. Extensive experiments have been performed to evaluate the performance and recommendation quality of our new neural architecture and training method.

Acronyms

Hierarchical Decision Network	HDN
Collaborative Filtering	CF
Normalized Discounted Cumulative Gain	NDCG
Hit Rate	HR
Matrix Factorization	MF
Neural Collaborative Filtering	NCF
Multi-Layer Perceptron	MLP
Neural Matrix Factorization	NeuMF
Convolutional Neural Collaborative Filtering	ConvNCF
Randed Distillation	RD

Symbols

u	user
v	item
$I_{u,v}$	past user interaction history
$\mathbf{h}_u$	user embedding
$\mathbf{h}_v$	item embedding
y_{u_t, v_t}	binary implicit feedback
$\hat{y}_{u_t, v_t}$	relevance score
$\gamma^{(n)}$	rejection ratio of layer n
N	number of decision modules in HDN

List of Figures

1.1	Comparison of factor based CF, representation learning based CF and interaction matching based CF	6
3.1	The Hierarchical Decision Network, where $\gamma^{(n)}$ represents the rejection ratio associated with the n -th decision module.	13
4.1	The actual rejection rate during distillation	27
4.2	Time comparison with limited batch size on MovieLens-20M	29
4.3	Timing and Performance when number of HDN modules changes	33

List of Tables

4.1	Recommendation performance measured by NDCG and HR when $K=20$	31
4.2	Recommendation performance measured by Recall when $K=20, 40$ and 60.	32
4.3	Result on MovieLens-20M and Yelp ($K = 20$). HDN-nr: remove item re- jection HDN-nr-: remove item rejection and rejection layer distillation HDN-nr-nd: remove all distillation and only train the score function . .	32

Chapter 1

Introduction

1.1 Background

1.1.1 Recommendation System

One of the major motivation of recommendation systems is that online business would like to recommend some items to their customers. In recommendation systems, an algorithm is developed to provide a list of items to the user, so that the user would potentially like the item recommended, and would be interested to buy it. In recent years, recommendation systems has been ubiquitous in e-commerce businesses, Internet content providers, advertisement agencies and so on.

For a recommendation system where an algorithm recommend items to users, data is collected when the user interacts with the website/mobile application. The data collected could be the user information, including their demographic information, item information, the interaction data of what items the user interacted with, and the context information when the user interacts with a certain item.

1.1.2 The recommendation problem

The recommendation problem involves users u and items v . Then the aim is to make use of past interactions $I_{u,v} = \{(u_0, v_0), (u_1, v_1), \dots, (u_n, v_n)\}$ (such as purchase, click,

rating) to prediction future interactions of a target user u_{target} and item i_{target} .

In some cases, we are given user and item attributes X_u and X_v , which contains information about the user (gender, age, etc.) and the item (content, price, color, etc.).

Besides user and item attributes, we are sometimes given information related with the interaction $X_{u,v}$ such as time of interaction, movie ratings and money spent.

Some recommendation problem are called cold-start problems of recommendation systems, works on cases where there are zero or few targeting users or items between past and future interactions. In this thesis, we are not dealing with the cold start problem, we assume that we have sufficient user-item interactions $I_{u,v}$ to make a recommendation.

Explicit and Implicit feedback

In some recommendation problems such as movie or restaurant reviews, user feedback are provided for each user-item interaction, usually in the form of ratings r_{u_t, v_t} . In some other cases, such as page-clicks, only the interactions themselves are given, without any other extra feedbacks. We still can model the interaction as a binary variable y_{u_t, v_t} , where $y_{u_t, v_t} = 1$ indicates that there exist an interaction between target user u_t and item v_t , and $y_{u_t, v_t} = 0$ indicates that there are no interactions.

1.1.3 Collaborative filtering

Collaborative filtering (CF) methods only rely on past user-item interaction $I_{u,v}$, without using any of the user/item features (X_u and X_v) and iteration specific features $X_{u,v}$. The main focus of the thesis is on collaborative filtering.

The two major approaches in classical CF methods are neighborhood based methods and factor based methods.

Neighborhood based models

In neighborhood based models, the recommendation algorithm is usually based on a similarity measure, the measure can either be user based (user-based methods) or item

based (item-based methods).

In user-based methods, recommendations are based on a user-user similarity measure $\text{Sim}(u_i, u_j)$. For a target user u_t and item v_t , we can find P_{u_t, v_t} , a set of neighborhood users of u_t that also interacted with the item v_t , and then we have:

$$\hat{y}_{u_t, v_t} = \sum_{u \in P_{u_t, v_t}} \text{Sim}(u, u_t) ,$$

where relevance score $\hat{y}_{u_t, v_t}$ is used to measure the relevance of the target user and item. The recommendation is then performed by finding the item with the highest relevance scores.

To obtain the similarity measure Sim , one of the ways is to use Pearson correlation coefficient or cosine distance on users, which are represented using the summation of all item vector IDs. The similarity also can be learned by regression over the past interaction or rating [1].

Item-based neighborhood recommendation methods are formulated in a similar way, but item similarity measures are used instead:

$$y_{u_t, v_t} = \sum_{v \in Q_{u_t, v_t}} \text{Sim}(v, v_t) ,$$

where Q_{u_t, v_t} represent items owned by u_t that are closed to v_t using the similarity measurement.

Classical factor based models / Matrix factorization

Another common approach of collaborative filtering is factor based CF. The idea behind factor based models is to assume a latent space representation of user and item $\mathbf{h}_u$ and $\mathbf{h}_i$, and the relevance score can be approximated by the inner product of the target user and item:

$$\hat{y}_{u_t, v_t} = \langle \mathbf{h}_{u_t}, \mathbf{h}_{v_t} \rangle \tag{1.1}$$

By using the past interaction data I , we can learn the projections from u_i to $\mathbf{h}_{u_i}$ and from v_i to $\mathbf{h}_{v_i}$ by minimizing the difference of relevance score and actual user feedback:

$$\underset{\mathbf{h}_u, \mathbf{h}_v}{\operatorname{argmin}} \sum_{(u_i, v_i) \in I} \ell(y_{u_i, v_i}, \hat{y}_{u_i, v_i}), \quad (1.2)$$

where u_i and v_i are identifiers of the relevant user and item.

When represented in matrix form, the past interaction is formulated as a matrix $\mathbf{I}_{(u,v)}$ and the user and item embeddings are put in matrices $\mathbf{H}_u$ and $\mathbf{H}_v$, we have the matrix factorization formulation of factor-based CF.

$$\mathbf{I}_{(u,v)} = \mathbf{H}_u \mathbf{H}_v^\top, \quad (1.3)$$

In some recommendation literature all kinds of unconstrained factorization methods are also referred as SVD, but most methods do not require orthogonal constraints [1].

Instead of directly project user IDs to vectors, NOSED [2] utilizes similar idea from the neighborhood methods and represent user as summation of latent factors of items it interacted:

$$\mathbf{h}_u = \sum_{v \in Q_u} \mathbf{h}_v, \quad (1.4)$$

where Q_u represent the set of items that user u has interacted. In this way, a user is only represented by its items, similar to item-based neighborhood methods.

Extending the idea of NSVD, SVD++ [3] model has a hybrid take and combined factor-based and NSVD item-based modeling by adding back a factor term for user $\mathbf{h}_{u0}$:

$$\mathbf{h}_u = \mathbf{h}_{u0} + \sum_{v \in Q_u} \mathbf{h}_v.$$

This recommendation method has won the Netflix Challenge [4], which is a famous competition where participants create different recommendation algorithms to compete for best recommendation quality.

1.2 Deep learning for Collaborative Filtering

Before the emergence of deep learning, a typical machine learning application is separated as two steps: feature engineering and apply a learning algorithm using the features. Deep learning, which are typically large scale deep Artificial Neural Networks, has more representational power and can deal with some raw features without feature engineering [5]. It has been successful applied in various domains from computer vision, speech recognition to natural language processing and robotics.

Deep learning has also been applied in the field of recommendation systems, including CF methods. Deep learning based CF has been majorly explored in two directions: representation learning and matching function learning.

Both of the methods are based on factor based CF, but utilized deep learning in different places.

As shown in Figure 1.1, in representation learning based methods [6, 7], the embeddings of users and items are learned through a deep neural network. After the representations are learned, the inner product between the embeddings of users and items is used to compute the user-item relevant score for recommendation:

$$\mathbf{h}_{u_t} = \text{NeuralNetwork}(u_t), \mathbf{h}_{v_t} = \text{NeuralNetwork}(v_t) \quad (1.5)$$

$$\hat{y}_{u_t, v_t} = \langle \mathbf{h}_{u_t}, \mathbf{h}_{v_t} \rangle. \quad (1.6)$$

In this way, computation in the inference (i.e., recommendation) phase is efficient, but the interactions between the embeddings of users and items are modeled linearly as the inner product is a linear operation.

To address the limitation of representation learning CF approaches, matching function learning has been proposed to capture nonlinear interactions between the embeddings $\mathbf{h}_u$ and $\mathbf{h}_v$ beyond the inner product by designing a deep architecture. The Neural Collaborative Filtering (NCF) framework [8] is a representative in the direction of

Figure 1.1: Comparison of factor based CF, representation learning based CF and interaction matching based CF

matching function learning:

$$\hat{y}_{u_t, v_t} = \text{NeuralNetwork}(\mathbf{h}_{u_t}, \mathbf{h}_{v_t}). \quad (1.7)$$

Various neural network architectures have been proposed to model user-item interactions under the NCF framework [9, 10].

1.3 Major contribution of the Dissertation

1.3.1 Limitations of current deep learning based CF

For representation learning based deep CF methods, the linear matching limits the nonlinear interactions between user and items. For interaction matching based deep CF methods, the potential computation cost of performing a recommendation is very high. Intuitively, if the deep neural network has a deeper architecture, it is able to capture deeper interactions between users and items. Empirically, it has also been shown in previous work that the depth of the deep architecture has a proportional effect on the recommendation quality [8]. However, the depth of the architecture is a double-edged sword. On one hand, a deeper architecture makes capturing deeper interactions between users and items for more accurate recommendation possible, on the other hand, a deeper architecture causes higher computation cost in real-time inference, which makes it less practical in real time. To be specific, to recommend items to a target user, most state-of-the-art deep learning-based methods require a full forward pass to retrieve a rating score for every pair of the user and all the items. When the item set is very large, this inference or recommendation process is highly time intensive, especially when the recommender system runs on a device with limited computational resources.

1.3.2 Hierarchical Decision Network for faster deep learning based CF

To address the aforementioned issue of deep CF methods, in this thesis, we propose a novel network architecture, Hierarchical Decision Network (HDN). In the designed architecture as shown in Figure 3.1, the network consists of multiple stacked decision module and a final scoring module. The goal of a hierarchical decision module is to provide an early decision on whether the item should be rejected (i.e., not recommended to the user) or need to be further passed to the next decision module or to the scoring module to compute the final relevant score for recommendation. Different from previous deep CF networks, the proposed HDNs do not only recommend items based on

relevance scores, but also utilizes decision modules to quickly decide whether an item should be further “investigated” for recommendation. By using the stacked decision modules, HDNs are able to quickly filter a large portion of items that are most unlikely matched with the user’s preferences before forwarding them to the next layers for further computation. As a result, for each user, there may be only a few items that are finally passed to the scoring layer to compute relevant scores. As a result, HDNs do not only save computation time in forward computation, but also save the time for ranking item candidates for each user.

In addition to the new proposed architecture, we also develop a distillation-based training algorithm to train a HDN. In deep learning, knowledge distillation [11] is a technique which aims to transfer knowledge from a large and well-trained teacher network to train a smaller student network. In CF, the key knowledge is the ranking information of items for each user. Therefore, we first train a deep and precise neural network by using a deep CF method as the teacher network, and consider a HDN as the student network. The outputs of the teacher network, i.e., scores of user-item pairs, are distilled to guide the learning of the rejection thresholds of each decision modules.

This thesis consist of the proposal of a novel deep learning architecture HDN for faster deep learning based recommendation, a distillation-based training algorithm for knowledge transfer, and experiments on real life datasets on both the computational cost and recommendation performance of HDN. Chapter 2 includes a literature review of relevant works. Chapter 3 will describe the HDN architecture and the distillation training algorithm, as well as a theoretical analysis of the computational cost reduction of HDN. Chapter 4 will introduce the experiments, and the results are analysed in Chapter ??, following a conclusion in Chapter 5

Chapter 2

Literature Review

2.1 Neural Collaborative Filtering Framework

Recall in Chapter 1, in classical matrix factorization-based CF methods, given a pair of a user and an item (u_t, v_t) , the relevance score $\hat{y}_{u_t, v_t}$ is obtained by computing the inner product between their embeddings $\mathbf{h}_{u_t}$ and $\mathbf{h}_{v_t}$:

$$\hat{y}_{u_t, v_t} = \langle \mathbf{h}_{u_t}, \mathbf{h}_{v_t} \rangle. \quad (2.1)$$

The idea of the Neural Collaborative Filtering (NCF) framework [8] is to use a neural network to replace the inner product, such that deeper interactions between the user-item pair can be captured:

$$\hat{y}_{u_t, v_t} = \text{NeuralNetwork}(\mathbf{h}_{u_t}, \mathbf{h}_{v_t}). \quad (2.2)$$

Recently, various network architecture have been proposed based on the NCF framework [6, 8–10]. For instance, in [8], a neural matrix factorization network (NeuMF) is

proposed as follows:

$$\hat{y}_{u_t, v_t} = \mathbf{w}^\top \begin{bmatrix} \mathbf{h}_{u_t} \odot \mathbf{h}_{v_t} \\ \text{MLP} \left(\begin{bmatrix} \mathbf{h}_{u_t} \\ \mathbf{h}_{v_t} \end{bmatrix} \right) \end{bmatrix}, \quad (2.3)$$

where $\odot$ is the element-wise product and $\mathbf{h}_{u_t} \odot \mathbf{h}_{v_t}$ is to model the linear or nonlinear interaction between $\mathbf{h}_{u_t}$ and $\mathbf{h}_{v_t}$ based on Generalized Matrix Factorization, and MLP denotes one or more Multiple Layer Perceptron layer(s) to capture “deep” interactions between $\mathbf{h}_{u_t}$ and $\mathbf{h}_{v_t}$. The learnable parameter $\mathbf{w}$ is introduced to generate the relevant score $\hat{y}_{u_t, v_t}$.

2.1.1 Outer Product-based Neural Collaborative Filtering

In [9], deep networks are designed based on “interaction maps”, which are the outer products of the user and the item embeddings $\mathbf{h}_{u_t} \otimes \mathbf{h}_{v_t}$. For instance, ConvNCF [9] employs a 2D Convolutional Neural Network (CNN) on the “interaction maps”, and uses the inner product between the vector generated by the CNN and a parameter vector to generate the final score:

$$\hat{y}_{u_t, v_t} = \mathbf{w}^\top \text{CNN}(\mathbf{h}_{u_t} \otimes \mathbf{h}_{v_t}), \quad (2.4)$$

where CNN may contain multiple convolutional layers. For inference or making recommendation for each target user, one needs to first compute outer product between the user embedding and the embeddings of every item, and pass them through the CNN to compute a relevant score for each item for the user. Along with the deeper network, it results in more computational cost for inference.

2.2 Approaches to Reduce Computation

2.2.1 Multiple Staged Recommendation

One common approach in the industry is to reduce the computational cost by using a multiple-step process [12], which has an extra candidate generation step before the ranking step. Items that are likely to be interacted by a certain user are pre-selected in the candidate generation step. An extra model is built for fine-grained user preference modeling, which provides scores to re-rank the selected items. This two-stage approach can avoid running a large model for a large batch of items. However, it does not enable end-to-end training of the network, where the network training information does not flow from each other, and requires extra manual tuning of the two steps.

2.2.2 Re-ranking Based Method

Re-ranking based methods improve the two-stage method by using the ranking information of the first stage to help the model in the second stage. A deep learning-based re-ranking model, PRM, has been proposed in [13]. PRM is an attention based deep network to re-rank the items after candidate generation. It has performance improvement over two-staged process without re-ranking as well as previous re-ranking methods. Although the re-ranking methods have improvements on the second fine-grained step, the potential loss of items from the first step could not be recovered.

2.2.3 Ranked Distillation

In [14], a distillation-based method named Ranking Distillation (RD) is proposed for ranking problems, which utilizes the top-K ranked items from the teacher model for distillation. Comparing to RD, our work does not focus on general knowledge transfer for learning-to-rank tasks, but one tailored to our designed network architecture. Our distillation-based method for the decision units also allows user preference information for the lower-ranked items transferred from the teacher model, besides the top-K

items from RD. For the scoring unit, we do not use ranking-based distillation, the combination of our work and RD could be possible for further exploration.

2.2.4 Quantization for Deep CF

Quantization [15] of neural network is also a common approach to reduce the computation cost. In [16], the authors have applied the quantization method, KD encoding [17], on the embeddings layer for both users and items. Their proposed method MGQE has achieved performance that matches and sometimes improves over original models. Although quantization reduces the storage space of the user and item embeddings, all the user-items pairs are still passed through the whole neural network, which still requires the same computational resources. In [18], the authors used a codebook-based approach to represent items as a combination of the closest codeword. The quantization of the representation enables smaller storage space and faster inference, while sacrificing representation power compared with the deep learning based user-item matching methods. In addition, ideas from both of the above quantization methods can also be added to our proposed network by altering the embedding layer.

Chapter 3

Methodology¹

We have developed a novel neural network architecture, Hierarchical Decision Network (Figure 3.1), for faster deep collaborative filtering.

Figure 3.1: The Hierarchical Decision Network, where $\gamma^{(n)}$ represents the rejection ratio associated with the n -th decision module.

¹The work in this chapter has been published in Chen, Y., & Pan, S. J. (2021). Towards Faster Deep Collaborative Filtering via Hierarchical Decision Networks. Proceedings of the AAAI Conference on Artificial Intelligence, 35(5), 4001-4009.

3.1 Hierarchical Decisions Network

An HDN consists of N decision modules and a scoring module. Similarly to classical factor-based CF, in a HDN the user and the item are first embedded into vectors as $\mathbf{h}_u$ and $\mathbf{h}_v$, respectively, and then passed to the first decision module, followed by subsequent decision modules and the scoring module if necessary.

3.1.1 Decision Module

For a target user u_t and an item v_t , their embeddings $\mathbf{h}_{u_t}$ and $\mathbf{h}_{v_t}$ are passed to the first decision model $D^{(1)}$, which outputs a decision $\hat{d}_{\text{rej}}$ of whether to filter or reject this item for the user, and a user-item interaction embedding $\mathbf{h}^{(1)}$:

$$[\hat{d}_{\text{rej}}^{(1)}, \mathbf{h}^{(1)}] = D^{(1)}(\mathbf{h}_{u_t}, \mathbf{h}_{v_t}). \quad (3.1)$$

If $\hat{d}_{\text{rej}} > 0$, then the item is rejected and the pair of embeddings will not be passed to the subsequent module. Otherwise, the pair of embeddings $\mathbf{h}_{u_t}$ and $\mathbf{h}_{v_t}$ together with the generated interaction embedding $\mathbf{h}^{(1)}$ will be passed to the second layer. A subsequent decision module $D^{(n)}$, where $1 < n \leq N$, takes the user and the item embeddings, $\mathbf{h}_{u_t}$ and $\mathbf{h}_{v_t}$, and the interaction embedding $\mathbf{h}^{(n-1)}$ generated by $D^{(n-1)}$ as input and outputs the rejection decision and a new interaction embedding for the next layer:

$$[\hat{d}_{\text{rej}}^{(n)}, \mathbf{h}^{(n)}] = D^{(n)}(\mathbf{h}_{u_t}, \mathbf{h}_{v_t}, \mathbf{h}^{(n-1)}). \quad (3.2)$$

In the HDN architecture, one may design various architecture for the decision module D , e.g., design a specific architecture for certain applications. In this paper, as our focus is to verify the effectiveness of main architecture of HDNs, we implement a simple decision block design for the decision module. To be specific, we first aggregate the user and the item embeddings by using the element-wise product, $\mathbf{h}_{u_t} \odot \mathbf{h}_{v_t}$, and then concatenate it with the user-item interaction embedding $\mathbf{h}^{(n-1)}$ from the previous layer

to generate a unified vector:

$$\mathbf{h}_{\text{uni}}^{(n)} = \begin{bmatrix} \mathbf{h}_{u_t} \odot \mathbf{h}_{v_t} \\ \mathbf{h}^{(n-1)} \end{bmatrix}. \quad (3.3)$$

The rejection decision is obtained by computing the inner product between the unified embedding $\mathbf{h}_{\text{uni}}^{(n)}$ and a weight vector $\boldsymbol{\theta}$:

$$\hat{d}_{\text{rej}}^{(n)} = \boldsymbol{\theta}_{\text{rej}}^{(n)\top} \mathbf{h}_{\text{uni}}^{(n)}. \quad (3.4)$$

And the interaction embedding is obtained by passing through a fully-connected layer:

$$\mathbf{h}^{(n)} = a \left(W^{(n)} \mathbf{h}_{\text{uni}}^{(n)} \right), \quad (3.5)$$

where $a(\cdot)$ is the activation function. In this work, we adopt RELU [5] for $a(\cdot)$.

3.1.2 Scoring Module

The scoring module takes $\mathbf{h}_{u_t}$, $\mathbf{h}_{v_t}$ and the interaction embedding from the last decision layer $\mathbf{h}^{(N)}$ as input, and outputs the relevance score $\hat{y}_{u_t, v_t}$ via

$$\hat{y}_{u_t, v_t} = W_{\text{score}} \begin{bmatrix} \mathbf{h}_{u_t} \odot \mathbf{h}_{v_t} \\ \mathbf{h}^{(N)} \end{bmatrix}, \quad (3.6)$$

which is similar to the design of the decision module.

3.2 Training HDN: Decision Distillation

For all the decision modules at different layers, we aim to obtain a decision on whether certain items for the target user should be rejected. Since there is a trade-off between speed and accuracy for different application scenarios, we introduce the rejection ratio $\gamma^{(n)}$, which is the ratio of rejected items to all the items being passed to the decision

module $D^{(n)}$. A higher rejection ratio enables less computation as less items are passed to the subsequent modules, while a lower rejection ratio may enable better recommendation accuracy as the chance of filtering relevant items becomes smaller.

However, it requires very intensive manual loss tuning to reach the desired rejection ratio for early decision modules, which is undesirable. Alternatively, a reinforcement learning-based algorithm may work, where rewards can only be computed at the scoring layer and the decision ratio updates are defined as actions, but its training process is expensive. Instead, here, we propose a distillation-based algorithm, which considers a HDN as a student network, and aims to transfer ranking information from a teacher network, i.e., a precise deep CF model, to help training the student network. In this way, it becomes possible to calculate the direct loss of each decision module. In our distillation-based algorithm, any existing CF method (including black-box models) can be used as the teacher network as long as it is able to output a relevance score for a given user-item pair.

3.2.1 Distillation of the relevance score

Similar to knowledge distillation [11], all possible pairs of $\mathbf{h}_u$ and $\mathbf{h}_v$, including those having historical interactions (a.k.a., positive instances) and those having no historical interactions (a.k.a., negative instances), can be sampled to be fed to the student and the teacher networks for training. For each sampled user-item pair (u_t, v_t) , the teacher and the student networks compute a relevant score t_{u_t, v_t} and $\hat{y}_{u_t, v_t}$, respectively. The difference of distribution between them measured via the Kullback-Leibler(KL) divergence is used as the loss. For rating recommendation problems, the ratings can be relaxed to a continuous score that follows a Gaussian distribution. Let the scores of the teacher model and the student network be in $p \sim \mathcal{N}(\mu, \sigma)$ and $\hat{p} \sim \mathcal{N}(\hat{\mu}, \hat{\sigma})$, respectively. The loss of the final score is:

$$\ell_s = \text{KL}(p, \hat{p}) = \log \frac{\hat{\sigma}}{\sigma} + \frac{\sigma^2 + (\mu - \hat{\mu})^2}{2\hat{\sigma}^2} - \frac{1}{2} \quad (3.7)$$

For implicit feedback cases, we model a relevance score as the probability p of a Bernoulli variable by letting $p = \sigma(t)$ and $\hat{p} = \sigma(\hat{y})$, where σ is the sigmoid function. The loss ℓ_s is:

$$\sum_{u_t \in U, v_t \in V} p_{u_t, v_t} \log \hat{p}_{u_t, v_t} + (1 - p_{u_t, v_t}) \log(1 - \hat{p}_{u_t, v_t}), \quad (3.8)$$

where U and V denote all the users and the items in the historical interaction set, respectively.

3.2.2 Item Rejection as Binary Classification

For a user u_t , denote by $\hat{d}_{v_t} > 0$ a rejection of the item v_t and by $\hat{d}_{v_t} \leq 0$ not rejecting the item. Since we hope to keep a rejection ratio $\gamma^{(n)}$ for all the users at the decision module $D^{(n)}$, the number of decisions $\sum_{v_t \in V_t} \mathbb{I}(\hat{d}_{v_t}^{(n)} > 0)$ on layer n must satisfy the following:

$$\frac{\sum_{v_t \in V_t} \mathbb{I}(\hat{d}_{v_t}^{(n)} > 0)}{|V_t|} = \gamma^{(n)}, \quad (3.9)$$

where V_t is the set of all the items in the query.

For each target user, during the distillation training process, the teacher can provide the final relevant score κ_{v_t} for each item candidate v_t . However, in order to obtain a reference decision $d_{v_t, \gamma} \in \{-1, +1\}$ from the teacher network, where “+1” denotes a rejection, while “-1” denotes acceptance, we need to define a threshold score $\bar{\kappa}_\gamma$ for each rejection ratio γ . In this work, the threshold score $\bar{\kappa}_\gamma$ is obtained by ranking the scores for all items from the teacher network, and setting the value to be the score of the K -th top item, where

$$K = \sum_{v_t \in V_t} \mathbb{I}(\hat{d}_{v_t} < 0) = (1 - \gamma)|V_t| \quad (3.10)$$

This is done independently for each user, and each user shares the same threshold score for all the decision modules.

After obtaining the threshold score, a hinge loss is used for the binary decisions for user-item pairs:

$$\ell_{\mathbf{D}^{(n)}, u_t, v_t} = \max(0, 1 - \tilde{d}_{u_t, v_t}^{(n)} d_{v_t, \gamma^{(n)}}), \quad (3.11)$$

$$\text{where } d_{v_t, \gamma^{(n)}}^{(n)} = \begin{cases} 1, & \kappa_{v_t} \geq \bar{\kappa}_{\gamma^{(n)}} \\ -1, & \kappa_{v_t} < \bar{\kappa}_{\gamma^{(n)}} \end{cases} \quad (3.12)$$

By combining the decision and the scoring loss, we have the final loss for decision distillation of HDNs as follows,

$$\ell = \ell_S + \sum_{u_t \in U, v_t \in V} \sum_{n=1}^N \ell_{\mathbf{D}^{(n)}, u_t, v_t}. \quad (3.13)$$

All the parameters of a HDN can be learned by minimizing (3.13) by gradient decent. The overall training algorithm is summarized in Algorithm 1.

3.3 Computational Cost for HDN Inference

To analyze the computational cost reduction of HDN for deep CF based recommendation, we assume that a general deep network with M layers has an average computational cost of C FLOPs (number of floating-point operations). For T user-item pairs to recommend, the computational cost is for inference of the deep neural network is in $O(CMT)$.

For simplicity for comparison analysis, we assume HDN has the same number of layers as the teacher network, i.e., $N + 1 = M$ (a HDN has N decision layers and 1 scoring layer). In practice, N can be smaller than M . When using HDN with a fixed rejection rate γ , since the items are rejected by previous layers, the computational cost

Algorithm 1 Distillation based training on HDN

Given teacher model $\mathcal{T}$, Set of all users and items U and V , Interaction data D , HDN student model $\mathcal{S}$ with $\gamma^{(n)}$

Train $\mathcal{T}$ using D

Obtain threshold score $\bar{\kappa}_\gamma$

Initialize $\mathcal{S}$ and $\mathbf{h}_u, \mathbf{h}_v$ for all user and items

while epoch < MAX **do**

 Sample user u and item v from D

$p = \sigma(\mathcal{T}(u, v))$

$n = 1$

while $n < N$ **do**

 Calculate $d_{v_t}^{(n)}$ using $\mathcal{T}$ and $\gamma^{(n)}$

$(\hat{d}_{\text{rej}}^{(n)}, \mathbf{h}^{(n)}) = \mathbf{D}^{(n)}(u, v, \mathbf{h}^{(n-1)})$

$\ell_{\mathbf{D}^{(n)}} = \max(0, 1 - \hat{d}^{(n)} d_{v_t, \gamma^{(n)}})$

$n++ = 1$

end while

$\hat{p} = \sigma(\mathcal{S}(u, v, \mathbf{h}^{(n)}))$

 Construct $\ell_{\mathcal{S}}$ based on (3.7) or (3.8)

$\ell = \ell_{\mathcal{S}} + \sum_{u_t \in U, v_t \in V} \sum_{n=1}^N \ell_{\mathbf{D}^{(n)}, u_t, v_t}$

 gradient decent on $\nabla \ell$

 epoch += 1

end while

for layer k is

$$(1 - \gamma)^{k-1} CT. \quad (3.14)$$

Let $\alpha = 1 - \gamma$. The total inference cost of HDN is

$$CT + \alpha CT + \alpha^2 CT + \dots + \alpha^M CT, \quad (3.15)$$

which can be simplified as $\frac{CT(1-\alpha^{M+1})}{1-\alpha}$. Let $f(\alpha) = \frac{1-\alpha^{M+1}}{1-\alpha}$, we can observe that $f(\alpha)$ is

monotonically increasing from 0 to 1. When $1 > \gamma \geq 0.5$, i.e. $0 < \alpha \leq 0.5$, we have

$$f(\alpha) \leq \frac{1 - 0.5\alpha^{M+1}}{1 - 0.5} = 2 - \alpha^{M+1} < 2. \quad (3.16)$$

Hence the computational cost of HDN is

$$\frac{CT(1 - \alpha^{M+1})}{1 - \alpha} < 2CT, \quad (3.17)$$

given a rejection rate γ that is larger than or equal to $\frac{1}{2}$.

In summary, for any deep neural network with any cost of C per layer, the final computation cost is proportional to the number of layers, which is in $O(CMT)$. In HDN, due to the rejection of the items, although having the same depth and cost per layer, the computation cost is reduced to $O(2CT) = O(CT)$, which is irrelevant to the depth of the network. By using HDN, the network can enjoy the benefit of the learning power of deep networks without suffering from the cost when performing recommendations.

Chapter 4

Experiments, Results and Discussion¹

After we formulated HDN as a new network architecture to reduce the inference computational cost, several experiments were designed to test if the architecture is able to perform as expected.

4.1 Experiment Data

4.1.1 Dataset description

The experiments are performed in the following datasets:

- *MovieLens*: We have selected MovieLens [19] for testing the recommendation performance of our model due to its popular use in the recommendation system research.

MovieLens is a set of datasets of movie ratings of different size. Both the 1 million and 20 million datasets are used for evaluation of performance on datasets of different scales. MovieLens-1M contains 1 million movie ratings by 6,040 users and 3,760 movies as items, and MovieLens-20M contains 20 million interactions of 138,493 users and 26,744 items.

¹The work in this chapter has been published in Chen, Y., & Pan, S. J. (2021). Towards Faster Deep Collaborative Filtering via Hierarchical Decision Networks. Proceedings of the AAAI Conference on Artificial Intelligence, 35(5), 4001-4009.

- *Yelp*: The Yelp dataset is originally published on Yelp Dataset Challenge. It consists of user reviews on restaurants. We have selected Yelp as an example of a sparse dataset, it has 25,677 users and 25,815 reviews ratings as items, with only 730,790 user-item interactions.
- *Pinterest*: The Pinterest dataset is constructed by [20] using data from `pinterest.com`, when a user ‘like’ an image on the website, it is counted as an interaction. We have selected the dataset as a different type of recommendation task. There are 55,187 users, 9,916 items and 1,500,809 interactions, in this dataset.

4.1.2 Dataset preprocessing and splitting

We followed the same setups for the NCF paper [8], where only the implicit feedbacks are used in all datasets. For datasets with ratings, if an item is rated by a user, it is labeled as 1, otherwise, it is labeled as 0. The actual ratings are not used in our experiments.

For all the datasets, we sampled 20% of the items of each user as testing-set from the original interaction data, the rested are used as training-set. In order to evaluate the collaborative filtering performance and avoid the accidentally cold-start scenario due to arbitrary sampling, the sampled items that only appears in testing-set are moved back to the training-set. We first trained the teacher model using the training-set, then perform our distillation method to train HDN with a well-trained teacher network. The testing is then performed on the testing-set for both the teacher method and our method.

4.2 Experiment Design

We investigated the performance of HDN in four experiments to verify if HDN has the capability to reject unrelated items in the hierarchical structure. We would like first to know if HDN is rejecting as intended, then test if HDN has actually reduced the inference time and maintained the recommendation quality. Then we examine if the designs in the distillation helps HDN train and if varying number of modules affect HDN performance.

4.2.1 Test of design concept

The main motivation of the design of HDN is to reduce the number of items for a user, and the rejection ratio γ has been set to control the rejection rate. Different $\gamma^{(n)}$ would affect the trade-off between the inference time and recommendation accuracy. Therefore, knowing whether HDN rejects the same ratio as γ is the first step to know HDN is behaving as expected.

The experiment is designed to record the ratio of items that are kept in HDN throughout the training process, and observe if the ratio on each layer n will converge to the desired reject rate $\gamma^{(n)}$.

4.2.2 Test on Inference Computational Cost Reduction

Besides our analysis in the Chapter3, we would like to show the theatrical reduction of inference computation cost in real life. If the inference is done sequentially, the computation cost can be measured as total compute time. However, for deep learning based applications, the computation is usually done in parallel, where the computational cost reflects on both computational time and memory. Testing the recommendation sequentially for simplicity is unrealistic as it is not a common practice, and it would take too much time for large datasets. It is also not realistic to consider computing with infinite memory, in which case, all the recommendations could be done in parallel. One way to measure the cost is to simulate the real-life scenario, where there is limited memory, and compare the inference time.

For most deep learning applications, the memory constraints can be reflected on the maximum batch size of the input data for inference. But the actual cost of parallel inference in HDN is hard to measure, as the memory cost varies with time. In the case of HDN, the computation cost reduction is hard to measure, since the batch size reduces during the inference due to item rejection, and would be ignored. However, when a whole batch is rejected at one of the rejection layers, the inference time will be reduced for this batch. In our experiment, we first sort all items by popularity to enable more

batches for early rejection. And we would like to show in this section that, with this benefit alone, HDN can achieve faster inference time.

The item sorting only needs to be done once for all users in a very short time ($< 1s$), they are omitted from the results. After that, the whole dataset is divided into many batches which are then passed through HDN and NeuMF to perform recommendation. Finally, the inference time of the whole dataset is measured. Since the recommendation process involves both scoring and selecting the top-K items, we have measured the time consumed for both procedures. We have used the `topk` function in PyTorch to select the top K items, which is faster than sorting the whole item list.

4.2.3 Test on Recommendation Quality

Next, we evaluate the recommendation quality of HDN compared with teacher model to observe whether there would be loss of performance by rejecting items early.

To test whether our proposed hierarchical architecture is effective on capturing deep user-item interactions, a small network with dot product of user and item embedding (MF student) is also used as a student model and trained using Knowledge Distillation. Since a two-stage process [12] can also filter out items prior to fine-tuned ranking, we would also like to compare it with HDN. We simulate the process by using Matrix Factorization to select the top 100 items and then rank them using NeuMF. We have further included popRank, where only the item popularity is included for ranking, and Matrix Factorization(MF) as baselines.

To test the performance, we first retrieve the top ranked items from the models, and then measure the Hit Ratio(HR) and Normalized Discounted Cumulative Gain(NDCG) [21] for both models.

To evaluate the recommendation quality, one of the common way is to use Hit Ratio(HR), which measures the ratio of whether a positive test item is in the top-K ranking list.

Another way to evaluate how the recommended items ranked compared with true preferences of the users. Discounted cumulative gain (DCG) and normalized discounted

cumulative gain (NDCG) [22] are measures of ranking quality.

The definition of $\text{NDCG}@k$ is:

$$\text{NDCG}_K = \frac{\text{DCG}_K}{\text{IDCG}_K}, \quad (4.1)$$

$$\text{where } \text{DCG}_K = \sum_{i=1}^K \frac{\mathbb{I}(i \in \text{positive target items})}{\log_2(i+1)} \quad (4.2)$$

$$\text{IDCG}_K = \sum_{i=1}^K \frac{1}{\log_2(i+1)} \quad (4.3)$$

4.2.4 Test of distillation method

We further investigated the effectiveness of our distillation method by training HDN differently. If trained without distillation, HDN cannot maintain the fixed rejection ratio γ , and the rejection layers would not be trained to reject items as desired. In this experiment, we put item rejection aside first and test the result while passing all items through all layers HDN (HDN-nr). We then removed the information for distillation from the teacher model by training the rejection layers using the implicit feedback (HDN-nr-). Finally, we then removed the distillation process completely, ignoring all the item rejections and only train on the score function (HDN-nr-nd). We have tested all the variations above on MovieLens-20M to observe how distillation helps the training of HDN.

4.2.5 Experiment on number of decision modules

Last but not least, to test the robustness of the recommendation quality and computational cost reduction of HDN, we have varied the number of rejection modules N , and observed how it affects inference time and recommendation quality. The experiment is performed on HDN with $N \in 2, 3, 4, 5$ modules on MovieLens-20M. In this experiment, we kept $\gamma^{(n)} = 0.5$ for rejection modules.

4.3 Implementation

We implemented HDN and the baseline method using PyTorch [23]. Gradient descent of minibatch size of 512 and Adam [24] optimizer with a learning rate of 0.001 was used. 50 epochs of distillation training were performed for all datasets. The timing experiments were performed in an NVIDIA DGX-1 machine.

In the experiments, if not specified, the HDN that we have constructed has $N = 3$ decision modules and RELU is used as activation unit between modules. We have used a factor size of 32 for user and item embedding in both the teacher model and HDN, and 32 for user-item embedding $\mathbf{h}^{(n)}$ in all the modules.

We have used NeuMF and ConvNCF as teacher models for our distillation-based training on HDN. For both the teacher models, we have trained using the same parameters and the pretraining technique mentioned [8, 9].

Although our distillation model allows training using any possible teacher-student pair, it would be very time-consuming. Instead, we used a scheme similar to negative sampling during distillation, and use the score information for negative samples from the teacher model. We used a positive to negative ratio of 1 : 5 to train the HDN in our experiments.

4.4 Results and Discussion

4.4.1 Results on Item Rejection Rate

We tested the actual rejection ratio versus the proposed γ to see if HDN was actually rejecting the amount of item requested in all datasets. The results from MovieLens-1M with different rejection ratios are shown as an example in Figure 4.1. From the figure

Figure 4.1: The actual rejection rate during distillation

we can see that in the two different cases, HDN all started with a random average rejection ratio and converged to the desired γ set before within 5 episodes of training. This indicates that HDN has indeed rejected items as expected when performing inference.

4.4.2 Results on Inference Time

We tested the inference time in a large dataset MovieLens-20M, and vary the batch size to test the inference time with different batch-sizes. In Figure 4.2a, the inference time comparison of HDN and teacher models are shown, where HDN1/2/3 represent the time to pass each HDN rejection module. Figure 4.2b shows the time for ranking all the items, and 4.2c shows the total time used for getting the top-K items. The results in Figure 4.2 indicate that HDN retrieve the recommendations much faster than both NeuMF and ConvNCF for both inference and ranking time, especially when there is more memory limitation(smaller batch size). The cost of each rejection module is also shown, where the subsequent layers require significantly less computational time due to the rejection of the items.

Since this measurement does not take into account the batch size reduction during HDN inference, more fine-grained control over the parallelization other than sequential batches could potentially yield further inference time improvements.

(a) Inference time

(b) Ranking time

(c) Total time

Figure 4.2: Time comparison with limited batch size on MovieLens-20M

4.4.3 Results on Recommendation Quality

We have experimented on all three datasets, and found that our model has maintained performance with both the teacher networks. The results are consistent for all K s (number of top items used for evaluation) that we have tested, and the result when $K = 20$ is presented in Table 4.1 as an example.

Both the teacher models (NeuMF, ConvNCF) have higher NDCG and HR than the baselines. As a more complex model, ConvNCF yields better results on larger datasets (ML20M).

For both the teacher models, HDN maintains the recommendation quality for all the four datasets that we have tested. The results are significantly better than the baseline distilled model (MF student). HDN also has consistently better results compared with the simulated two-stage model.

Since HDN has rejected items early, some potential positive interactions may be marked as we increase K to test if HDN would suffer from low recall:

In table 4.2, we can see that HDN can maintain the performance when $K=20$ and $K=40$. Lower recall are observed for some datasets when $K=60$. This indicates that the rejection modules of HDN can will not affect the recall of the recommendation system when K is not large, which is quite sufficient for a lot of recommendation applications.

4.4.4 Experiment on Distillation-based Training

As seen the results in Table 4.3, the performance of HDN is not as good without rejecting items (HDN-nr), and has significant degradation when the distillation process is replaced (HDN-nr-nd and HDN-nr-). The slight decrease in performance of HDN-nr is expected. Since the last layer is not designed to rank all items, it can be served as a baseline since it is trained with distillation. There is a further performance decrease when trained directly on implicit feedback only (HDN-nr-) compared with (HDN-nr). A possible reason is that the rejection layers could utilize the full item-ranking information from the teacher model, so that items could be rejected accurately, hence the

	ML1M		ML20M		Pinterest		Yelp	
	NDCG	HR	NDCG	HR	NDCG	HR	NDCG	HR
PopRank	0.354	0.757	0.323	0.690	0.031	0.080	0.050	0.125
MF	0.601	0.890	0.492	0.771	0.172	0.443	0.119	0.310
MF(top100) + NeuMF	0.613	0.901	0.529	0.779	0.177	0.446	0.127	0.327
NeuMF(teacher)	0.620	0.962	0.557	0.847	0.193	0.501	0.135	0.334
MF student	0.609	0.903	0.511	0.781	0.174	0.445	0.122	0.311
HDN	0.633	0.954	0.562	0.850	0.197	0.458	0.137	0.329
HDN	0.634	0.951	0.563	0.845	0.201	0.462	0.131	0.321
ConvNCF(teacher)	0.617	0.960	0.579	0.866	0.181	0.499	0.135	0.336
MF student	0.605	0.901	0.507	0.793	0.173	0.447	0.120	0.312
HDN	0.627	0.958	0.578	0.863	0.183	0.487	0.133	0.331
HDN	0.631	0.956	0.576	0.865	0.180	0.486	0.135	0.329

Table 4.1: Recommendation performance measured by NDCG and HR when K=20

Recall @K	ML1M			ML20M		
	20	40	60	20	40	60
PopRank	0.081	0.138	0.182	0.089	0.148	0.192
MF	0.236	0.349	0.436	0.172	0.255	0.313
NeuMF	0.247	0.367	0.450	0.229	0.331	0.398
HDN $\gamma = 0.5$	0.248	0.368	0.448	0.230	0.329	0.391
Recall @K	Pinterest			Yelp		
	20	40	60	20	40	60
PopRank	0.014	0.025	0.035	0.034	0.054	0.071
MF	0.119	0.197	0.241	0.070	0.129	0.169
NeuMF	0.126	0.206	0.269	0.082	0.134	0.175
HDN $\gamma = 0.5$	0.126	0.204	0.266	0.081	0.127	0.169

Table 4.2: Recommendation performance measured by Recall when K=20, 40 and 60.

	Yelp		ML20M	
	NDCG	HR	NDCG	HR
NeuMF	0.125	0.334	0.557	0.847
HDN	0.137	0.329	0.562	0.850
HDN-nr	0.131	0.324	0.556	0.844
HDN-nr—	0.128	0.319	0.549	0.841
HDN-nr-nd	0.120	0.308	0.530	0.818

Table 4.3: Result on MovieLens-20M and Yelp ($K = 20$).

HDN-nr: remove item rejection

HDN-nr—: remove item rejection and rejection layer distillation

HDN-nr-nd: remove all distillation and only train the score function

final scoring layers will receive items with more relevance and focus on them. When HDN is trained without rejections (HDN-nr-nd), it implies that HDN needs to model all pairs of user-item interactions. Being smaller and not as optimized in design compared with NeuMF, it yields significantly lower NDCG and HR when the distillation process is absent.

From this experiment we could see that other than helping the rejection layers to keep a specific rejection rate, the distillation training for HDN that we have proposed in this paper is also a crucial process to maintain the performance.

4.4.5 Experiment on Number of Decision Modules N

The results for recommendation when number of HDN modules changes are shown in Figure 4.3. In Figure 4.3a and 4.3b, the NDCG and HR are shown. The total recommen-

Figure 4.3: Timing and Performance when number of HDN modules changes

ation time including inference and ranking are shown in 4.3c. For the results, we can observe that 3 rejection modules are sufficient for HDN to maintain the recommendation quality of NeuMF measured by NDCG and HR. There is no significant increase of recommendation quality on HDN with more rejection modules, which indicates that $N = 3$ is sufficient to learn from the teacher recommendation system NeuMF. The recommendation time increases with more HDN layers, but since most items are rejected in earlier layers, the inference time does not increase significantly.

Chapter 5

Conclusion

In conclusion, by leverage the representation power of deep neural networks, deep collaborative filtering leads to good recommendation quality. However, the deep interaction matching based methods suffer from the trade-off of network complexity versus speed.

The main contribution of the thesis is the proposed Hierarchical Decision Network architecture, which solve the problem by dropping irrelevant items in a hierarchical network structure during the recommendation process. The computational cost is reduced to be depth independent according to our analysis.

Another contribution is the distillation based training method provided for training HDN, which is also shown effective for training HDN to reach the recommendation quality of a teacher network. We have tested the methods on real-life datasets, and the result indicated great potential to improve the inference time for deep CF, while maintaining the recommendation quality.

5.1 Future work

5.1.1 Alternative Decision Module

A simple decision module is put into HDN architecture and used for the experiments, in more complex recommendation cases, the simple decision module mentioned in this

work may not be sufficient. The design of a larger decision module can be helpful for these use cases.

5.1.2 Content Based Recommendation

Content-based recommendation models can be integrated into HDN, such that the decisions and scores can be based on both user/item features and past interactions.

5.1.3 Direct Training of HDN

Current version of HDN uses distillation based training, which requires a teacher model. By utilizing policy gradients [25], HDN can be trained directly from data, which may remove the limitation from the teacher model and improve the recommendation quality of HDN.

5.1.4 Integration with other model compression methods

Besides distillation, quantization and pruning are popular methods of enabling faster deep network inference. The integration of these methods may also help further reduce the computational cost.

5.1.5 Explore model acceration methods with de-biased recommendation systems

Recently as recommendation systems are getting more and more popular, the research community starts to pay more attention to biases throughout the process of the recommendation [26], from user interaction, data collection to statistical modeling. The researchers have proposed methods such as propensity scores based methods [26, 27], causal inference [28] and better evaluation methods [29] to de-bias the recommendation process and get higher quality recommendations.

In most scenarios, recommendation system also need to be fair to people and treat different social groups equally while maintaining accurate predictions. The topic of

fairness in recommendation system is also under active exploration, e.g. [30, 31]. All the methods of making faster deep recommendation systems above can be explored on the models from the recent research, towards a brighter future powered by recommendation systems with fewer bias and fairer results.

5.2 Author's list of publication

- Chen, Y., & Pan, S. J. (2021). Towards Faster Deep Collaborative Filtering via Hierarchical Decision Networks. Proceedings of the AAAI Conference on Artificial Intelligence, 35(5), 4001-4009.

Bibliography

- [1] C.C. Aggarwal. *Recommender Systems: The Textbook*. Springer International Publishing, 2016.
- [2] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, volume 2007, pages 5–8, 2007.
- [3] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 426–434, 2008.
- [4] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA, 2007.
- [5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.
- [6] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. Deep matrix factorization models for recommender systems. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 3203–3209. ijcai.org, 2017.
- [7] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR*

Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019., pages 165–174, 2019.

- [8] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 173–182, 2017.
- [9] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. Outer product-based neural collaborative filtering. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 2227–2233, 2018.
- [10] Feng Xue, Xiangnan He, Xiang Wang, Jiandong Xu, Kai Liu, and Richang Hong. Deep item-based collaborative filtering for top-n recommendation. *ACM Trans. Inf. Syst.*, 37(3):33:1–33:25, April 2019.
- [11] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.
- [12] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*, pages 191–198, 2016.
- [13] Changhua Pei, Yi Zhang, Yongfeng Zhang, Fei Sun, Xiao Lin, Hanxiao Sun, Jian Wu, Peng Jiang, Junfeng Ge, Wenwu Ou, and Dan Pei. Personalized re-ranking for recommendation. In Toine Bogers, Alan Said, Peter Brusilovsky, and Domonkos Tikk, editors, *Proceedings of the 13th ACM Conference on Recommender Systems, RecSys 2019, Copenhagen, Denmark, September 16-20, 2019*, pages 3–11. ACM, 2019.
- [14] Jiayi Tang and Ke Wang. Ranking distillation: Learning compact ranking models with high performance for recommender system. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 2289–2298, 2018.

- [15] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.*, 18:187:1–187:30, 2017.
- [16] Wang-Cheng Kang, Derek Zhiyuan Cheng, Ting Chen, Xinyang Yi, Dong Lin, Lichan Hong, and Ed H. Chi. Learning multi-granular quantized embeddings for large-vocab categorical features in recommender systems. In Amal El Fallah Seghrouchni, Gita Sukthankar, Tie-Yan Liu, and Maarten van Steen, editors, *Companion of The 2020 Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 562–566. ACM / IW3C2, 2020.
- [17] Ting Chen, Martin Renqiang Min, and Yizhou Sun. Learning k-way d-dimensional discrete codes for compact embedding representations. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 853–862. PMLR, 2018.
- [18] Defu Lian, Haoyu Wang, Zheng Liu, Jianxun Lian, Enhong Chen, and Xing Xie. Lightrec: A memory and search-efficient recommender system. In Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen, editors, *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 695–705. ACM / IW3C2, 2020.
- [19] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):19, 2016.
- [20] Xue Geng, Hanwang Zhang, Jingwen Bian, and Tat-Seng Chua. Learning image and user features for recommendation in social networks. In *Proceedings of the IEEE ICCV*, pages 4274–4282, 2015.
- [21] Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. Learning to rank using gradient descent. In

- [22] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002.
- [23] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.
- [24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [25] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000.
- [26] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. Bias and debias in recommender system: A survey and future directions. *arXiv preprint arXiv:2010.03240*, 2020.
- [27] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. Learning to rank with selection bias in personal search. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 115–124, 2016.
- [28] Yixin Wang, Dawen Liang, Laurent Charlin, and David M Blei. Causal inference for recommender systems. In *Fourteenth ACM Conference on Recommender Systems*, pages 426–431, 2020.
- [29] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. Recommendations as treatments: Debiasing learning and evaluation. In *international conference on machine learning*, pages 1670–1679. PMLR, 2016.

- [30] Meike Zehlike, Ke Yang, and Julia Stoyanovich. Fairness in ranking: A survey. *ArXiv*, abs/2103.14000, 2021.
- [31] Yunqi Li, Hanxiong Chen, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. User-oriented fairness in recommendation. In *Proceedings of the Web Conference 2021*, pages 624–632, 2021.