Towards Faster Deep Collaborative Filtering via Hierarchical Decision Networks

Yu Chen[1] and **Sinno Jialin Pan** [1]
[1] Nanyang Technological University
{chenyu, sinnopan}@ntu.edu.sg

Abstract

For personalized recommendations, collaborative filtering (CF) methods aim to recommend items to users based on data of historical user-item interactions. Deep learning has indicated success in improving performance of CF methods in recent works. However, to generate an item recommendation list for each user, a lot of deep learning-based CF methods require every pair of users and items to be passed through multiple neural layers. This requires intensive computation and makes real-time end-to-end neural recommendations very costly. To address this issue, in this paper, we propose a new deep learning-based hierarchical decision network to filter out irrelevant items to save computation cost while maintaining good recommendation accuracy of deep CF methods. We also develop a distillation-based training algorithm, which uses a well-trained CF model as a teacher network to guide the training of the decision network. We conducted extensive experiments on real-world benchmark datasets to verify the effectiveness and the efficiency of our decision network for making recommendations. The experimental results indicate that the proposed decision network is able to maintain or even improve the recommendation quality in terms of various metrics and meanwhile enjoy lower computational cost.

Introduction

In personalized recommendation, users' preferences are usually modeled from historical user-item interactions. Specifically, factor-based collaborative filtering (CF) methods model each user and each item as latent factors $\mathbf{h}_u$ and $\mathbf{h}_v$, respectively, to capture the characteristics of users and items, and the outcome of their dot product is considered as a relevance score between the item and the user for recommendation.

In the past decade, deep learning (LeCun, Bengio, and Hinton 2015) has been widely applied in various application domains, from computer vision and speech recognition to natural language processing and cybersecurity. Recently, deep learning has also raised attention in recommender systems, especially for CF methods. In general, deep learning-based CF has been explored in mainly two directions: representation learning and matching function learning. In the direction of representation learning (Xue et al. 2017; Wang et al. 2019), 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. 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 (He et al. 2017) is a representative in the direction of matching function learning. In NCF, a deep neural network of multiple Generalized Matrix Factorization (GMF) layers and Multi-Layer Perceptron (MLP) layers is used as a black-box to capture the underlying interactions between users and items. Later, various neural network architectures have been proposed to model user-item interactions (He et al. 2018; Xue et al. 2019). Recently, a unified framework that integrates the two approaches of representation learning and matching function learning is proposed (Deng et al. 2019).

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 (He et al. 2017). 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 server with limited computational resources.

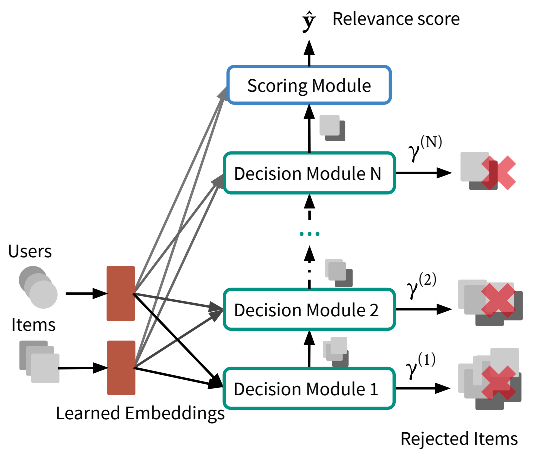


Figure 1: The Hierarchical Decision Network, where $\gamma^{(n)}$ represents the rejection ratio associated with the n-th decision module.

Our Novelty

To address the aforementioned issue of deep CF methods, in this paper, we propose a novel network architecture, Hierarchical Decision Networks (HDNs). In our designed architecture as shown in Figure 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 (Hinton, Vinyals, and Dean 2015) 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. To the best of our knowledge, paired with the distillation-based training algorithm, HDNs are the first deep learning architecture that aims to reduce inference cost of deep matching function models for CF.

Related Work

One similar approach to this problem is by using a two-step process (Covington, Adams, and Sargin 2016), 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. 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 (Pei et al. 2019). 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. We have implemented a naive two-staged recommendation and PRM as baseline methods to compare with HDN for recommendation quality.

In (Tang and Wang 2018), 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.

Quantization (Hubara et al. 2017) of neural network is also a common approach to reduce the computation cost. In (Kang et al. 2020), the authors have applied the quantization method, KD encoding (Chen, Min, and Sun 2018), 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 (Lian et al. 2020), 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.

Methodology

Neural Collaborative Filtering Framework

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 $\mathbf{h}_{v_t}$:

$$\hat{y}_{u_t,v_t} = \langle \mathbf{h}_{u_t}, \mathbf{h}_{v_t} \rangle . \tag{1}$$

The idea of the Neural Collaborative Filtering (NCF) framework (He et al. 2017) is to use a neural network to replace the inner product, such that deeper interactions between the user-item pair can be captured:

$$\hat{y}_{u_t,v_t} = \mathrm{NeuralNetwork}\left(\mathbf{h}_{u_t}, \mathbf{h}_{v_t}\right). \tag{2}$$

Recently, various network architecture have been proposed based on the NCF framework (He et al. 2017; Xue et al. 2017; He et al. 2018; Xue et al. 2019). For instance, in (He et al. 2017), a neural matrix factorization network (NeuMF) is proposed as follows:

$$\hat{y}_{u_t,v_t} = \mathbf{w}^\top \begin{bmatrix} \mathbf{h}_{u_t} \odot \mathbf{h}_{v_t} \\ \mathrm{MLP}\left(\begin{bmatrix} \mathbf{h}_{u_t} \\ \mathbf{h}_{v_t} \end{bmatrix}\right) \end{bmatrix}, \tag{3}$$

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 $\hat{y}_{u_t,v_t}$. In (He et al. 2018), 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 (He et al. 2018) 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:

$$\hat{y}_{u_t,v_t} = \mathbf{w}^\top \mathrm{CNN}(\mathbf{h}_{u_t} \otimes \mathbf{h}_{v_t}), \tag{4}$$

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.

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.

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 $\mathrm{D}^{(1)}$, 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 $\mathbf{h}^{(1)}$:

$$\left[\hat{d}^{(1)}_{\text{rej}}, \mathbf{h}^{(1)}\right] = \mathrm{D}^{(1)}(\mathbf{h}_{u_t}, \mathbf{h}_{v_t}). \tag{5}$$

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 $\mathrm{D}^{(n)}$, 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 $\mathrm{D}^{(n-1)}$ as input and outputs the rejection decision and a new interaction embedding for the next layer:

$$\left[\hat{d}^{(n)}_{\text{rej}}, \mathbf{h}^{(n)}\right] = \mathrm{D}^{(n)}(\mathbf{h}_{u_t}, \mathbf{h}_{v_t}, \mathbf{h}^{(n-1)}). \tag{6}$$

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:

$$\mathbf{h}^{(n)}_{\text{uni}} = \begin{bmatrix} \mathbf{h}_{u_t} \odot \mathbf{h}_{v_t} \\ \mathbf{h}^{(n-1)} \end{bmatrix}. \tag{7}$$

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

$$\hat{d}^{(n)}_{\text{rej}} = \theta^{(n)\top}_{\text{rej}} \mathbf{h}^{(n)}_{\text{uni}}. \tag{8}$$

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

$$\mathbf{h}^{(n)} = a\left(W^{(n)} \mathbf{h}^{(n)}_{\text{uni}}\right), \tag{9}$$

where $a(\cdot)$ is the activation function. In this work, we adopt RELU (LeCun, Bengio, and Hinton 2015) for $a(\cdot)$.

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

$$\hat{y}_{u_t,v_t} = W_{\text{score}} \begin{bmatrix} \mathbf{h}_{u_t} \odot \mathbf{h}_{v_t} \\ \mathbf{h}^{(N)} \end{bmatrix}, \tag{10}$$

which is similar to the design of the decision module.

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 $\mathrm{D}^{(n)}$. 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.

Distillation of the relevance score Similar to knowledge distillation (Hinton, Vinyals, and Dean 2015), 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:

$$\ell_{\mathrm{S}} = \mathrm{KL}(p, \hat{p}) = \log\frac{\hat{\sigma}}{\sigma} + \frac{\sigma^2 + (\mu - \hat{\mu})^2}{2\hat{\sigma}^2} - \frac{1}{2} \tag{11}$$

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:

$$\sum_{u_t \in U, v_t \in V} p_{u_t,v_t} \log \hat{p}_{u_t,v_t} + (1 - p_{u_t,v_t}) \log(1 - \hat{p}_{u_t,v_t}), \tag{12}$$

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

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

$$\frac{\sum_{v_t \in V_t} \mathbb{I}(\hat{d}^{(n)}_{v_t} > 0)}{|V_t|} = \gamma^{(n)}, \tag{13}$$

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

$$\ell_{\mathrm{D}^{(n)},u_t,v_t} = \max(0, 1 - \hat{d}^{(n)}_{u_t,v_t} d_{v_t,\gamma^{(n)}}), \tag{14}$$

$$\text{where } d^{(n)}_{v_t,\gamma^{(n)}} = \begin{cases} 1, & \kappa_{v_t} \ge \bar{\kappa}_{\gamma^{(n)}} \\ -1, & \kappa_{v_t} < \bar{\kappa}_{\gamma^{(n)}} \end{cases} \tag{15}$$

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

$$\ell = \ell_{\mathrm{S}} + \sum_{u_t \in U, v_t \in V} \sum_{n=1}^{N} \ell_{\mathrm{D}^{(n)},u_t,v_t}. \tag{16}$$

All the parameters of a HDN can be learned by minimizing (16) by gradient decent. The overall training algorithm is summarized in Algorithm 1.

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 pratice, 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 for layer k is

$$(1-\gamma)^{k-1} CT. \tag{17}$$

Let $\alpha = 1 - \gamma$. The total inference cost of HDN is $CT + \alpha CT + \alpha^2 CT + \ldots + \alpha^M CT$, 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 $1 > \gamma \ge 0.5$, i.e. $0 < \alpha \le 0.5$, we have

$$f(\alpha) \le \frac{1 - 0.5\alpha^{M+1}}{1 - 0.5} = 2 - \alpha^{M+1} < 2. \tag{18}$$

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 $\gamma^{(n)}$
        $\left(\hat{d}_{\text{rej}}^{(n)}, \mathbf{h}^{(n)}\right) = \mathrm{D}^{(n)}(u, v, \mathbf{h}^{(n-1)})$
        $\ell_{\mathrm{D}^{(n)}} = \max(0, 1 - \hat{d}^{(n)} d_{v_t, \gamma^{(n)}})$
        $n += 1$
    end while
    $\hat{p} = \sigma(\mathcal{S}(u, v, \mathbf{h}^{(n)}))$
    Construct $\ell_{\mathrm{S}}$ based on (11) or (12)
    $\ell = \ell_{\mathrm{S}} + \sum_{u_t \in U, v_t \in V} \sum_{n=1}^{N} \ell_{\mathrm{D}^{(n)}, u_t, v_t}$
    gradient decent on $\nabla \ell$
    epoch += 1
end while
```

Hence the computational cost of HDN is

$$\frac{CT(1 - \alpha^{M+1})}{1 - \alpha} < 2CT, \tag{19}$$

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.

Experiments

We investigated the performance of HDN in four experiments to verify if HDN has the capability to reject unrelated items in the hierarchical structure, reduce the inference time and maintain the recommendation quality. Additionally, we tested the effectiveness of our distillation method.

Experiment Datasets

The experiments are performed in the following datasets:

- *MovieLens*: MovieLens datasets are widely-used among the recommendation system community (Harper and Konstan 2016). We have used both the 1 million and 20 million version. 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.
- *Pinterest*: The Pinterest dataset is constructed by (Geng et al. 2015) using data from `pinterest.com`. There are 55,187 users, 9,916 items and 1,500,809 interactions, in this dataset.
- *Yelp*: The Yelp dataset is originally published on Yelp Dataset Challenge. It consists of user reviews on restaurants. It is quite sparse: 25,677 users and 25,815 reviews ratings as items, with only 730,790 user-item interactions.

We followed the same setups for the NCF paper(He et al. 2017), 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 rest is 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.

Implementation Details

In the experiments, we constructed a HDN with $N = 3$ 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 (He et al. 2017, 2018).

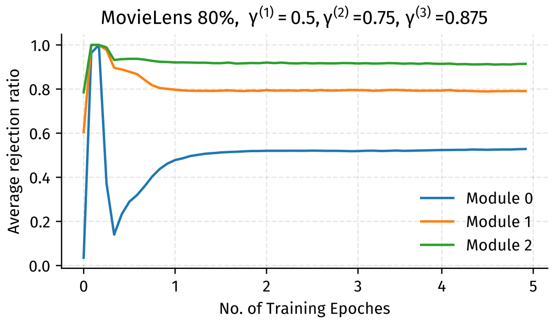
Although our distillation model allows training using any possible teacher-student pair, it would be very time-consuming. Instead, we used a scheme similar to negative sampling during distillation, and used the score information for negative samples from the teacher model. We used a positive to negative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

We implemented HDN and the baseline method using PyTorch(Paszke et al. 2017). Gradient descent of minibatch size of 512 and Adam(Kingma and Ba 2014) 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 a NVIDIA DGX-1 machine.

Experiment on Item Rejection Rate

Different rejection ratios $\gamma^{(n)}$ would affect the trade-off between the inference time and recommendation accuracy. With a user-defined $\gamma^{(n)}$, HDN would learn to reject items at a fixed rate in different layers. We test the actual rejection ratio versus the proposed γ to see if HDN is actually rejecting the amount of item requested in all datasets and found out number of filtered items are similar with rejection ratio specified.

The results from MovieLens-1M with different rejection rations are shown as an example in Figure 2, the values shown are the ones after item rejection of the previous layers.



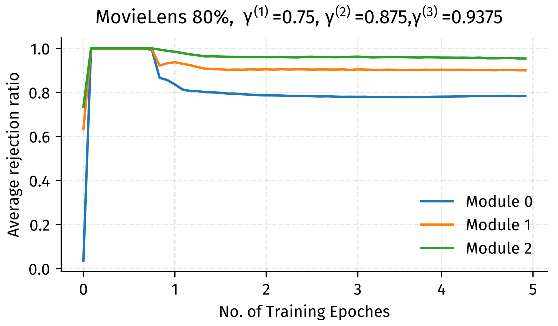


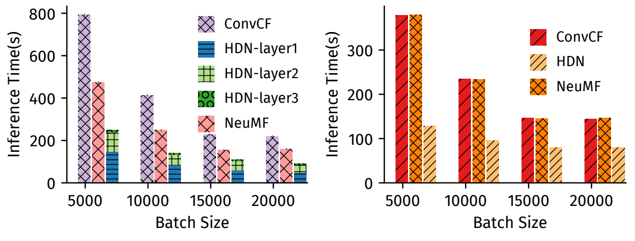
Figure 2: The actual rejection rate during distillation

Experiment on Inference Time

Besides our analysis in the previous sessions, we would like to show the theoretical 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. In our experiment, we first sort all items by popularity to enable more batches for early rejection. When a whole batch is rejected at one of the rejection layers, the inference time will be reduced for this batch. 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.



(a) Inference time (b) Ranking time

Figure 3: Time comparison with limited batch size on MovieLens-20M ($\gamma_1 = 0.75, \gamma_2 = 0.875, \gamma_3 = 0.9375$)

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 3a, 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 3b shows the time for for retrieving the top-K items by ranking scores. The results in Figure 3 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 could potentially yield further inference time improvements.

Experiment 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 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)(Burges et al. 2005) for both models.

HR@K measures the ratio of whether a positive test item is in the top-K ranking list. NDCG@K is Discounted Cumulative Gain (DCG) normalized using the best possible ranking result and we have used the standard NDCG, which is using the logarithmic discount function.

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. We have also simulated a two-stage process (Covington, Adams, and Sargin 2016) by using Matrix Factorization to select the top 100 items and then rank them using NeuMF. Besides the direct approach, re-ranking method PRM (Pei et al. 2019) is also used in the second stage. Since we did not include extra user feature in our datasets, PRM-BASE is used.

	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
MF(top100) + PRM-BASE	0.616	0.924	0.537	0.798	0.181	0.463	0.128	0.329
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 ($\gamma_1 = 0.5, \gamma_2 = 0.75, \gamma_3 = 0.875$)	0.633	0.954	0.562	0.850	0.197	0.458	0.137*	0.329
HDN ($\gamma_1 = 0.75, \gamma_2 = 0.875, \gamma_3 = 0.9375$)	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 ($\gamma_1 = 0.5, \gamma_2 = 0.75, \gamma_3 = 0.875$)	0.627	0.958	0.578	0.863	0.183	0.487	0.133	0.331
HDN ($\gamma_1 = 0.75, \gamma_2 = 0.875, \gamma_3 = 0.9375$)	0.631	0.956	0.576	0.865	0.180	0.486	0.135	0.329

Table 1: Recommendation performance measured by NDCG and HR when K=20

PopRank, where the only the item popularity is included for ranking, and Matrix Factorization(MF) as also included as baselines. The results are consistent for all Ks (number of top items used for evaluation) that we have tested, and the result when $K = 20$ is presented in Table 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). As a student model for both, 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 the two-stage model.

Experiment on Effect of Distillation-based Training

We further investigated the effectiveness of our distillation method by training HDN differently. The purpose of the distillation based training is for the rejection layers to maintain a fixed rejection ratio γ. If HDN is trained without distillation, we can use the output value of the rejection layer and remove items according to the same fixed rejection ratio(HDN-nd) during test time to achieve the same effect. Unlike HDN, which allows testing on any pair(s) of user and item, this process of rejection-while-testing have to be performed after all the items of a user have been scored and ranked.

Other variation of rejection has been performed: we put item rejection aside first and test the result 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 have tested HDN trained without distillation by ignoring all the item rejections and only train on the score function (HDN-nr-nd).

As seen the results in Table 2, without distillation, the performance of HDN have did not reach the same as previous methods, which is expected considering the rejection layers did not learn to reject items a specific ratio.

The performance of HDN is also 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,

	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-nd	0.129	0.328	0.553	0.842
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 2: Result on MovieLens-20M and Yelp ($K = 20$)

it can be used 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 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.

Conclusion and Future Work

In conclusion, we have proposed a novel deep neural network HDN for faster inference of deep CF and designed a distillation based training method. We have tested the methods on real-life datasets and the result indicated great potential to improve the inference time for deep CF. 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. Besides, 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.

Acknowledgments

This research is partially supported by the NTU Singapore Nanyang Assistant Professorship (NAP) grant M4081532.020 and a research grant AN-GC-2018-022 under Alibaba-NTU Singapore Joint Research Institute.

References

Burges, C. J. C.; Shaked, T.; Renshaw, E.; Lazier, A.; Deeds, M.; Hamilton, N.; and Hullender, G. N. 2005. Learning to rank using gradient descent. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, 89–96. doi:10.1145/1102351.1102363. URL https://doi.org/10.1145/1102351.1102363.

Chen, T.; Min, M. R.; and Sun, Y. 2018. Learning K-way D-dimensional Discrete Codes for Compact Embedding Representations. In Dy, J. G.; and Krause, A., eds., *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*, 853–862. PMLR. URL http://proceedings.mlr.press/v80/chen18g.html.

Covington, P.; Adams, J.; and Sargin, E. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*, 191–198. doi:10.1145/2959100.2959190. URL https://doi.org/10.1145/2959100.2959190.

Deng, Z.; Huang, L.; Wang, C.; Lai, J.; and Yu, P. S. 2019. DeepCF: A Unified Framework of Representation Learning and Matching Function Learning in Recommender System. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, 61–68. URL https://aaai.org/ojs/index.php/AAAI/article/view/3769.

Geng, X.; Zhang, H.; Bian, J.; and Chua, T.-S. 2015. Learning image and user features for recommendation in social networks. In *Proceedings of the IEEE ICCV*, 4274–4282.

Harper, F. M.; and Konstan, J. A. 2016. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5(4): 19.

He, X.; Du, X.; Wang, X.; Tian, F.; Tang, J.; and Chua, T. 2018. 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.*, 2227–2233. doi:10.24963/ijcai.2018/308. URL https://doi.org/10.24963/ijcai.2018/308.

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, 173–182. doi:10.1145/3038912.3052569. URL https://doi.org/10.1145/3038912.3052569.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop*. URL http://arxiv.org/abs/1503.02531.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2017. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *J. Mach. Learn. Res.* 18: 187:1–187:30. URL http://jmlr.org/papers/v18/16-456.html.

Kang, W.; Cheng, D. Z.; Chen, T.; Yi, X.; Lin, D.; Hong, L.; and Chi, E. H. 2020. Learning Multi-granular Quantized Embeddings for Large-Vocab Categorical Features in Recommender Systems. In Seghrouchni, A. E. F.; Sukthankar, G.; Liu, T.; and van Steen, M., eds., *Companion of The 2020 Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, 562–566. ACM / IW3C2. doi:10.1145/3366424.3383416. URL https://doi.org/10.1145/3366424.3383416.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *nature* 521(7553): 436.

Lian, D.; Wang, H.; Liu, Z.; Lian, J.; Chen, E.; and Xie, X. 2020. LightRec: A Memory and Search-Efficient Recommender System. In Huang, Y.; King, I.; Liu, T.; and van Steen, M., eds., *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, 695–705. ACM / IW3C2. doi:10.1145/3366423.3380151. URL https://doi.org/10.1145/3366423.3380151.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*.

Pei, C.; Zhang, Y.; Zhang, Y.; Sun, F.; Lin, X.; Sun, H.; Wu, J.; Jiang, P.; Ge, J.; Ou, W.; and Pei, D. 2019. Personalized re-ranking for recommendation. In Bogers, T.; Said, A.; Brusilovsky, P.; and Tikk, D., eds., *Proceedings of the 13th ACM Conference on Recommender Systems, RecSys 2019, Copenhagen, Denmark, September 16-20, 2019*, 3–11. ACM. doi:10.1145/3298689.3347000. URL https://doi.org/10.1145/3298689.3347000.

Tang, J.; and Wang, K. 2018. 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*, 2289–2298. doi:10.1145/3219819.3220021. URL https://doi.org/10.1145/3219819.3220021.

Wang, X.; He, X.; Wang, M.; Feng, F.; and Chua, T. 2019. 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.*, 165–174. doi:10.1145/3331184.3331267. URL https://doi.org/10.1145/3331184.3331267.

Xue, F.; He, X.; Wang, X.; Xu, J.; Liu, K.; and Hong, R. 2019. Deep Item-based Collaborative Filtering for Top-N Recommendation. *ACM Trans. Inf. Syst.* 37(3): 33:1–33:25. ISSN 1046-8188. doi:10.1145/3314578. URL http://doi.acm.org/10.1145/3314578.

Xue, H.; Dai, X.; Zhang, J.; Huang, S.; and Chen, J. 2017. Deep Matrix Factorization Models for Recommender Systems. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 3203–3209. ijcai.org. doi:10.24963/ijcai.2017/447. URL https://doi.org/10.24963/ijcai.2017/447.