Neural Collaborative Autoencoder

Qibing Li, Xiaolin Zheng, *Senior Member, IEEE,* Xinyue Wu

Abstract—In recent years, deep neural networks have yielded state-of-the-art performance on several tasks. Although some recent works have focused on combining deep learning with recommendation, we highlight three issues of existing models. First, these models cannot work on both explicit and implicit feedback, since the network structures are specially designed for one particular case. Second, due to the difficulty on training deep neural networks, existing explicit models do not fully exploit the expressive potential of deep learning. Third, neural network models are easier to overfit on the implicit setting than shallow models. To tackle these issues, we present a generic recommender framework called *Neural Collaborative Autoencoder* (NCAE) to perform collaborative filtering, which works well for both explicit feedback and implicit feedback. NCAE can effectively capture the subtle hidden relationships between interactions via a non-linear matrix factorization process. To optimize the deep architecture of NCAE, we develop a three-stage pre-training mechanism that combines supervised and unsupervised feature learning. Moreover, to prevent overfitting on the implicit setting, we propose an error reweighting module and a sparsity-aware data-augmentation strategy. Extensive experiments on three real-world datasets demonstrate that NCAE can significantly advance the state-of-the-art.

Index Terms—Recommender System, Collaborative Filtering, Neural Network, Deep Learning

1 Introduction

In recent years, recommender systems (RS) have played a significant role in E-commerce services. A good recommender system may enhance both satisfaction for users and profit for content providers. For example, nearly 80% of movies watched on Netflix are recommended by RS [7]. The key to design such a system is to predict users' preference on items based on past activities, which is known as collaborative filtering (CF) [28]. Among the various CF methods, matrix factorization (MF) [11], [13], [15], [30] is the most used one, which models the user-item interaction function as the inner product of user latent vector and item latent vector. Due to the effectiveness of MF, many integrated models have been devised, such as CTR [36], HFT [22] and timeSVD [14]. However, the inner product is not sufficient for capturing subtle hidden factors from the interaction data [8].

Currently, a trend in the recommendation literature is the utilization of deep learning to handle the auxiliary information [20], [37], [38] or directly model the interaction function [8], [29], [33], [39]. Thus, based on these two usage scenarios, deep learning based recommender systems can be roughly categorized into integration models and neural network models [42]. Integration models utilize deep neural networks to extract the hidden features of auxiliary information, such as item descriptions [37], [38], user profiles [20] and knowledge bases [41]. The features are then integrated into the CF framework to perform hybrid recommendation. Although integration models involve both deep learning and CF, they actually belong to MF-based models because they use an inner product to model the interaction data, and thus face the same issue like MF.

On the other hand, neural network models directly perform collaborative filtering via modeling the interaction data. Due to the effectiveness of deep components, neural network models are able to discover the non-linear hidden relationships from data [18], [33]. For example, Collaborative Filtering Network (CFN) [33] is a state-of-the-art model for explicit feedback, which utilizes DAE [34] to encode sparse user/item preferences (rows or columns of the observed rating matrix) and aims to reconstruct them in the decoder layer. However, we notice that existing explicit neural models do not fully exploit the representation power of deep architectures; stacking more layers yields little performance gain [33], [43]. This is mainly caused by two reasons. First, without a proper pre-training strategy, training deep neural networks is difficult [6]. Second, due to the sparse nature of RS, conventional layer-wise unsupervised pre-training [2], [10] does not work in this case[1]. Besides, neural network models are easier to overfit on the implicit setting due to the highly non-linear expressiveness (i.e., predicting all ratings as 1, since observed interactions are all converted as 1). Neural Collaborative Filtering (NCF) [8] is a state-of-the-art implicit neural model that can capture the non-linear hidden factors while combatting overfitting (NCF samples negative feedback from unobserved data to perform pairwise learning like BPR). However, the NCF architecture is designed at the interaction level, which is time-consuming to rank all items for all users during evaluation. Furthermore, providing more item correlation patterns may be the key factors to combat overfitting for implicit neural models (see the discussion in Sec. 4.2).

To address the aforementioned issues, we present a simple deep learning based recommender framework called *Neural Collaborative Autoencoder* (NCAE) for both explicit feedback and implicit feedback. The central idea of NCAE is to learn hidden structures that can reconstruct user/item preferences via a non-linear matrix factorization process. The NCAE architecture is designed at the user/item level, which takes the sparse user or item vectors as inputs for

• *Q. Li, X. Zheng and X. Wu are with the College of Computer Science and Technology, Zhejiang university, Hangzhou, China. E-mail: qblee@zju.edu.cn, xlzheng@zju.edu.cn and wxinyue@zju.edu.cn*

Manuscript received April 19, 2005; revised August 26, 2015.

1. Supervised pre-training in the first hidden layer is critical to the performance, since unsupervised reconstruction method may lose user/item information. (see Sec. 5.2)

batch training and evaluation. By utilizing a sparse forward module and a sparse backward module, NCAE is scalable to large datasets and robust to sparse data; a new training loss is also designed for sparse user/item inputs. We further develop a novel three-stage pre-training mechanism, combining supervised and unsupervised feature learning to train the deep architecture of NCAE. As a result, NCAE with deeper architectures is more powerful and expressive for explicit feedback. Besides, NCAE includes two ingredients to prevent overfitting on the implicit setting: an error reweighting module that treats all unobserved interactions as negative feedback (i.e., whole-based method [9], [11]), and a sparsity-aware data-augmentation strategy that provides more item correlation patterns in training and discovers better ranking positions of true positive items during inference. The key contributions of this paper are summarized as follows:

- We present a scalable and robust recommender framework named NCAE for both explicit feedback and implicit feedback, where we adapt several effective approaches from the deep learning literature to the recommendation domain, including autoencoders, dropout, pre-training and data-augmentation. NCAE can capture the subtle hidden factors via a non-linear matrix factorization process.
- Thanks to the effectiveness of three-stage pre-training mechanism, NCAE can exploit the representation power of deep architectures to learn high-level abstractions, bringing better generalization. We further explore several variants of our proposed pre-training mechanism and compare our method with conventional pre-training strategies to better understand its effectiveness.
- Compared with other implicit neural models, NCAE can utilize the error reweighting module and the sparsity-aware data augmentation to combat overfitting, and thus performs well on the implicit setting without negative sampling.
- Extensive experiments on three real-world datasets demonstrate the effectiveness of NCAE on both explicit and implicit settings.

The paper is organized as follows. In Section 2, we discuss related work on applying neural networks to recommender systems. In Section 3, we provide the problem definition and notations. We describe the proposed NCAE in Section 4. We conduct experiments in Section 5 before concluding the paper in Section 6.

2 Related Work

We review the existing models in two groups, including integration models and neural network models [42].

Integration models combine DNNs with the CF framework such as PMF [23] to perform hybrid recommendation, which can deal with the cold-start problem and alleviate data sparsity. The deep components of integration models are primarily used to extract the hidden features of auxiliary information [4], [20], [37], [38], [41]. For example, Collaborative Deep Learning (CDL) [37] integrates Stack Denoising Autoencoder (SDAE) [35] and PMF [23] into a unified probabilistic graph model to jointly perform deep content feature learning and collaborative filtering. However, integration models utilize an inner product to model the interaction data, which is not sufficient for capture the complex structure of interaction data [8]. Different from integration models, NCAE can model the interaction function via a non-linear matrix factorization process. Our focus is to design a deep learning based recommender framework for both explicit feedback and implicit feedback. Therefore, we remain the hybrid recommender (combining NCAE with deep content feature learning) to the future work.

Neural network models utilize DNNs to learn the interaction function from data, which are able to discover the non-linear hidden relationships [8], [18], [33]. We may further divide neural network models into two sub-categories: explicit neural models and implicit neural models. Restricted Boltzmann Machine (RBM) for CF [27] is the early pioneer work that applies neural networks to explicit feedback. Recently, autoencoders have become a popular building block for explicit models [29], [33]. For example, [33] proposes CFN that achieves the best performance for explicit feedback (with well-tuned parameters and additional data pre-precessing). Compared to CFN, our NCAE employs a new training loss to balance the impact of sparse user/item vectors and achieves comparable performance without complex procedures of CFN. By utilizing the three-stage pre-training, NCAE with deeper architectures is more powerful and expressive than CFN. To our best knowledge, NCAE obtains a new state-of-the-art result for explicit feedback. On the other hand, implicit neural models also exploit the representation capacity of neural networks and combat overfitting by sampling negative interactions. Compared to the state-of-the-art implicit model NCF, our NCAE takes user vectors as inputs for batch evaluation; NCAE explicitly gathers information from other users for batch training, which may be a better network structure for CF (see Sec. 5.3). Besides, the error reweighting module and the sparsity-aware data-augmentation can provide more item correlation patterns for NCAE, greatly enhancing the top-M recommendation performance. Overall, NCAE is a generic framework for CF that performs well on the two settings.

3 Problem Definition

We start by introducing some basic notations. Matrices are written as bold capital letters (e.g., $\mathbf{X}$) and vectors are expressed as bold lowercase letters (e.g., $\mathbf{x}$). We denote the i-th row of matrix $\mathbf{X}$ by $\mathbf{X}_i$ and its (i, j)-th element by $\mathbf{X}_{ij}$. For vectors, $\mathbf{x}_i$ denotes the i-th element of $\mathbf{x}$.

In collaborative filtering, we have M users, N items and a partially observed user-item rating matrix $\mathbf{R} = [\mathbf{R}_{ij}]_{M \times N}$. Fig. 1 shows an example of explicit feedback and implicit feedback. Generally, explicit feedback problem can be regarded as a regression problem on observed ratings, while the implicit one is known as a one-class classification problem [24] based on interactions.

We let $\mathcal{R}$ denote the set of user-item pairs (i, j, r_{ij}) where values are non-zero, $\bar{\mathcal{R}}$ denote the set of unobserved, missing triples. Let $\mathcal{R}_i$ denote the set of item preferences in

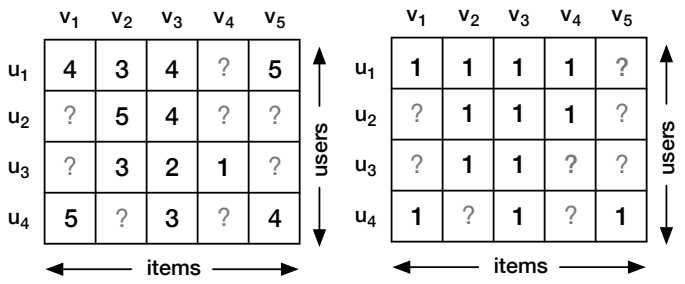


Fig. 1: A simple example illustrating the rating matrices for explicit feedback and implicit feedback.

the training set for a particular user i; similar notations for $\bar{\mathcal{R}}_i$. The goal of the recommender system is to pick a subset of items from the candidate set $\bar{\mathcal{R}}_i$ for each user i. Besides, we use a sparse vector $\mathbf{u}_i \in \mathbb{R}^N$ as the input of NCAE, which has only $|\mathcal{R}_i|$ observed values of user i: $\mathbf{u}_{ij} = r_{ij}$ if j is in $\mathcal{R}_i$, otherwise, $\mathbf{u}_{ij} = 0$; $\tilde{\mathbf{u}}_i$, $\hat{\mathbf{u}}_i$ are the corrupted version of $\mathbf{u}_i$ and the dense estimate of $\mathbf{R}_i$; similar notations $\mathcal{R}_j$, $\mathbf{v}_j$, $\tilde{\mathbf{v}}_j$ and $\hat{\mathbf{v}}_j$ for each item j.

For neural network settings, K_l, $\mathbf{W}^l \in \mathbb{R}^{K_l \times K_{l-1}}$, $\mathbf{b}^l \in \mathbb{R}^{K_l}$ are represented as the hidden dimension, the weight matrix and the bias vector of the l-th layer. L is the number of layers. For convenience, we use $\mathbf{W}^+$ to denote the collection of all weights and biases. ϕ^l and $\mathbf{z}^l \in \mathbb{R}^{K_l}$ are the activation function and the activation output vector of layer l, respectively. In our work, we use $\tanh(\cdot)$ as activation function ϕ^l for every layer l.

4 Proposed Methdology

In this section, we first introduce a new recommender framework — Neural Collaborative Autoencoder (NCAE), which consists of an input dropout module, a sparse forward module, a sparse backward module and an error reweighing module. Then we propose a sparsity-aware data-augmentation strategy and a three-stage layer-wise pre-training mechanism to enhance the performance.

4.1 Neural Collaborative Autoencoder

Previous work based on autoencoders can be simply categoried into two style: user-based and item-based [29], [33], [39]. For simplicity, the user-based autoencoder is considered in our paper[2]. The collaborative filtering problem can be interpreted based on autoencoders: *transform the sparse vector* $\mathbf{u}_i$ *into dense vector* $\hat{\mathbf{u}}_i$ *for every user* i. Formally, user-based NCAE is defined as:

$$\hat{\mathbf{u}}_i = nn(\tilde{\mathbf{u}}_i) = \mathbf{z}_i^L = \phi^L(\mathbf{W}^L(..\phi^1(\mathbf{W}^1\tilde{\mathbf{u}}_i + \mathbf{b}^1)..) + \mathbf{b}^L), \tag{1}$$

where the first L-1 layer of NCAE aims at building a low-rank representation for user i, and the last layer L can be regarded as item representation learning or a task-related decoding part. For the explicit setting, we make a slight modification and let $nn(\tilde{\mathbf{u}}_i) = 2.25 * \mathbf{z}_i^L + 2.75$, since ratings are 10 scale with 0.5 constant increments (0.5 to 5.0) and the range of $\tanh(\cdot)$ is [-1,1].

2. Item-based autoencoders will be used for explicit feedback in the experiments.

In general, NCAE is strongly linked with matrix factorization. For NCAE with only one hidden layer and no output activation function, $nn(\tilde{\mathbf{u}}_i) = \mathbf{W}^2\phi^1(\mathbf{W}^1\tilde{\mathbf{u}}_i + \mathbf{b}^1) + \mathbf{b}^2$ can be reformulated as:

$$\hat{\mathbf{u}}_i = nn(\tilde{\mathbf{u}}_i) = \underbrace{[\mathbf{W}^2\ \mathbf{I}_N]}_{\mathbf{V}} \underbrace{\begin{bmatrix} \phi^1(\mathbf{W}^1\tilde{\mathbf{u}}_i + \mathbf{b}^1) \\ \mathbf{b}^2 \end{bmatrix}}_{\mathbf{U}_i}, \tag{2}$$

where $\mathbf{V}$ is the item latent matrix and $\mathbf{U_i}$ is the user latent factors for user i. NCAE with more hidden layers and output activation function ϕ^L can be regarded as *non-linear matrix factorization*, aiming at learning deep representations. A nice benefit of the learned NCAE is that it can fill in every vector $\mathbf{u}_i$, even if that vector was not in the training data.

Fig. 2 shows a toy structure of three-layer NCAE, which has two hidden layers and one output layer. Formal details are as follows.

Input Dropout Module. Like denoising autoencoder (DAE) [34], NCAE learns item correlation patterns via training on a corrupted preference set $\tilde{\mathbf{u}}_i$. The only difference is that NCAE only corrupts observed ratings, i.e., non-zero elements of $\mathbf{u}_i$. As shown in Fig. 2 (left panel), for sparse vector $\mathbf{u}_4$, only v_1, v_3, v_5 are considered and v_5 is finally dropped. This promotes more robust feature learning, since it forces the network to balance the rating prediction and the rating reconstruction.

In general, corruption strategies include two types: the additive Gaussian noise and the multiplicative dropout/mask-out noise. In our work, dropout [32] is used to distinguish the effects of observed items. For every epoch, the corrupted input $\tilde{\mathbf{u}}_i$ is generated from a conditional distribution $p(\tilde{\mathbf{u}}_i|\mathbf{u}_i)$ and non-zero values in $\mathbf{u}_i$ are randomly dropped out (set to 0) independently with probability q:

$$\begin{aligned} P(\tilde{\mathbf{u}}_{id} = \delta\mathbf{u}_{id}) &= 1 - q \\ P(\tilde{\mathbf{u}}_{id} = 0) &= q. \end{aligned} \tag{3}$$

To make the corruption unbiased, uncorrupted values in $\mathbf{u}_i$ are set to $\delta = 1/(1-q)$ times their original values. We find $q = 0.5$ usually leads to good results. When predicting the dense $\hat{\mathbf{u}}_i$, NCAE takes uncorrupted $\mathbf{u}_i$ as input.

Sparse Forward Module. In collaborative filtering, the rating matrix $\mathbf{R}$ is usually very sparse, i.e., more than 95% unobserved values. To handle the sparse input $\mathbf{u}_i$ or corrupted $\tilde{\mathbf{u}}_i$, NCAE will ignore the effects of unobserved values, in both forward stage and backward stage. For missing values in $\mathbf{u}_i$, the input edges (a subset of W^1) and the output edges (a subset of W^L, when training) will be inhibited by zero-masking.

Formally, in the input layer ($l = 0$) and the output layer ($l = L$), there are N nodes for all items. For hidden layer $l < L$, there are K_l nodes, $K_l \ll N$. As shown in Fig. 2 (left panel), we only forward $\tilde{\mathbf{u}}_{41}$, $\tilde{\mathbf{u}}_{43}$, $\tilde{\mathbf{u}}_{45}$ with their linking weights. Specifically, when NCAE first maps the input $\tilde{\mathbf{u}}_i$ to latent representation $\mathbf{z}_i^1$, we only need to forward non-zero elements of $\tilde{\mathbf{u}}_i$ with the corresponding columns of $\mathbf{W}^1$:

$$\mathbf{z}_i^1 = \phi^1(\mathbf{W}^1\tilde{\mathbf{u}}_i + \mathbf{b}^1) = \phi^1(\sum_{j \in \mathcal{R}_i} \mathbf{W}^1_{*,j}\tilde{\mathbf{u}}_{ij}), \tag{4}$$

where $\mathbf{W}^1_{*,j}$ represents the j-th column vector of W^1. This reduces the time complexity from $O(K_1 N)$ to $O(K_1|\mathcal{R}_i|)$,

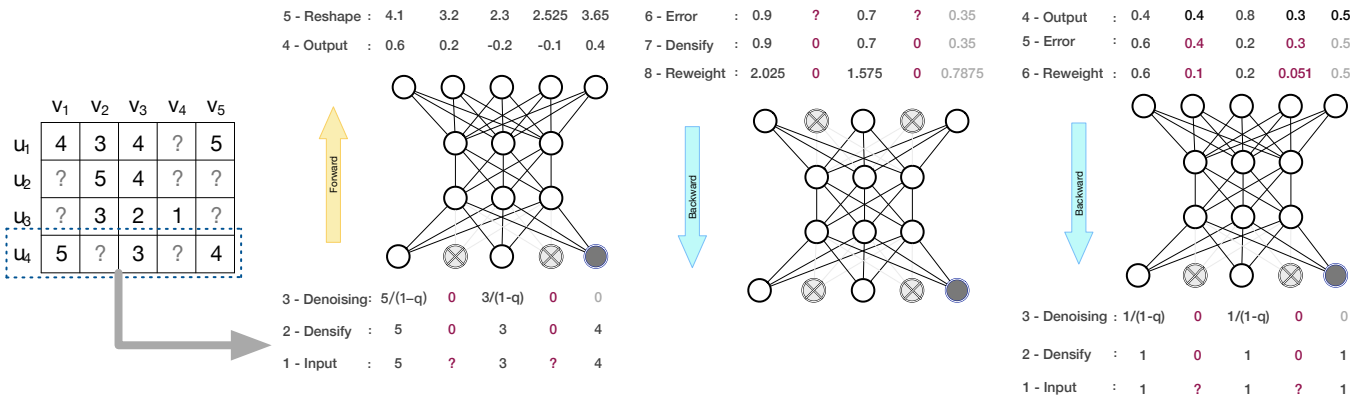


Fig. 2: A three-layer NCAE. Left: Sparse forward procedure with input dropout. Middle: Sparse backward procedure for the explicit feedback, where errors for known values are reweighed by 2.25 due to output reshape ($nn(\tilde{\mathbf{u}}_i) = 2.25 * \mathbf{z}_i^L + 2.75$.) Right: Sparse backward for implicit feedback, where popularity-based error reweighting is used on unknown values.

$|\mathcal{R}_i| \ll N$. For every middle layer, forward and backward procedures follow the same training paradigm like MLP with $O(K_l K_{l-1})$. In the output layer, the latent low-rank representation $\mathbf{z}_i^{L-1}$ is then decoded back to reconstruct the sparse $\mathbf{u}_i$ in training or to predict the dense $\hat{\mathbf{u}}_i$ during inference. Finally, the recommendation list for user i is formed by ranking the unrated items in $\hat{\mathbf{u}}_i$.

Sparse Backward Module. To ensure that unobserved ratings do not bring information to NCAE, errors for them are turned to be zero before back-propagation. In other words, no error is back-propagated for missing values via zero-masking[3]. As shown in Fig. 2 (middle panel), unobserved v_2 and v_4 are ignored, and error for v_3 is computed as $|nn(\tilde{\mathbf{u}}_i)_3 - \mathbf{u}_{i3}|$, similarly for v_1, v_5. Furthermore, we can emphasize the prediction criterion and the reconstruction criterion for errors of observed ratings via two hyperparameters α and β, respectively. Then we employ a new loss function for sparse rating inputs, which separates the two criteria and disregard the loss of unobserved ratings:

$$\mathcal{L}_{2,\alpha,\beta}(\mathbf{u}_i,\tilde{\mathbf{u}}_i) = \frac{\alpha}{|\mathcal{K}(\mathbf{u}_i)|}\left(\sum_{j\in\mathcal{C}(\tilde{\mathbf{u}}_i)\cap\mathcal{K}(\mathbf{u}_i)} [nn(\tilde{\mathbf{u}}_i)_j - \mathbf{u}_{ij}]^2\right) + \frac{\beta}{|\mathcal{K}(\mathbf{u}_i)|}\left(\sum_{j\in\mathcal{K}(\mathbf{u}_i)-\mathcal{C}(\tilde{\mathbf{u}}_i)} [nn(\tilde{\mathbf{u}}_i)_j - \mathbf{u}_{ij}]^2\right), \tag{5}$$

where $\mathcal{K}(\mathbf{u}_i)$ are the indices for observed values of $\mathbf{u}_i$, $\mathcal{C}(\tilde{\mathbf{u}}_i)$ are the indices for dropped elements of $\tilde{\mathbf{u}}_i$. Take the user vector $\mathbf{u}_4$ for example, $\mathcal{K}(\mathbf{u}_4) = \{1,3,5\}$ and $\mathcal{C}(\mathbf{u}_4) = \{5\}$. Different from [33], we divide the loss of DAE by $|\mathcal{K}(\mathbf{u}_i)|$ (equals to $|\mathcal{R}_i|$) to balance the impact of each user i on the whole. Actually, NCAE performs better than other methods even when we directly set $\alpha = \beta = 1$ for all experiments. Finally, we learn the parameters of NCAE by minimizing the following average loss over all users:

$$\mathscr{L} = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}_{2,\alpha,\beta}(\mathbf{u}_i, \tilde{\mathbf{u}}_i) + \frac{\lambda}{2}||\mathbf{W}^+||^2, \tag{6}$$

3. Otherwise, NCAE learns to predict all ratings as 0, since missing values are converted to zero and the rating matrix $\mathbf{R}$ is quite sparse.

where we use the squared L2 norm as the regularization term to control the model complexity.

During back-propagation, only weights that are connected with observed ratings are updated, which is common in MF and RBM methods. In practice, we apply Stochastic Gradient Descent (SGD) to learn the parameters. For user i, take $\frac{\partial \mathscr{L}}{\partial \mathbf{W}_j^L}$, $\frac{\partial \mathscr{L}}{\partial \mathbf{b}_j^L}$ and $\frac{\partial \mathscr{L}}{\partial \mathbf{z}_i^L}$ as a example:

$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}_j^L} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{u}}_{ij}}\frac{\partial \hat{\mathbf{u}}_{ij}}{\partial \mathbf{W}_j^L} + \lambda \mathbf{W}_j^L, \tag{7}$$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{b}_j^L} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{u}}_{ij}}\frac{\partial \hat{\mathbf{u}}_{ij}}{\partial \mathbf{b}_j^L} + \lambda \mathbf{b}_j^L, \tag{8}$$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{z}_i^L} = \sum_{j\in\mathcal{R}_i} \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{u}}_{ij}}\frac{\partial \hat{\mathbf{u}}_{ij}}{\partial \mathbf{z}_i^L}. \tag{9}$$

This leads to a scalable algorithm, where one iteration for user i runs on the order of the number of non-zero entries. The time complexity is reduced from $O(NK_{L-1})$ to $O(|\mathcal{R}_i|K_{L-1})$, $|\mathcal{R}_i| \ll N$.

Error Reweighting Module. As mentioned before, neural network models are easier to overfit on the implicit setting, since only the observed interactions are provided and the model may learn to predict all ratings as 1. To combat overfitting, we introduce a set of variables c_{ij} to determine *which unobserved items a user do not like* and then propose an error reweighting module for implicit NCAE. Specifically, unobserved values of $\mathbf{u}_i$ are turned to zero, and errors for them are first computed as $|nn(\tilde{\mathbf{u}}_i)_j - 0|$, then reweighted by the confidence level c_{ij}. In general, we can parametrize c_{ij} based on user exposure [21] or item popularity [9].

In our work, we implement a simple error reweighting module based on item popularity, since we do not use any auxiliary information (beyond interactions). Similar to [9], we assume that popular items not interacted by the user are more likely to be true negative ones. Therefore, unobserved values of $\mathbf{u}_i$ on popular items means that the user i is not interested with those items [9]. We can reweight errors of unobserved places by item popularity c_j:

$$c_j = c_0 \frac{f_j^\omega}{\sum_{k=1}^N f_k^\omega}, \tag{10}$$

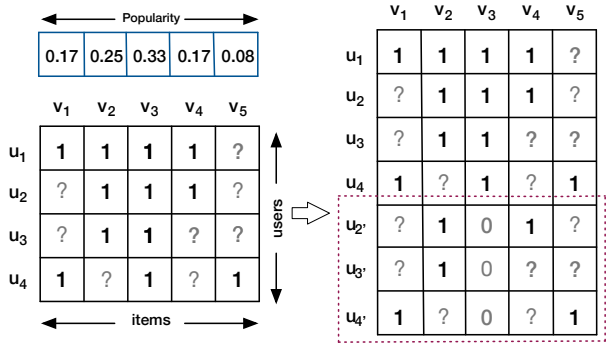


Fig. 3: Sparsity-aware data-augmentation: $\mathbf{u}_2$, $\mathbf{u}_3$ and $\mathbf{u}_4$ are augmented when $\epsilon = 0.8$ and $p = 0.5$. The item popularity vector is pre-computed via Eq. (10) when $\omega = 1$ and $c_0 = 1$.

where f_j denotes the frequency of item j: $|\mathcal{R}_j|/\sum_{k=1}^{N}|\mathcal{R}_k|$. c_0 is the initial confidence for unobserved values and ω controls the importance level of popular items. As shown in Fig. 2 (right panel), we assume that the item popularity vector is precomputed as [0.17, 0.25, 0.33, 0.17, 0.08], and errors for missing places v_2 and v_4 are reweighted by 0.25 and 0.17, respectively. Finally, we propose a new loss function for implicit NCAE:

$$\mathcal{L}_2(\mathbf{u}_i, \tilde{\mathbf{u}}_i) = \sum_{j \in \mathcal{K}(\mathbf{u}_i)} [nn(\tilde{\mathbf{u}}_i)_j - \mathbf{u}_{ij}]^2 + \sum_{j \in \mathcal{J} - \mathcal{K}(\mathbf{u}_i)} c_j [nn(\tilde{\mathbf{u}}_i)_j]^2, \tag{11}$$

where $\mathcal{J} = \{1, 2, ..., N\}$. We set $\alpha = 1$ and $\beta = 1$, and do not divide the loss by $|\mathcal{K}(x)|$, since it is whole-based optimization for all items of $\mathbf{u}_i$. During back-propagation, the time complexity of one user is $O(NK_{L-1})$, which is impractical when the number of items is large. An alternative solution is to build a feature-wise update scheme on the output layer [5], [9], [40], which will reduce the complexity to $O((K_{L-1} + |\mathcal{R}_i|)K_{L-1})$.

4.2 Sparsity-aware Data Augmentation

In general, the input dropout module and the error reweighting module proposed in Sec. 4.1 can provide more item correlation patterns via reweighting the input and the output error, respectively. It is empirically observed that these modules can effectively combat overfitting on the implicit setting. Following this way of thinking, data augmentation strategy can be utilized to provide more item correlation patterns directly with data pre-processing. However, unlike the augmentation on images, which enlarges the dataset using label-preserving transformations like image rotation [16], [31], how to select item correlation patterns *without losing the item ranking order of user preferences* is a key challenge on the implicit setting, since observed interactions are all represented as 1. In general, we have several alternatives to address this problem. For instance, for a particular user, we can reserve items that are viewed the most by him/her or that have similar contents.

As mentioned before, we only use the interaction data. Therefore, we employ a distinct form of augmentation based on item popularity, called sparsity-aware data-augmentation. The main assumptions include two points: only the users whose ratings are sparse need to be augmented, which reduces the biases caused by denser users that have interacted with more items; *less popular items interacted by a user are more likely to reveal his/her interest*. Hence, we can drop popular items from the sparse vector $\mathbf{u}_i$ to construct augmented sample $\mathbf{u}_{i'}$ and reserve the most relevant item correlation patterns.

Fig. 3 shows a toy example, where r_{23} is dropped from u_2 to construct $u_{2'}$, since v_3 is the most popular one that contributes least on the user profile. Specifically, we introduce a sparsity threshold ϵ and a drop ratio p to control the scale of augmented set. The augmentation procedure follows: If $|\mathcal{R}_i|/N < \epsilon$, we will augment $\mathbf{u}_i$ and drop the top $\lfloor |\mathcal{R}_i| * p \rfloor$ popular items to form $\mathbf{u}_{i'}$.

4.3 Greedy Layer-wise Training

For explicit feedback, we can employ the NCAE architecture with stacks of autoencoders to learn deep representations. However, learning is difficult in multi-layer architectures. Generally, neural networks can be pre-trained using RBMs, autoencoders or Deep Boltzmann Machines to initialize the model weights to sensible values [1], [10], [17], [26], [35]. Hinton et al. [10] first proposed a greedy, layer-by-layer unsupervised pre-training algorithm to perform hierarchical feature learning. The central idea is to train one layer at a time with unsupervised feature learning and taking the features produced at that level as input for the next level. The layer-wise procedure can also be applied in a purely supervised setting, called the greedy layer-wise supervised pre-training [2], which optimizes every hidden layer with target labels. However, results of supervised pre-training reported in [2] were not as good as unsupervised one. Nonetheless, we observe that supervised pre-training combined with unsupervised deep feature learning gives the best performance. We now explain how to build a new layer-wise pre-training mechanism for recommender systems, and define:

Definition 1: *Shallow Representation(SR)* for user i is the first layer hidden activation $\mathbf{z}_i^1$ with supervised pre-training, where $\mathbf{z}_i^1 = \phi^1(\mathbf{W}^1\tilde{\mathbf{u}}_i + \mathbf{b}^1)$.

Definition 2: *Deep Representation(DR)* for user i is the $L-1$ layer hidden activation $\mathbf{z}_i^{L-1}$ with unsupervised pre-training using *SR*, where $\mathbf{z}_i^{L-1} = \phi^{L-1}(\mathbf{W}^{L-1}(..\phi^2(\mathbf{W}^2\mathbf{z}_i^1 + \mathbf{b}^2)..) + \mathbf{b}^{L-1})$.

Definition 3: *Item Representation(V)* for user-based NCAE is the L layer weight matrix, where $V = \mathbf{W}^L$. Similar definition for *U* of item-based one.

Following the definitions, the proposed layer-wise pre-training can be divided into three stages: supervised *SR* learning, unsupervised *DR* learning and supervised *V* learning. Here we consider a three-layer NCAE. Formal details are as follows:

Supervised *SR* Learning. To train the weights between the input layer and the *SR* layer, we add a decoder layer on top of *SR* to reconstruct the input, as shown in Fig. 4, stage 1. Formally:

$$\begin{aligned} \mathbf{z}_i^1 &= \phi^1(\mathbf{W}^1\tilde{\mathbf{u}}_i + \mathbf{b}^1) \\ \mathbf{z}_i^{'1} &= \phi^{'1}(\mathbf{W}^{'1}\mathbf{z}_i^1 + \mathbf{b}^{'1}), \end{aligned} \tag{12}$$

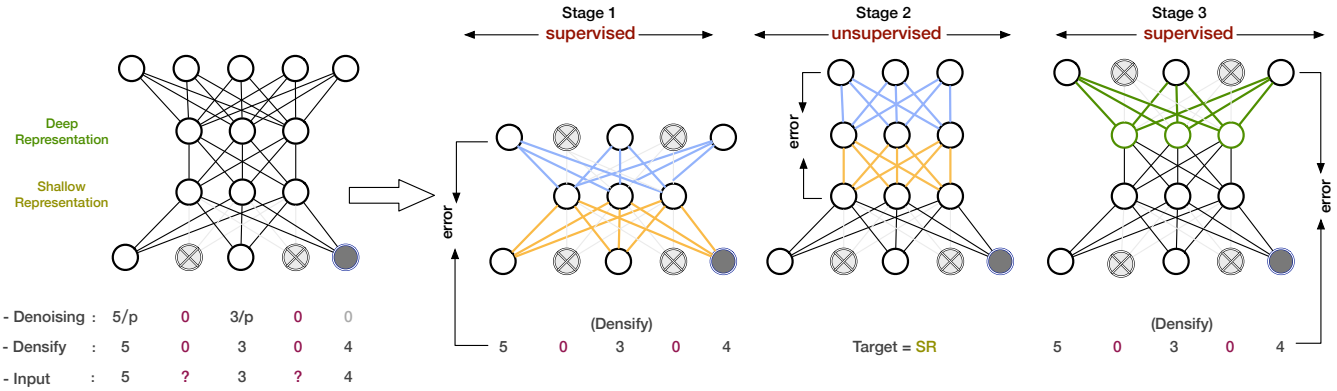


Fig. 4: Left: A three-layer user-based Neural Collaborative Autoencoder with explicit feedback. Right: Proposed greedy layer-wise training mechanism consists of three-stage learning. In the first two stages, an encoder-decoder paradigm is performed to learn *SR* and *DR* with corresponding weights. (The encoder part is plotted in orange, the decoder is plotted in blue.) In stage 3, item representation layer (V, in green) is added on top of *DR* to perform supervised training.

where $\mathbf{W}^{'1}$, $\mathbf{b}^{'1}$, $\mathbf{z}_i^{'1}$ and $\phi^{'1}$ are the weight matrix, the bias vector, the activation output and the activation function for the first decoder. Similar notations for the decoder of layer l are used in the rest of paper. We use $\mathbf{z}_i^{'1} * 2.25 + 2.75$ as estimates $\hat{\mathbf{u}}_i^{'}$ on the explicit setting.

Different from conventional unsupervised pre-training strategies that reconstruct the dense vector of $\tilde{\mathbf{u}}_i$ (or reconstruct observed places of $\tilde{\mathbf{u}}_i$ without introducing noise), we use the supervised signal $\mathbf{u}_i$ to form an encoder-decoder scheme. Therefore, we minimize the loss function $\sum_{i=1}^{M} \mathcal{L}_{2,\alpha,\beta}(\mathbf{z}_i^{'1}, \mathbf{u}_i)/M$ like Eq. (6) to train $\mathbf{W}^1$, $\mathbf{W}^{'1}$, $\mathbf{b}^1$ and $\mathbf{b}^{'1}$. Supervised *SR* pre-training balances the prediction and the reconstruction criteria in the first layer and encourages the network to discover better item correlation patterns of the sparse input.

Unsupervised *DR* Learning. After the *SR* is learned, the weights $\mathbf{W}^1$ and $\mathbf{b}^1$ of the first layer are frozen, and the decoder part of *SR* is removed. Then we add a new hidden layer with its decoder part on top of *SR* layer to perform unsupervised *DR* learning, as shown in Fig. 4, stage 2. Similar to the unsupervised training procedure proposed by [10], [35], only the new added top layer is trained. Thus, we can minimize the square loss between *SR* and corresponding decoder output $\mathbf{z}^{'2}$ to train $\mathbf{W}^2$, $\mathbf{W}^{'2}$, $\mathbf{b}^2$ and $\mathbf{b}^{'2}$:

$$\underset{\mathbf{W}^2,\mathbf{W}^{'2},\mathbf{b}^2,\mathbf{b}^{'2}}{\arg\min} \frac{1}{MK_1} \sum_{i=1}^{M} \sum_{k=1}^{K_1} (\mathbf{z}_{ik}^{'2} - \mathbf{z}_{ik}^{1})^2, \tag{13}$$

where K_1 is the hidden dimension of the *SR* layer. We take average over elements of $\mathbf{z}_i^1$ to ensure *DR* $\mathbf{z}_i^2$ can effectively reconstruct each hidden factor of *SR*. Furthermore, *DR* learning can be implemented by multi-layer autoencoders with unsupervised pre-training.

Supervised V Learning. In this stage, we perform a supervised pre-training for the top layer (see Fig. 4, green connection of V). The main purpose of this stage is to find a better weight initializer for V to reduce the shrinking between layers caused by back-propagating large top error signals when fine-tuning the whole network. Specifically, after *DR* is learned, we fix weights $\mathbf{W}^2$ and $\mathbf{b}^2$ along with $\mathbf{W}^1$, $\mathbf{b}^1$ (or formally $\mathbf{W}^{L-1}$, $\mathbf{b}^{L-1}$, ..., $\mathbf{W}^1$, $\mathbf{b}^1$), and remove the decoder part. However, there is no need for adding a new decoder part in this stage, for the reason $\mathbf{z}_i^L$ is task-related:

$$nn(\tilde{\mathbf{u}}_i) = \mathbf{z}_i^L = \phi^L(\mathbf{W}^L \mathbf{z}_i^{L-1} + \mathbf{b}^L), \tag{14}$$

where L=3 in this case. Similar to stage 1, supervised signal $\mathbf{u}_i$ is used to train V and minimize the following average loss function:

$$\underset{\mathbf{W}^L,\mathbf{b}^L}{\arg\min} \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}_{2,\alpha,\beta}(nn(\tilde{\mathbf{u}}_i), \mathbf{u}_i). \tag{15}$$

After layer-by-layer pre-training, the parameters of NCAE are usually initialized close to a fairly good solution (see Sec. 5.2 for details). Finally, we can perform fine-tuning using supervised back-propagation, minimizing the Eq. (6) corresponding to all parameters $\mathbf{W}^+$ to get a better task-related performance.

5 Experiments

In this section, we conduct a comprehensive set of experiments that aim to answer five key questions: (1) Does our proposed NCAE outperform the state-of-the-art approaches on both explicit and implicit settings? (2) How does performance of NCAE vary with model configurations? (3) Is the three-stage pre-training useful for improving the expressiveness of NCAE? (4) Do the error reweighting module and the sparsity-aware data augmentation work on the implicit setting? (5) Is NCAE scalable to large datasets and robust to sparse data?

5.1 Experimental Setup

We first describe the datasets and the training settings, and then elaborate two evaluation protocols, including evaluation metrics, compared baselines and parameter settings.

Datasets. To demonstrate the effectiveness of our models on both explicit and implicit settings, we used three real-world datasets obtained from MovieLens 10M[4], Delicious[5]

4. https://grouplens.org/datasets/movielens/10m
5. https://grouplens.org/datasets/hetrec-2011

TABLE 1: Statistics of datasets

Dataset	#users	#items	#ratings	sparsity
ML-10M	69,878	10,073	9,945,875	98.59%
Delicious	1,867	69,226	104,799	99.92%
Lastfm	1,892	17,632	92,834	99.72%

and Lastfm[5]. MovieLens 10M dataset consists of user's explicit ratings on a scale of 0.5-5. For Delicious and Lastfm datasets, we consider a user feedback for an item as 1 if the user has bookmarked (or listened) the item. Table 1 summarizes the statistics of datasets.

Training Settings. We implemented NCAE using Python and Tensorflow[6], which will be released publicly upon acceptance. Weights of NCAE are initialized using Xavier-initializer [6] and trained via SGD with a mini-batch size of 128. Adam optimizer [12] is used to adapt the step size automatically with the learning rate set to 0.001. We use grid search to tune other hyperparameters of NCAE and compared baselines on a separate validation set. Then the models are retrained on the training plus the validation set and finally evaluated on the test set. We repeat this procedure 5 times and report the average performance.

Methodology. We evaluate with two protocols:

- Explicit protocol. We randomly split MovieLens 10M dataset into 80%-10%-10% training-validation-test datasets. Since ratings are 10-scale, we use Root Mean Square Error (*RMSE*) to evaluate prediction performance [19], [29], [33]:

$$RMSE = \sqrt{\frac{1}{|\mathbf{R}_{te}|}\sum_{\tau=1}^{|\mathbf{R}_{te}|}\left(R_\tau - \hat{R_\tau}\right)^2} \tag{16}$$

where $|\mathbf{R}_{te}|$ is the number of ratings in the test set, $\mathbf{R}_\tau$ is the real rating of an item and $\hat{R_\tau}$ is its corresponding predicted rating. For the explicit setting, we compare NCAE with the following baselines:

- **ALS-WR** [44] is a classic MF model via iteratively optimizing one parameter with others fixed.
- **BiasedMF** [15] incorporates user/item biases to MF, which performs gradient descent to update parameters.
- **SVDFeature** [3] is a machine learning toolkit for feature-based collaborative filtering, which won the KDD Cup for two consecutive years.
- **LLORMA** [19] usually performs best among conventional explicit methods, which relaxes the low-rank assumption of matrix approximation.
- **I-AutoRec** [19] is a one-hidden layer neural network, which encodes sparse item preferences and aims to reconstruct them in the decoder layer.
- **V-CFN** [33] is a state-of-the-art model for explicit feedback, which is based on DAE. This model can be considered as an extension of AutoRec.

In the experiments, we train a 3-layer item-based NCAE for MovieLens 10M dataset, which has two hidden layers with equal hidden factors K. We employ K as 500 unless otherwise noted. For regularization, we set weight decay $\lambda = 0.0002$ and input dropout ratio $q = 0.5$. Parameter settings for compared baselines are the same in original papers. We compare NCAE with the best results reported in authors' experiments under the same 90% (training+validation)/10% (test) data splitting procedure.

- Implicit protocol. Similar to [9], we adopt the leave-one-out evaluation on Delicious and Lastfm datasets, where the latest interaction of each user is held out for testing. To determine the hyperparameters, we randomly sample one interaction from the remaining data of each user as the validation set. We use Hit Ratio (*HR*) and Normalized Discounted Cumulative Gain (*NDCG*) for top-M evaluations, and report the score averaged by all users. Without special mention, we set the cut-off point to 100 for both metrics. We compare NCAE with the following top-M methods:

- **POP** is a non-personalized model that rank items by their popularity. We use the implementation in [25].
- **BPR** [25] samples negative interactions and optimizes the MF model with a pairwise ranking loss. We set the learning rate and weight decay λ to the best values 0.1 and 0.01.
- **WMF** [11] treats all unobserved interactions as negative instances, weighting them uniformly via a confidence value c. We set c and λ to the best values 0.05 and 0.1.
- **eALS** [9] is a state-of-the-art MF-based model, weighting all unobserved interactions non-uniformly by item popularity. We set λ to the best value 0.05.
- **NCF** [8] is a state-of-the-art neural network model for implicit feedback that utilizes a multi-layer perceptron to learn the interaction function. Similar to [8], we employ a 4-layer MLP with the architecture of $2K \rightarrow K \rightarrow K/2 \rightarrow 1$, where the embedding size for the GMF component is set to K.

We train a 2-layer user-based NCAE for the two datasets. Since it is easier to overfit the training set on the implicit setting, we set input dropout ratio $q = 0.5$ and weight decay $\lambda = 0.01$, and further utilize the error reweighing module and the sparsity-aware data augmentation strategy. Similar to [9], we compute the item popularity using empirical parameters ($c_0 = 512$ and $\omega = 0.5$) for both NCAE and eALS. For a fair comparison, we report the performance of all methods when K is set to [8, 16, 32, 64, 128]. We employ K as 128 for NCAE unless otherwise noted.

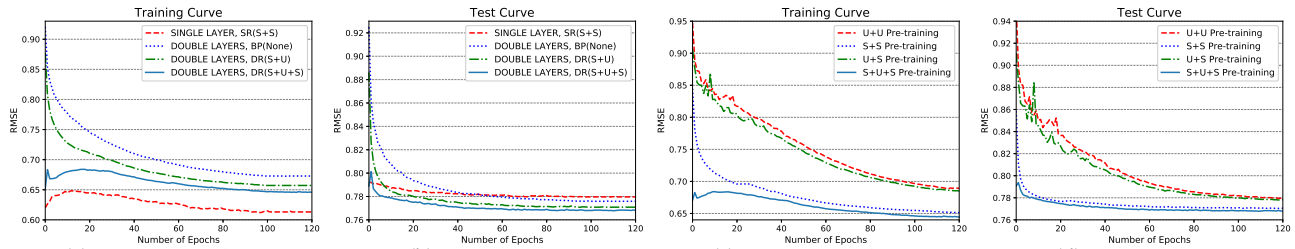
5.2 Explicit Protocol

We first study how model configurations impact NCAE's performance. Then we demonstrate the effectiveness of pre-training. Finally we compare NCAE with explicit baselines.

1) Analysis of Model Configurations

Table 2 shows the performance of NCAE with different configurations, where "SINGLE" means one hidden layer, "*" means the utilization of our pre-training mechanism, and the size of hidden factors K is set to [200, 300, 400, 500]. It can be observed that increasing the size of hidden factors is beneficial, since a larger K generally enlarges the model capacity. Without pre-training, NCAE with two hidden layers may perform worse than that with one hidden layer,

6. https://www.tensorflow.org



(a) Training *RMSE* (b) Test *RMSE* (c) Training *RMSE* (d) Test *RMSE*

Fig. 5: Utility of three-stage pre-training on MovieLens 10M. (a, b) Effect of *SR*, *DR* and *V* pre-training components. (c, d) Performance comparison with layer-wise unsupervised pre-training and layer-wise supervised pre-training strategies.

TABLE 2: Test *RMSE* with varying configurations

Configurations	200	300	400	500
SINGLE	0.791	0.784	0.781	0.779
SINGLE*	0.790	0.783	0.780	0.778
DOUBLE	0.803	0.788	0.781	0.775
DOUBLE*	0.794	0.779	0.771	**0.767**

which is due to the difficulty on training deep architectures. NCAE with pre-training performs better than that without pre-training, especially when a deeper architecture is used.

2) Utility of Pre-training

To demonstrate the utility of three-stage pre-training, we conducted two sub-experiments: effectiveness analysis for *SR*, *DR* and *V* components; performance comparison with conventional pre-training strategies. Firstly, we term "U" as "unsupervised" and "S" as "supervised". Thus, "S+U+S" means our proposed three-stage pre-training method, and "S+U" means "supervised + unsupervised" pre-training for the first two layers. Besides, "BP(None)" means NCAE without pre-training and "DOUBLE" means NCAE with two-hidden layers (3-layer NCAE). For a fair comparison, we set layer-wise pre-training epochs to 10 for each layer of compared strategies. It takes nearly 6.3 seconds per epoch.

Fig. 5(a,b) shows the training/test *RMSE* curves of NCAE on different pre-training configurations, where epoch-0 outputs *RMSE* after pre-training. From it, we get:

- Compared with "BP(None)", "S+U+S" gets lower *RMSE* on epoch-0 in both learning and inferring phases, and brings better generalization after fine-tuning, indicating the usefulness of pre-training for initializing model weights to a good local minimum.
- "S+U+S" gets better generalization on the test set than one-hidden layer NCAE with "S+S", verifying the necessity of *DR* to learn high-level representation of the input. Besides, "S+S" performs best on the training set, but gets worst *RMSE* on the test set (even when we set a larger weight decay λ), which may be caused by overfitting or its model capacity.
- "S+U+S" performs better than "S+U" at the beginning of learning phase, indicating that *V* pre-training can keep the top error signal in a reasonable range, and thus reduce weight shrinking in the lower layers when fine-tuning. Moreover, *RMSE* of "S+U+S" increases slightly at the beginning, which is in accord with the hypothesis of shrinking.

TABLE 3: Test *RMSE* on ML-10M (90%/10%).

Algorithms	*RMSE*
BiasedMF	0.803
ALS-WR	0.795
SVDFeature	0.791
LLORMA	0.782
I-AutoRec	0.782
V-CFN	0.777
NCAE	0.775
NCAE*	**0.767**

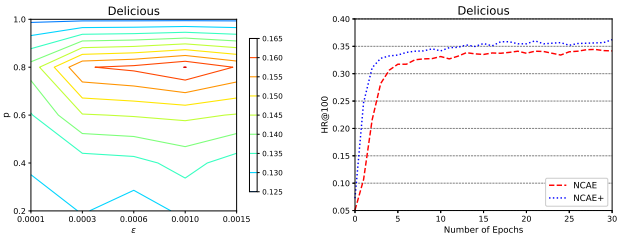
Fig. 5(c,d) shows the performance of 3-layer NCAE with different pre-training strategies. From it, we get:

- "U+U" performs worst in both learning and inferring phases. The greedy layer-wise unsupervised pre-training do not emphasize the prediction criterion. Therefore, the network cannot preserve the task-related information as the layer grows (compared to "U+S" and "S+S") due to the sparsity nature of the recommendation problem.
- "S+U+S" outperforms "S+S" at the beginning of learning and finally gets similar generalization power. This shows the effectiveness of greedy layer-wise supervised pre-training. "S+S" also performs similarly to "S+U" (see DR(S+U)). However, since M and N are usually large, i.e., $M \gg K$ and $N \gg K$, supervised pre-training for all layers is time-consuming and space-consuming. Besides, "S+U+S" can reduce the shrinking of weights.
- Comparing "U+S" with "S+S" and "S+U+S", we can observe that supervised *SR* pre-training in the first layer is critical to the final performance; *SR* learning balances the prediction criterion and the reconstruction criterion, encouraging the network to learn a better representation of the sparse input.

3) Comparisons with Baselines Table 3 shows the performance of NCAE and other baselines on ML-10M, where the results of baselines are taken from original papers under the same 90%/10% data splitting. We conducted one-sample paired t-tests to verify that improvements of NCAE and NCAE* are statistically significant for $sig < 0.005$. NCAE without pre-training has already outperformed other models, which achieves *RMSE* of 0.775. Note that the strong baseline CFN shares a similar architecture like NCAE, but employs well-tuned parameters (α and β) and additional input pre-processing. The new training objective

TABLE 4: Performance of NCAE and NCAE+ on different clusters of users sorted by their ratings.

	NCAE		NCAE+	
Interval	*HR@100*	*NDCG@100*	*HR@100*	*NDCG@100*
0-1(55)	0.055	0.025	0.109	0.064
1-5(68)	0.279	0.072	0.338	0.136
5-10(55)	0.564	0.145	0.555	0.253
10-20(96)	0.323	0.076	0.333	0.154
20-30(78)	0.263	0.078	0.282	0.103



(a) *NDCG*@100 (effect of ϵ, p) (b) *HR*@100 ($\epsilon = 0.001$, $p = 0.8$)

Fig. 6: Effect of data augmentation on Delicious (K=128.)

of NCAE (Eq. 5, which balances the impact of each user) provides further performance gain even without these procedures. The performance can be further enhanced by using three-stage pre-training, with a test *RMSE* of 0.767. This demonstrates that the proposed pre-training method can enhance the expressiveness of NCAE, bringing better generalization.

5.3 Implicit Protocol

We first study how the sparsity-aware data augmentation impacts NCAE's performance. Then we compare NCAE with several top-M methods in detail.

To illustrate the impact of our data-augmentation strategy, we show the performance of NCAE w.r.t. sparsity threshold ϵ and drop ratio p in Fig. 6(a). Noticeably, $p = 1.0$ means that we drop all ratings of one user, i.e., we do not use data-augmentation in this case (with a *NDCG*@100 of 0.125). NCAE achieves the best performance when $\epsilon = 0.001$ and $p = 0.8$, with a *NDCG*@100 of 0.165. This demonstrates that the proposed augmentation strategy can preserve the item ranking order while providing more item correlation patterns, and thus improve the top-M performance. We then study the performance of NCAE on the *HR*@100 metric, where "NCAE+" means NCAE with data augmentation, $\epsilon = 0.001$ and $p = 0.8$. As shown in Fig. 6(b), we can see that the data-augmentation strategy can speed up convergence at the beginning of learning and finally get a better *HR*@100. The relative improvement on *NDCG*@100 (32.0%) is much larger than on *HR*@100 (4.2%), indicating that NCAE+ can discover better ranking positions of true positive items during inference.

Table 4 shows the recommendation performance w.r.t users with different scale of interactions on Delicious. For instance, the second user cluster "1-5(68)" contains 68 users with the number of ratings in $(1, 5]$ (except $[0, 1]$ on the first cluster). As can be seen, NCAE+ consistently outperforms NCAE, with the relative improvements of 127.1%, 55.0%, 36.4%, 52.9%, 19.6% (averaged by two metrics) on the five clusters. The sparsity-aware data augmentation can improve inactive user recommendation, since more item correlation patterns are provided for sparse users.

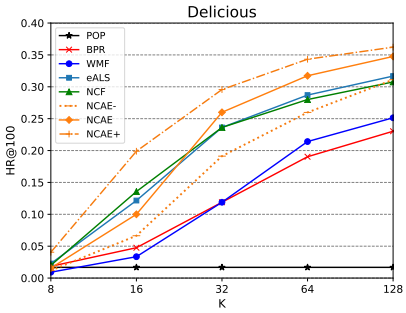
2) Comparisons with Baselines

To illustrate the effectiveness of our NCAE architecture, we propose a variant "NCAE-" that reweights the errors of all unobserved interactions uniformly with a confidence value $c = 0.05$, which is the same to WMF. Specifically, we do not plot the results of NCAE- with $c = 0$, since it is easy to overfit the training set and gives poor predictive performance without the error reweighting module. Besides, "NCAE+" means the utilization of data-augmentation.

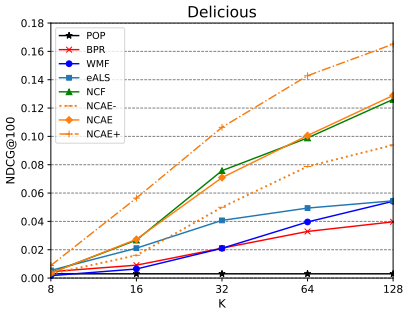
Fig. 7 and Fig. 8 show the performance of *HR*@100 and *NDCG*@100 on Delicious and Lastfm with respect to the number of hidden factors K. First, we can see that, with the increase of K, the performance of most models is greatly improved. This is because a larger K generally enlarges the model capacity. However, large factors may cause overfitting and degrade the performance, e.g., eALS on Lastfm. Second, comparing two whole-based (uniform) methods NCAE- and WMF, we observe that by using non-linear matrix factorization, NCAE- consistently outperforms WMF by a large margin. The high expressiveness of neural network models is sufficient to capture subtle hidden factors when modeling the interaction data. The same conclusion can be obtained from the comparison of two sample-based methods NCF and BPR. Third, NCAE achieves the best performance on both datasets, significantly outperforming the strong baselines NCF and eALS. Specifically, the average improvements of NCAE+ over other models[7] are 170.4%, 312.9%, 170.0%, 126.4%, 111.0% when K=8, 16, 32, 64, 128 on Delicious, verifying the effectiveness of our architecture and sparsity-aware data-augmentation. On Lastfm dataset, NCAE+ do not significantly outperforms NCAE, which is different to that on Delicious. We notice that there are two significant differences on the two datasets: 1) Lastfm is denser than Delicious; 2) Users of Lastfm are more preferred to listen the *popular* artists, which can be observed from the excellent performance of POP method on Lastfm. Thus, our data-augmentation strategy that drops popular items may not work well on Lastfm. The average improvements of NCAE+ over other models are 70.5%, 68.7%, 72.4%, 77.5%, 78.1% when K=8, 16, 32, 64, 128 on Lastfm.

Fig. 9 and Fig. 10 show the performance of Top-M recommendation, where K is set to the best value for each model. First, we can see that eALS and NCF are strong baselines that beats BPR and WMF for all M, which is consistent with the results of [8], [9]. Second, neural network models (e.g., NCF and NCAE-) outperform other models, especially on *NDCG*@M metrics of Delicious dataset, verifying the highly non-linear expressiveness of neural architectures. The performance of NCAE- can be further enhanced via utilizing the popularity-based error reweighting module (NCAE) and the sparsity-aware data augmentation (NCAE+). Lastly, NCAE+ consistently out-

7. We compute the relative improvements on both *HR*@100 and *NDCG*@100 metrics for all models except POP, since POP do not work on Delicious dataset. We report the average improvements of NCAE over all baselines and metrics.

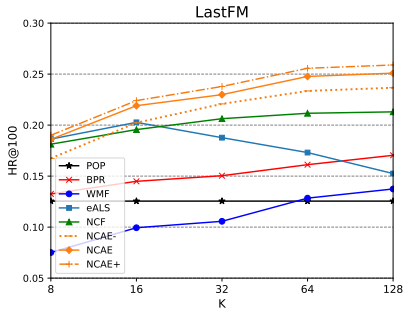


(a) Delicious — *HR*@100

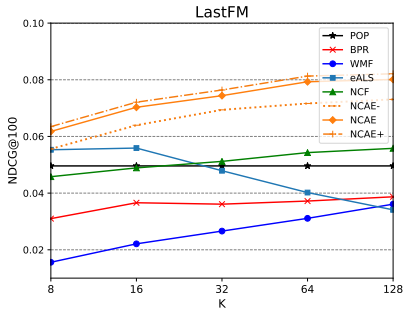


(b) Delicious — *NDCG*@100

Fig. 7: Performance of *HR*@100 and *NDCG*@100 w.r.t. the number of hidden factors on Delicious dataset.

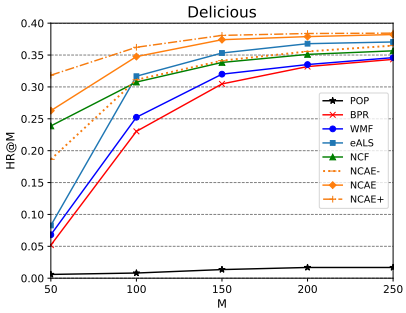


(a) Lastfm — *HR*@100

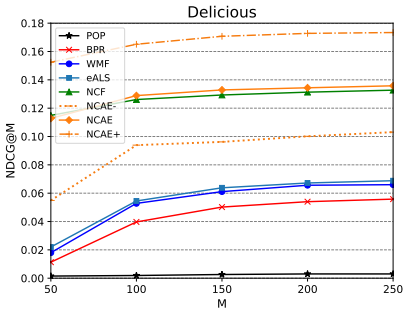


(b) Lastfm — *NDCG*@100

Fig. 8: Performance of *HR*@100 and *NDCG*@100 w.r.t. the number of hidden factors on Lastfm dataset.

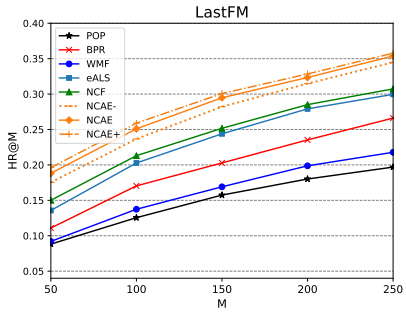


(a) Delicious — *HR*@M

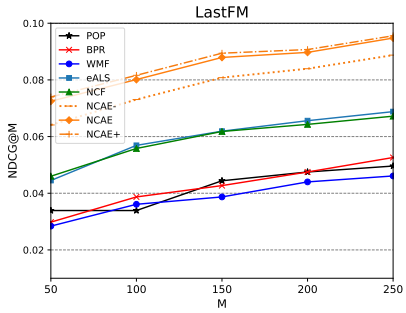


(b) Delicious — *NDCG*@M

Fig. 9: Evaluation of Top-M item recommendation where M ranges from 50 to 250 on Delicious dataset.



(a) Lastfm — *HR*@M



(b) Lastfm — *NDCG*@M

Fig. 10: Evaluation of Top-M item recommendation where M ranges from 50 to 250 on Lastfm dataset.

TABLE 5: Performance on Lastfm with varying data ratio.

Algorithms	Metrics	$D = 40\%$	$D = 60\%$	$D = 80\%$
eALS	*HR@50*	0.082	0.107	0.119
	NDCG@50	0.023	0.030	0.038
	HR@100	0.132	0.165	0.186
	NDCG@100	0.030	0.041	0.047
NCF	*HR@50*	0.101	0.125	0.147
	NDCG@50	0.030	0.040	0.047
	HR@100	0.154	0.174	0.195
	NDCG@100	0.040	0.050	0.056
NCAE+	*HR@50*	0.123	0.155	0.167
	NDCG@50	0.042	0.054	0.061
	HR@100	0.176	0.199	0.222
	NDCG@100	0.050	0.062	0.071

performs the best baseline NCF for every ranking position M, with the relative improvements of 33.0%, 24.3%, 22.3%, 20.5%, 18.1% when M=50, 100, 150, 200, 250 on Delicious, and 41.5%, 30.7%, 29.7%, 26.5%, 28.1% on Lastfm. Note that NCAE has fewer parameters than NCF (NCF stacks three hidden layers while NCAE is a shallow network). The autoencoder-based structures that directly take user vectors as inputs for batch training, may be better for collaborative filtering (the architecture of NCF is designed at the interaction level). Generally, the smaller M is, the larger improvement of NCAE+ against other models, indicating the reliability of our model on top-M recommendation.

5.4 Robustness and Scalability

1) Analysis of Robustness

We now explore the robustness of eALS, NCF and NCAE+ to different proportions of training set exploited. We randomly sample a D proportion of interaction data from each user. The test set and the hyperparameters remain unchanged. As shown in Table 5, we can observe that NCAE+ significantly outperforms the strong baselines eALS and NCF over all ranges of D, and NCAE+ (40%) has already performed slightly better than eALS (80%) and NCF (60%). Specifically, the average improvements of NCAE+ over eALS are 50.4%, 62.4%, 66.3% when training density D=80%, 60%, 40% on top-50 metrics, and 35.2%, 35.9%, 50.0% on top-100 metrics. Generally, as the training set gets sparser, NCAE+ gets a larger improvement against eALS (or NCF). This demonstrates the robustness of our model under the sparsity scenario.

2) Scalability and Computation Time

Analytically, by utilizing the sparse forward module and the sparse backward module[8], the training complexity for one user i is $O(K_1|\mathcal{R}_i| + \sum_{l=1}^{L-2} K_{l+1}K_l + |\mathcal{R}_i|K_{L-1})$, where L is the number of layers and K_l is the hidden dimension of layer l. All our experiments are conducted with one NVIDIA Tesla K40C GPU. For Lastfm and Delicious datasets, each epoch takes only 0.4 seconds and 2.1 seconds respectively, and each run takes 30 epochs.

For ML-10M dataset, it takes about 6.3 seconds and needs 60 epochs to get satisfactory performance. Since ML-10M is much larger than the other two datasets, this shows that NCAE is very scalable. Table 6 shows the average training time per epoch on ML-10M dataset w.r.t. hidden factors K and training set ratio D, where a 3-layer item-based NCAE with equal hidden factors is used. Then the time complexity can be reformulated as $O(K|\mathcal{R}_j| + K^2)$. As can be seen, the runtime scales almost linearly with K, since the modern GPU architecture parallelizes the matrix-vector product operation automatically. Besides, further observation shows that, the runtime grows slowly as the training size D increases, indicating the scalability of NCAE on large datasets. In practice, the second term $O(\sum_{l=1}^{L-2} K_{l+1}K_l)$ overtakes the extra cost.

8. We use *tf.sparse_tensor_dense_matmul()* and *tf.gather_nd()* to implement NCAE.

TABLE 6: Training time per epoch on ML-10M (in seconds).

K	$D = 10\%$	$D = 30\%$	$D = 50\%$	$D = 100\%$
100	1.31	2.34	3.22	4.95
300	1.99	3.10	3.98	5.68
500	2.48	3.85	4.68	6.31

6 Conclusion and Future Work

In this paper, we present a new framework named NCAE for both explicit feedback and implicit feedback, and adapt several effective deep learning approaches to the recommendation domain. Contrary to other attempts with neural networks, we employ a new loss function for sparse inputs, and propose a three-stage pre-training mechanism, an error reweighting module and a data-augmentation strategy to enhance the recommendation performance. We conducted a comprehensive set of experiments to study the impacts of difference components, verifying the effectiveness of our NCAE architecture. We also compared NCAE with several state-of-the-art methods on both explicit and implicit settings. The results show that NCAE consistently outperforms other methods by a large margin.

In future, we will deal with the cold-start problem and explore how to incorporate auxiliary information. We also plan to study the effectiveness of neural network models when an online protocol is used [9]. Another promising direction is to explore the potential of other neural structures for recommendation. Besides, it is important to design an interpretative deep model that can explicitly capture changes in user interest or item properties.

References

[1] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *TPAMI*, 2013.

[2] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *NIPS*, 2007.

[3] T. Chen, W. Zhang, Q. Lu, K. Chen, Z. Zheng, and Y. Yu. Svdfeature: a toolkit for feature-based collaborative filtering. *JMLR*, 2012.

[4] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, 2016.

[5] R. Devooght, N. Kourtellis, and A. Mantrach. Dynamic matrix factorization with priors on unknown values. In *KDD*, 2015.

[6] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[7] C. A. Gomez-Uribe and N. Hunt. The netflix recommender system: Algorithms, business value, and innovation. *TMIS*, 2016.

[8] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. In *WWW*, 2017.

[9] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*, 2016.

[10] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.
[11] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, 2008.
[12] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
[13] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, 2008.
[14] Y. Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 2010.
[15] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 2009.
[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.
[17] N. Le Roux and Y. Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural computation*, 2008.
[18] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 2015.
[19] J. Lee, S. Kim, G. Lebanon, and Y. Singer. Local low-rank matrix approximation. In *ICML*, 2013.
[20] S. Li, J. Kawale, and Y. Fu. Deep collaborative filtering via marginalized denoising auto-encoder. In *CIKM*, 2015.
[21] D. Liang, L. Charlin, J. McInerney, and D. M. Blei. Modeling user exposure in recommendation. In *WWW*, 2016.
[22] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *RecSys*, 2013.
[23] A. Mnih and R. R. Salakhutdinov. Probabilistic matrix factorization. In *NIPS*, 2008.
[24] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *ICDM*, 2008.
[25] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, 2009.
[26] R. Salakhutdinov and G. Hinton. Deep boltzmann machines. In *AISTATS*, pages 448–455, 2009.
[27] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML*, 2007.
[28] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.
[29] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. Autorec: Autoencoders meet collaborative filtering. In *WWW*, 2015.
[30] Y. Shi, M. Larson, and A. Hanjalic. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *CSUR*, 2014.
[31] P. Simard, D. Steinkraus, and J. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, 2003.
[32] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014.
[33] F. Strub, J. Mary, and R. Gaudel. Hybrid collaborative filtering with autoencoders. *arXiv preprint arXiv:1603.00806*, 2016.
[34] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008.
[35] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *JMLR*, 2010.
[36] C. Wang and D. M. Blei. Collaborative topic modeling for recommending scientific articles. In *KDD*, 2011.
[37] H. Wang, N. Wang, and D.-Y. Yeung. Collaborative deep learning for recommender systems. In *KDD*, 2015.
[38] H. Wang, S. Xingjian, and D.-Y. Yeung. Collaborative recurrent autoencoder: Recommend while learning to fill in the blanks. In *NIPS*, 2016.
[39] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *WSDM*, 2016.
[40] H.-F. Yu, C.-J. Hsieh, S. Si, and I. Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *ICDM*, 2012.
[41] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma. Collaborative knowledge base embedding for recommender systems. In *KDD*, 2016.
[42] S. Zhang, L. Yao, and A. Sun. Deep learning based recommender system: A survey and new perspectives. *arXiv preprint arXiv:1707.07435*, 2017.
[43] Y. Zheng, B. Tang, W. Ding, and H. Zhou. A neural autoregressive approach to collaborative filtering. In *ICML*, 2016.
[44] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. *Lecture Notes in Computer Science*, 2008.