Collaborative Denoising Auto-Encoders for Top-N Recommender Systems

Yao Wu
Simon Fraser University
Burnaby, BC, Canada
wuyaow@sfu.ca

Christopher DuBois
Dato Inc.
Seattle, WA, USA
chris@dato.com

Alice X. Zheng
Dato Inc.
Seattle, WA, USA
alicez@dato.com

Martin Ester
Simon Fraser University
Burnaby, BC, Canada
ester@cs.sfu.ca

ABSTRACT

Most real-world recommender services measure their performance based on the top-N results shown to the end users. Thus, advances in top-N recommendation have far-ranging consequences in practical applications. In this paper, we present a novel method, called Collaborative Denoising Auto-Encoder (CDAE), for top-N recommendation that utilizes the idea of Denoising Auto-Encoders. We demonstrate that the proposed model is a generalization of several well-known collaborative filtering models but with more flexible components. Thorough experiments are conducted to understand the performance of CDAE under various component settings. Furthermore, experimental results on several public datasets demonstrate that CDAE consistently outperforms state-of-the-art top-N recommendation methods on a variety of common evaluation metrics.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information Filtering

Keywords

Recommender Systems; Collaborative Filtering; Denoising Auto-Encoders

1. INTRODUCTION

In recent years, recommender systems have become widely utilized by businesses across industries. Given a set of users, items, and observed user-item interactions, these systems can recommend other items that the users might like. Personalized recommendation is one of the key applications of machine learning in e-commerce and beyond. Many recommendation systems use *Collaborative Filtering* (CF) methods to make recommendations. In production, recommender systems are often evaluated based on the performance of the top-N recommendations, since typically only a few recommendations are shown to the user each time. Thus, top-N recommendation methods are of particular interest.

In this paper, we present a new model-based collaborative filtering (CF) method for top-N recommendation called Collaborative Denoising Auto-Encoder (CDAE). CDAE assumes that whatever user-item interactions are observed are a *corrupted* version of the user's full preference set. The model learns latent representations of corrupted user-item preferences that can best reconstruct the full input.[1] In other words, during training, we feed the model a subset of a user's item set and train the model to recover the whole item set; at prediction time, the model recommends new items to the user given the existing preference set as input. Training on corrupted data effectively recovers co-preference patterns. We show that this is an effective approach for collaborative filtering.

Learning from intentionally corrupted input has been widely studied. For instance, Denoising Auto-Encoders [24] train a one-hidden-layer neural network to reconstruct a data point from the latent representation of its partially corrupted version. However, to our best knowledge, no previous work has explored applying the idea to recommender systems.

CDAE generalizes several previously proposed, state-of-the-art collaborative filtering models (see Section 3.2). But its structure is much more flexible. For instance, it is easy to incorporate non-linearities into the model to achieves better top-N recommendation results. We investigate the effects of various choices for model components and compare their performance against prior approaches on three real world data sets. Experimental results show that CDAE consistently outperforms state-of-the-art top-N recommendation methods by a significant margin on a number of common evaluation metrics.

Our contributions can be summarized as follows:

- We propose a new model CDAE, which formulates the top-N recommendation problem using the Auto-Encoder framework and learns from corrupted inputs. Compared to related methods, CDAE is novel in both model definition and objective function.
- We demonstrate that CDAE is a generalization of several state-of-the-art methods but with a more flexible structure.
- We conduct thorough experiments studying the impact of the choices of different components in CDAE, and show that CDAE outperforms state-of-the-art methods on three real world data sets.

The rest of the paper is organized as follows. Section 2 provides the problem definition, background, and useful notations. Section 3 describes our proposed model and learning algorithm in detail. We

[1] We follow the typical top-N recommendation setup and consider only user-item interaction data in this paper. Handling of additional data, such as user/item features and contextual information, is left as future work.

WSDM'16, February 22–25, 2016, San Francisco, CA, USA.

ISBN 978-1-4503-3716-8/16/02. . . $15.00

DOI: http://dx.doi.org/10.1145/2835776.2835837

discuss related work on applying neural network methods to recommender systems in Section 4. Experimental results for the components analysis and performance comparisons are presented in Section 5. We conclude with a summary of this work and discussion of future work in Section 6.

2. PROBLEM DEFINITION

Given a set of users $\mathcal{U} = \{u = 1, ..., U\}$, a set of items $\mathcal{I} = \{i = 1, ..., I\}$, and a log of the users' past preferences of items $\mathcal{O} = (u, i, y_{ui})$, our goal is to recommend to each user u a list of items that will maximize her/his satisfaction. In many cases, the log contains only *implicit* feedback where all the y_{ui} are 1; the rest of the triples are assumed missing. We use $\bar{\mathcal{O}}$ to denote the set of unobserved, missing triples, and $\mathcal{O}'$ an augmented user-item pairs dataset that includes some data sampled from $\bar{\mathcal{O}}$. (We discuss $\mathcal{O}'$ in more detail in the subsection on objective functions.) Let $\mathcal{O}_u$ denote the set of item preferences in the training set for a particular user u, and $\bar{\mathcal{O}}_u$ the unobserved preferences of user u. Items in $\bar{\mathcal{O}}_u$ are the candidates to be recommended to user u. The goal of the recommender is to pick for each user u a subset of items from the candidate set, the predicted values of which are most likely to be 1.

In some cases, y_{ui} are numeric ratings in the range of, say, $[1, 5]$ or binary values $\{0, 1\}$. For simplicity, we only consider the case of implicit feedback in this paper. Numeric ratings can be handled with slight modifications to our method.

In the rest of the paper, we use u to index a user, and i and j to index items. Vectors and matrices are denoted by bold symbols, where symbols in lower case (*e.g.*, $\boldsymbol{x}$) represent vectors and symbols in upper case (e.g., $\boldsymbol{X}$) represent matrices. Unless stated differently, $\boldsymbol{x}_i$ also represents a vector where i is used to distinguish different vectors. We denote the i-th row of matrix $\boldsymbol{X}$ by $\boldsymbol{X}_i$ and its (i, j)-th element by $\boldsymbol{X}_{ij}$.

2.1 Overview of Model-based Recommenders

Most machine learning models can be specified through two components: **model definition** and **objective function** during training. The model definition formulates the relationship between the input (*e.g.*, user ids, item ids, interactions, other features, etc.) and output (ratings or implicit feedback of items). The objective function is what the training process optimizes to find the best model parameters.

Recommender Models

In general, recommender models are defined as

$$\hat{y}_{ui} = \mathcal{F}_{\boldsymbol{\theta}}(u, i), \tag{1}$$

where $\hat{y}_{ui}$ is the predicted preference of user u on item i, and $\boldsymbol{\theta}$ denotes the model parameters we need to learn from training data.

Different choices of the function $\mathcal{F}_{\boldsymbol{\theta}}$ correspond to different assumptions about how the output depends on the input. Here we review 4 common recommender models.

Latent Factor Model (LFM). LFM models the preference $\hat{y}_{ui}$ as the dot product of latent factor vectors $\boldsymbol{v}_u$ and $\boldsymbol{v}_i$, representing the user and the item, respectively [9, 17]. [2]

$$\hat{y}_{ui} = \mathcal{F}_{\boldsymbol{v}}^{LFM}(u, i) = \boldsymbol{v}_u^\top \boldsymbol{v}_i \tag{2}$$

In addition, hierarchical latent factor models [1, 32] and the factorization machine [14] can model interactions between user or item side features.

[2] To simplify notation, we assume that the latent factors are padded with a constant to model the bias.

Similarity Model (SM). The Similarity model [8] models the user's preference for item i as a weighted combination of the user's preference for item j and the item similarity between i and j. It is a natural extension of an item-based nearest neighbor model. The difference is that SM does not use predefined forms of item similarity (*e.g.*, *Jaccard*, *Cosine*). Instead, it learns a similarity matrix from data [12].

$$\hat{y}_{ui} = \mathcal{F}_{\boldsymbol{S}}^{SM}(u, i) = \sum_{j \in \mathcal{O}_u \setminus \{i\}} y_{uj} \cdot \boldsymbol{S}_{ji} \tag{3}$$

Factorized Similarity Model (FSM). The problem with the Similarity Model is that the number of parameters is quadratic in the number of items, which is usually impractical. A straightforward solution is to factorize the similarity matrix into two low rank matrices [8, 7].

$$\hat{y}_{ui} = \mathcal{F}_{\boldsymbol{p},\boldsymbol{q}}^{FSM}(u, i) = \left(\sum_{j \in \mathcal{O}_u \setminus \{i\}} y_{uj} \cdot \boldsymbol{p}_j \right)^\top \boldsymbol{q}_i \tag{4}$$

LFSM (LFM+FSM). The above models can also be combined. For example, combining LFM and FSM results in the model SVD++ [8], which proved to be one of the best single models for the Netflix Prize.

$$\hat{y}_{ui} = \mathcal{F}_{\boldsymbol{p},\boldsymbol{q}}^{LFSM}(u, i) = \left(\sum_{j \in \mathcal{O}_u \setminus \{i\}} y_{uj} \cdot \boldsymbol{p}_j + \boldsymbol{p}_u \right)^\top \boldsymbol{q}_i \tag{5}$$

Objective Functions for Recommenders

Objective functions for training recommender models can be roughly grouped into two groups: point-wise and pair-wise.[3] Pair-wise objectives approximates ranking loss by considering the relative order of the predictions for pairs of items. Point-wise objectives, on the other hand, only depend on the accuracy of the prediction of individual preferences. Pair-wise functions are usually considered to be more suitable for optimizing top-N recommendation performance. However, as we demonstrate in our experiments, this is not necessarily the case for all data sets.

Regardless of the choice of a pair-wise or point-wise objective function, it is critical to properly take into account unobserved feedback within the model. Models that only consider the observed feedback fail to account for the fact that ratings are *not* missing at random. These models are not suitable for top-N recommenders [13, 23].

Let $\ell(\cdot)$ denote a loss function and $\Omega(\boldsymbol{\theta})$ a regularization term that controls model complexity and encodes any prior information such as *sparsity*, *non-negativity*, or *graph regularization*. We can write the general forms of objective functions for recommender training as follows.

Point-wise objective function.

$$\sum_{(u,i) \in \mathcal{O}'} \ell_{point}(y_{ui}, \hat{y}_{ui}) + \lambda \Omega(\boldsymbol{\theta}). \tag{6}$$

Here $\mathcal{O}'$ denotes an augmented dataset that includes unobserved user-item pairs. The problem with using only observed user-item pairs is that, when users provide only implicit "like"s without explicit ratings, all the observed values y_{ui} are equal to 1. In this case, directly optimizing the point-wise objective function over $\mathcal{O}$ leads

[3] Some models use list-wise objective functions [28, 22], but they are not as widely adopted as point-wise and pair-wise objectives.

Table 1: Overview of related model-based recommenders.

Name	Model	Objective Function
MF [9] / PMF [17]	LFM	ℓ_{point}^{SL}
SVD++ [8]	LFSM	ℓ_{point}^{SL}
MMMF [16]	LFM	ℓ_{point}^{HL}
WSABIE [29]	LFM	ℓ_{pair}^{HL}
BPR-MF [15]	LFM	ℓ_{pair}^{LL}
SLIM [12]	SM	ℓ_{point}^{SL}
FISM [7]	FSM	ℓ_{point}^{SL}, ℓ_{pair}^{SL}
WRMF [6]	LFM	*weighted* ℓ_{point}^{SL}
COFI [28]	LFM	NDCG loss
CLiMF [21]	LFM	MRR loss

to a trivial model that predicts all the ratings as 1.[4] A common solution is to augment $\mathcal{O}$ with a subset of *negative* user-item pairs[5] from the unobserved set $\bar{\mathcal{O}} = \{\mathcal{U} \times \mathcal{I}\} \setminus \mathcal{O}$, and train the model on the augmented set $\mathcal{O}'$. Several strategies for sampling negative user-item pairs are discussed in [13]. $\mathcal{O}'$ can also include duplicate samples to simulate weighted feedback (*e.g.*, [6]).

Pair-wise objective function.

$$\sum_{(u,i,j)\in\mathcal{P}} \ell_{pair}(y_{uij}, \hat{y}_{uij}) + \lambda\Omega(\boldsymbol{\theta}), \tag{7}$$

where $y_{uij} = y_{ui} - y_{uj}$, $\hat{y}_{uij} = \hat{y}_{ui} - \hat{y}_{uj}$, and $\mathcal{P}$ is a set of triplets sampled from $\mathcal{O}'$, each of which includes a user u and a pair of items i and j, where usually i is a positive item and j is a negative item.

For both pair-wise and point-wise objective functions, the choice of the loss function $\ell(\cdot)$ is important. Here we list a few commonly used loss functions for both point-wise and pair-wise objectives:

- Square Loss: $\ell^{SL}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$;
- Log Loss: $\ell^{LL}(y, \hat{y}) = \log(1 + \exp(-y \cdot \hat{y}))$;
- Hinge Loss: $\ell^{HL}(y, \hat{y}) = \max(0, 1 - y \cdot \hat{y})$;
- Cross Entropy Loss: $\ell^{CE}(y, \hat{y}) = -y\log(p) - (1 - y)\log(1 - p)$, where $p = \sigma(\hat{y}) = 1/(1 + \exp(-\hat{y}))$.

Note that, for Log and Hinge losses, the value y for the negative examples should be -1 instead of 0.

Table 1 summarizes recent models for top-n recommendation that fit this framework. For explicit feedback, the loss function can be slightly different (*e.g.*,[11]). Also, several recent papers study position-aware pair-wise loss functions (*e.g.*, WARP [29, 30], CLiMF [21]). For this paper, we do not propose any new objective functions. Any objective function that fits the described framework can be used with our model.

To summarize, the two key components of designing model-based recommenders are: 1) a suitable way to represent the relations between inputs and outputs. 2) a proper objective function and a proper way to deal with the relationship between observed and unobserved feedback.

2.2 Denoising Auto-Encoders

A classical auto-encoder [2] is typically implemented as a one-hidden-layer neural network that takes a vector $\boldsymbol{x} \in \mathbb{R}^D$ as input and maps it to a hidden representation $\boldsymbol{z} \in \mathbb{R}^K$ through a mapping function

$$\boldsymbol{z} = h(\boldsymbol{x}) = \sigma\left(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}\right),$$

where $\boldsymbol{W}$ is a $D \times K$ weight matrix and $\boldsymbol{b} \in \mathbb{K}$ is an offset vector. The resulting latent representation is then mapped back to a reconstructed vector $\hat{\boldsymbol{x}} \in \mathbb{R}^D$ through

$$\hat{\boldsymbol{x}} = \sigma\left(\boldsymbol{W}'\boldsymbol{z} + \boldsymbol{b}'\right).$$

The reverse mapping may optionally be constrained by *tied weights*, where $\boldsymbol{W}' = \boldsymbol{W}$.

The parameters of this model are trained to minimize the average reconstruction error:

$$\underset{\boldsymbol{W},\boldsymbol{W}',\boldsymbol{b},\boldsymbol{b}'}{\arg\min} \frac{1}{n}\sum_{i=1}^{n} \ell(\boldsymbol{x}_i, \hat{\boldsymbol{x}}_i), \tag{8}$$

where ℓ is a loss function such as the *square loss* or the *cross entropy loss* mentioned in the previous subsection.

The Denoising Auto-encoder (DAE) [24] extends the classical auto-encoder by training to reconstruct each data point $\boldsymbol{x}$ from its (partially) corrupted version $\tilde{\boldsymbol{x}}$. The goal of DAE is to force the hidden layer to discover more robust features and to prevent it from simply learning the identity function [24]. The corrupted input $\tilde{x}$ is typically drawn from a conditional distribution $p(\tilde{\boldsymbol{x}}|\boldsymbol{x})$. Common corruption choices are the additive Gaussian noise and the multiplicative mask-out/drop-out noise. Under mask-out/drop-out corruption, one randomly overwrites each of the dimensions of $\boldsymbol{x}$ with 0 with a probability of q:

$$\begin{aligned} P(\tilde{\boldsymbol{x}}_d = \delta\boldsymbol{x}_d) &= 1 - q \\ P(\tilde{\boldsymbol{x}}_d = 0) &= q \end{aligned} \tag{9}$$

To make the corruption unbiased, one sets the uncorrupted values to $\delta = 1/(1 - q)$ times their original value.

3. PROPOSED METHODOLOGY

In this section, we introduce a new model–Collaborative Denoising Auto-Encoder (CDAE). The model learns correlations between the user's item preference by training on a corrupted version of the known preference set. A preference set is binary, *i.e.*, containing only information about whether an item is preferred or not. Therefore, as we will see, CDAE is uniquely suitable for top-N preference recommendations.

3.1 Collaborative Denoising Auto-Encoder

Similar to the standard Denoising Auto-Encoder, CDAE is also represented as a one-hidden-layer neural network. The key difference is that the input also encodes a latent vector for the user, which allows CDAE to be a much better recommender model, as we see in section 5. Figure 1 shows a sample structure of CDAE. CDAE consists of 3 layers, including the input layer, the hidden layer and the output layer.

In the input layer, there are in total $I + 1$ nodes, where each of the first I nodes corresponds to an item, and the last node is a user-specific node (the red node in the figure), which means the node and its associated weights are unique for each user $u \in \mathcal{U}$ in the data. We refer to the first I nodes as *item input nodes*, and the last node as *user input node*. Given the historical feedback $\mathcal{O}$ by users on the item set $\mathcal{I}$, we can transform $\mathcal{O}$ into the training set containing U instances $\{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_U\}$, where $\boldsymbol{y}_u = \{y_{u1}, y_{u2}, ..., y_{uI}\}$ is the I-dimensional feedback vector of user u on all the item in $\mathcal{I}$. $\boldsymbol{y}_u$ is a sparse binary vector that only has $|\mathcal{O}_u|$ non-zero values: $y_{ui} = 1$ if i is in the set $\mathcal{O}_u$, otherwise, $y_{ui} = 0$.

[4] For explicit feedback data, only modeling observed feedback is also insufficient to make good top-N recommendations [13, 23].

[5] Here we use the term *negative* to denote missing feedback.

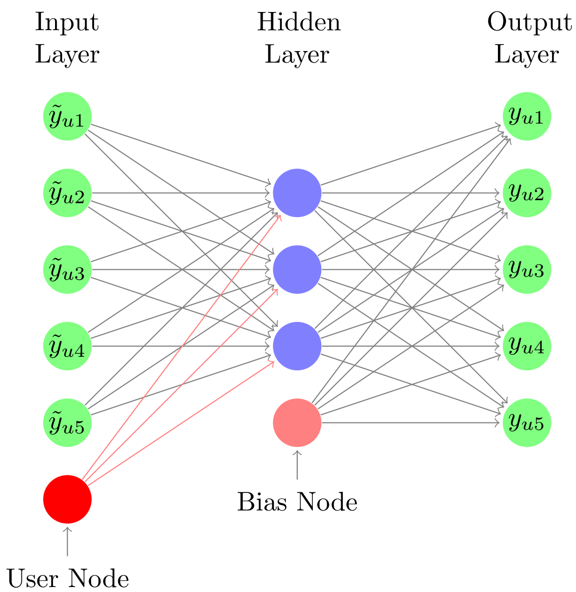


Figure 1: A sample CDAE illustration for a user u. The links between nodes are associated with different weights. The links with *red* color are user specific. Other weights are shared across all the users.

There are K nodes in the hidden layer and these nodes are fully connected to the nodes of the input layer. Here K is a predefined constant which is usually much smaller than the size of the input vectors. The hidden layer also has an additional node to model the bias effects (the pink node in the figure). We use $\boldsymbol{W} \in \mathbb{R}^{I \times K}$ to denote the weight matrix between the *item input nodes* and the nodes in the hidden layer, and $\boldsymbol{V}_u \in \mathbb{R}^K$ to denote the weight vector for the *user input node*. Note that $\boldsymbol{V}_u$ is a user-specific vector, *i.e.*, for each of the users we have one unique vector. From another point of view, $\boldsymbol{W}_i$ and $\boldsymbol{V}_u$ can be seen as the distributed representations of item i and user u respectively.

In the output layer, there are I nodes representing reconstructions of the input vector $\boldsymbol{y}_u$. The nodes in the output layer are fully connected with nodes in the hidden layer. The weight matrix is denoted by $\boldsymbol{W}' \in \mathbb{R}^{I \times K}$. We denote the weight vector for the bias node in the hidden layer by $\boldsymbol{b} \in \mathcal{R}^I$.

Formally, the inputs of CDAE are the corrupted feedback vector $\tilde{\boldsymbol{y}}_u$ which is generated from $p(\tilde{\boldsymbol{y}}_u|\boldsymbol{y}_u)$ as stated in Equation 9. Intuitively, the non-zero values in $\boldsymbol{y}_u$ are randomly dropped out independently with probability q. The resulting vector $\tilde{\boldsymbol{y}}_u$ is still a sparse vector, where the indexes of the non-zero values are a subset of those of the original vector.

CDAE first maps the input to a latent representations $\boldsymbol{z}_u$, which is computed as follows:[6]

$$\boldsymbol{z}_u = h\left(\boldsymbol{W}^\top \tilde{\boldsymbol{y}}_u + \boldsymbol{V}_u + \boldsymbol{b}\right), \tag{10}$$

where $h(\cdot)$ is an *element-wise* mapping function (e.g., *identity function* $h(\boldsymbol{x}) = \boldsymbol{x}$ or *sigmoid function* $h(\boldsymbol{x}) = \sigma(\boldsymbol{x}) = 1/(1+e^{-\boldsymbol{x}})$), and $\boldsymbol{b} \in \mathbb{R}^K$ is the offset vector.

At the output layer, the latent representation is then mapped back to the original input space to reconstruct the input vector. The output value $\hat{y}_{ui}$ for node i is computed as follows:

$$\hat{y}_{ui} = f\left(\boldsymbol{W}_i'^\top \boldsymbol{z}_u + b_i'\right), \tag{11}$$

where $\boldsymbol{W}' \in \mathbb{R}^{I \times K}$ and $\boldsymbol{b}'$ are the weight matrix and the offset vector for the output layer, respectively. $f(\cdot)$ is also a mapping function.

We learn the parameters of CDAE by minimizing the average reconstruction error:

$$\underset{\boldsymbol{W},\boldsymbol{W}',\boldsymbol{V},\boldsymbol{b},\boldsymbol{b}'}{\arg\min} \frac{1}{U}\sum_{u=1}^{U} \mathbb{E}_{p(\tilde{\boldsymbol{y}}_u|\boldsymbol{y}_u)}\left[\ell\left(\tilde{\boldsymbol{y}}_u, \hat{\boldsymbol{y}}_u\right)\right] + \mathcal{R}\left(\boldsymbol{W}, \boldsymbol{W}', \boldsymbol{V}, \boldsymbol{b}, \boldsymbol{b}'\right), \tag{12}$$

where $\mathcal{R}$ is the regularization term to control the model complexity. In this paper we use the squared $L2$ Norm.

$$\mathcal{R}(\cdot) = \frac{\lambda}{2}\left(\|\boldsymbol{W}\|_2^2 + \|\boldsymbol{W}'\|_2^2 + \|\boldsymbol{V}\|_2^2 + \|\boldsymbol{b}\|_2^2 + \|\boldsymbol{b}'\|_2^2\right) \tag{13}$$

We apply Stochastic Gradient Descent (SGD) to learn the parameters. Algorithm 1 provides the detailed procedure. Because the number of output nodes equals the number of items, the time complexity of one iteration over all users is $O(UIK)$, which is impractical when the number of users and the number of items are large. Instead of computing the gradients on all the outputs, we only sample a subset of the negative items $\mathcal{S}_u$ from $\bar{\mathcal{O}}_u$ and compute the gradients on the items in $\mathcal{O}_u \cup \mathcal{S}_u$. The size of $\mathcal{S}_u$ is proportional to the size of $\mathcal{O}_u$. So the overall complexity of the learning algorithm is linear in the size of $\mathcal{O}$ and the number of latent dimensions K. A similar method has been discussed in [5]. An alternative solution is to build a *Hierarchical Softmax* tree on the output layer [10], but it requires the loss function on the output layer to be softmax loss.

Algorithm 1 Learning algorithm for CDAE

Initialize parameters with random values.
$iter \leftarrow 0$
while $iter < maxIter$ or error on validation set decreases **do**
  for all $u \in \mathcal{U}$ **do**
    Sample $\tilde{\boldsymbol{y}}_u \sim p(\tilde{\boldsymbol{y}}_u|\boldsymbol{y}_u)$ using Equation 9.
    Compute $\boldsymbol{z}_u$ using Equation 10.
    Sample negative samples $\mathcal{S}_u \subset \bar{\mathcal{O}}_u$.
    for all $i \in \mathcal{O}_u \cup \mathcal{S}_u$ **do**
      Update $\boldsymbol{W}_i'$ and b_i' using Equation 14, 15 and 20.
    end for
    Compute $\frac{\partial \ell}{\partial \boldsymbol{z}_u}$ using Equation 16.
    for all $j \in \{j | j \in \mathcal{I}$ *and* $\tilde{y}_{uj} > 0\}$ **do**
      Update $\boldsymbol{W}_j$ using Equation 17 and 20.
    end for
    Update $\boldsymbol{V}_u$ using Equation 18 and 20.
    Update $\boldsymbol{b}$ using Equation 19 and 20.
  end for
  $iter \leftarrow iter + 1$
end while

The gradients for the parameters are as follows:

$$\frac{\partial \ell}{\partial \boldsymbol{W}_i'} = \frac{\partial \ell}{\partial \hat{y}_{ui}}\frac{\partial \hat{y}_{ui}}{\partial \boldsymbol{W}_i'} + \lambda \boldsymbol{W}_i' \tag{14}$$

$$\frac{\partial \ell}{\partial b_i'} = \frac{\partial \ell}{\partial \hat{y}_{ui}}\frac{\partial \hat{y}_{ui}}{\partial b_i'} + \lambda b_i' \tag{15}$$

$$\frac{\partial \ell}{\partial \boldsymbol{z}_u} = \sum_{i \in \mathcal{O}_u \cup \mathcal{S}_u} \frac{\partial \ell}{\partial \hat{y}_{ui}}\frac{\partial \hat{y}_{ui}}{\partial \boldsymbol{z}_u} + \lambda \boldsymbol{z}_u \tag{16}$$

[6] Here we compute the hidden representation using the sum of the weight vectors. Other choices such as concatenation and max pooling are also possible.

Table 2: Sample Mapping Functions. Note that all the operations in this table are element-wise.

	$h(\boldsymbol{x})$	Gradient $\frac{\partial h}{\partial \boldsymbol{x}}$
Identity	$\boldsymbol{x}$	$\boldsymbol{1}$
Sigmoid	$\sigma(\boldsymbol{x})$	$\sigma(\boldsymbol{x})(\boldsymbol{1}-\sigma(\boldsymbol{x}))$
Tanh	$\tanh(\boldsymbol{x})$	$\boldsymbol{1}-\tan^2(\boldsymbol{x})$

$$\frac{\partial \ell}{\partial \boldsymbol{W}_j} = \frac{\partial \ell}{\partial \boldsymbol{z}_u}\frac{\partial \boldsymbol{z}_u}{\partial \boldsymbol{W}_j} + \lambda \boldsymbol{W}_j \tag{17}$$

$$\frac{\partial \ell}{\partial \boldsymbol{V}_u} = \frac{\partial \ell}{\partial \boldsymbol{z}_u}\frac{\partial \boldsymbol{z}_u}{\partial \boldsymbol{V}_u} + \lambda \boldsymbol{V}_u \tag{18}$$

$$\frac{\partial \ell}{\partial \boldsymbol{b}} = \frac{\partial \ell}{\partial \boldsymbol{z}_u}\frac{\partial \boldsymbol{z}_u}{\partial \boldsymbol{b}} + \lambda \boldsymbol{b} \tag{19}$$

We apply AdaGrad [4] to automatically adapt the step size during the learning procedure. The update formula is as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta \boldsymbol{g}_\theta^{(t)}}{\sqrt{\beta + \sum_{s=1}^{t} \boldsymbol{g}_\theta^{(s)}}}, \tag{20}$$

where $\boldsymbol{\theta}^{(t)}$ is the value of the parameter $\boldsymbol{\theta}$ at the t-th SGD step and $\boldsymbol{g}_\theta^{(t)}$ is its gradient at step t.

Recommendation. At prediction time, CDAE takes user u's existing reference set (without corruption) as input, and the items from the candidate set $\bar{\mathcal{O}}_u$ that have largest prediction values on the output layer are recommended to him/her.

3.2 Discussion

Components. CDAE is very flexible in that the mapping function $h(\cdot)$, point-wise or pair-wise objectives, the loss function $\ell(\cdot)$, and the corruption probability q can all be chosen to suit the application. Table 2 lists mapping function choices explored in this paper. As for the loss function, all the choices discussed in Section 2.1, both *point-wise* and *pair-wise*, can be used. Different choices of these functions result in different variants of the model with different representation capabilities. Our experiments show that no single variant always produces the best results. One benefit of the proposed general framework is that one can try several variants and find the one that best fits the task.

It's worth noting that pair-wise objective does not perform much better than point-wise objective for CDAE, at least in our experiments. Similar results have also been reported in [7]. This may be a characteristic of implicit feedback data. See section 5.4 for more details.

Generalization of other models. CDAE is a generalization of latent factor models. The representations mentioned in Section 2.1 can all be interpreted as special cases of this framework.

Specifically, if we choose the *identity* mapping function for both $h(x)$ and $f(x)$ and not add noise to the inputs, the output value of $\hat{y}_{ui}$ in Equation 11 becomes:

$$\begin{aligned}\hat{y}_{ui} &= f\left(\boldsymbol{W}_i'^{\top}\boldsymbol{z}_u + b_i'\right)\\ &= \boldsymbol{W}_i'^{\top} h\left(\boldsymbol{W}^{\top}\tilde{\boldsymbol{y}}_u + \boldsymbol{V}_u + \boldsymbol{b}\right) + b_i'\\ &\cong \boldsymbol{W}_i'^{\top}\left(\sum_{j\in\mathcal{O}_u}\tilde{y}_{ui}\boldsymbol{W}_j + \boldsymbol{V}_u\right).\end{aligned} \tag{21}$$

In the last step we omit the bias term to make the comparison clearer. We can see that the representation in Equation 21 is equivalent to that in Equation 5, *i.e.*, the **LFSM** model.

If we set the corruption level q to 1, all the non-zero values in the input vector would be dropped out. We get the following prediction:

$$\hat{y}_{ui} = \boldsymbol{W}_i'^{\top}\boldsymbol{V}_u, \tag{22}$$

which is equivalent to the representation in Equation 2, *i.e.*, the LFM model. Alternatively, if we remove the *user input node* and its associated weights, the resulting model is equivalent to **FSM** in Equation 4:

$$\hat{y}_{ui} = \boldsymbol{W}_i'^{\top}\left(\sum_{j\in\mathcal{O}_u}\tilde{y}_{ui}\boldsymbol{W}_j\right). \tag{23}$$

Another possible mapping function is the *linear* function $h(\boldsymbol{x}) = \boldsymbol{U}^{\top}\boldsymbol{x}$, where $\boldsymbol{U}$ is a $K\times K$ transform matrix. If we use a user-specific matrix $\boldsymbol{U}_u \in \mathbb{R}^{K\times K}$ on the hidden layer, the representation becomes

$$\hat{y}_{ui} = \boldsymbol{W}_i'^{\top}\left(\boldsymbol{U}_u^{\top}\left(\sum_{j\in\mathcal{O}_u}\tilde{y}_{ui}\boldsymbol{W}_j\right)\right), \tag{24}$$

which is related to the Latent Collaborative Retrieval model proposed in [30].

We tried the *linear* mapping function in our experiments, but found that its performance is not as good as others. One reason is likely that it has many more parameters to estimate, which can easily lead to overfitting. We also experimented with setting $\boldsymbol{U}_u$ to be a diagonal matrix to reduce the number of parameters, but the improvements over the identity were still not significant. With this in mind, we do not include the linear mapping function in Table 2, nor report the results in the rest of this paper.

Summary. CDAE is a flexible framework for top-N recommendation. It generalizes several very popular existing methods. The proposed framework is naturally compatible with the denoising trick, which can further improve the results of recommendation, as shown in the next section.

4. RELATED WORK

An overview of the model-based collaborative filter methods has been discussed in Section 2.1. In this section, we discuss the few related work on neural networks for recommender systems.

Restricted Boltzmann Machines (RBM) [18] is the first work that applies neural network models to recommender systems. However, RBM targets rating prediction, not top-N recommendation, and its loss function considers only the observed ratings. It is technically challenging to incorporate negative sampling, which would be required for top-N recommendation, into the training of RBM. For this reason, we do not compare with RBM in our experiments, but test several other neural network baselines that work for top-N recommendation (see Section 5.4).

We are aware of a concurrent proposal called AutoRec [20], which uses the Auto-Encoder for rating prediction. The main differences are as follows: 1) AutoRec only considers the observed ratings in the loss function, which does not guarantee the performance for top-N recommendation. 2) They use the vanilla Auto-Encoder structure, while we prove that introducing user factors in the model can greatly improve performance. 3) AutoRec does not employ the denoising technique, which is a major part of our work.

Another related work is [27], which also uses the Auto-Encoder for recommender systems. This work studies the particular problem of article recommendation, and improves the well-known model Collaborative Topic Regression [26] by replacing its Topic Model component by a Bayesian Auto-Encoder, which is used for learning the latent feature representations for the articles. Different from

Table 3: Dataset Statistics

	#users	#items	#dyads	density(%)
ML	69K	8.8K	5M	0.82
Netflix	37K	11K	4.8M	1.18
Yelp	9.6K	7K	243K	0.36

this model, our model is a generic model and addresses the general top-N recommendation problem, and the inputs are user behaviors instead of item/article features.

5. EXPERIMENTAL RESULTS

Our experimental evaluation consists of two parts. First, we study the effects of various choices of the components of CDAE. Second, we compare CDAE against other state-of-the-art top-N recommendation methods. The source code of CDAE will be available from https://github.com/jasonyaw/CDAE.

5.1 Data Sets and Experimental Setup

We use 3 popular data sets: MovieLens 10M (ML)[7], Netflix[8] and Yelp (from Yelp Dataset Challenge[9] in 2014). For each data set, we keep those with ratings no less than 4 stars and treat all other ratings as missing entries. Those ratings that are retained are converted to a y_{ui} score of 1. This processing method is widely used in previous work on recommendation with implicit feedback (e.g., [15, 31, 7]). We iteratively remove users and items with fewer than 5 ratings. For each user, we randomly hold 20% of the ratings in the test set, and put the other ratings in the training set. The statistics of the resulting data sets are shown in Table 3.

5.2 Implementation Details

We perform 5-fold cross validation on the training data sets to select the best hyperparameters for all the models, and then use the best hyperparameters to train models on the whole training data sets.

We use Stochastic Gradient Decent (SGD) to learn the parameters for both the proposed method and comparison partners. AdaGrad [4] is used to automatically adapt the step size during the learning procedures. We set $\beta = 1$ and try different step sizes $\eta \in \{1, 0.1, 0.01, 0.001\}$ and report the best result for each model.

For negative sampling, we experiment with different numbers of negative samples and find that $NS = 5$ consistently produces good results. This means that, for each user, the number of negative samples is 5 times the number of observed ratings of this user.

5.3 Evaluation Metrics

In the case of top-N recommender systems, we present each user with N items that have the highest predicted values but are not adopted by the user in the training data. We evaluate different approaches based on which of the items are actually adopted by the user in the test data.

Precision and Recall. Given a top-N recommendation list $C_{N,\text{rec}}$, precision and recall are defined as

$$\begin{aligned} \text{Precision}@N &= \frac{|C_{N,\text{rec}} \bigcap C_{\text{adopted}}|}{N} \\ \text{Recall}@N &= \frac{|C_{N,\text{rec}} \bigcap C_{\text{adopted}}|}{|C_{\text{adopted}}|}, \end{aligned} \tag{25}$$

where C_{adopted} are the items that a user has adopted in the test data. The precision and recall for the entire recommender system are computed by averaging the precision and recall over all the users, respectively.

Mean Average Precision (MAP). Average precision (AP) is a ranked precision metric that gives larger credit to correctly recommended items in top ranks. AP@N is defined as the average of precisions computed at all positions with an adopted item, namely,

$$\text{AP}@N = \frac{\sum_{k=1}^{N} \text{Precision}@k \times \text{rel}(k)}{\min\{N, |C_{\text{adopted}}|\}}, \tag{26}$$

where $\text{Precision}(k)$ is the precision at cut-off k in the top-N list $C_{N,\text{rec}}$, and $\text{rel}(k)$ is an indicator function equaling 1 if the item at rank k is adopted, otherwise zero. Finally, Mean Average Precision (MAP@N) is defined as the mean of the AP scores for all users.

Usually, these metrics are consistent with each other, *i.e.*, if a model performs better than another model on one metric, it is more likely that it will also produce better results on another metric. Due to space limits, we mainly show the results of MAP@N with $N = \{1, 5, 10\}$ on several evaluation tasks since it takes the positions into consideration.

5.4 Analysis of CDAE Components

The main components of the proposed CDAE model include the types of the mapping functions, the loss function and the level of corruption. Different choices of these components result in different variants of the model that make different top-N recommendations. In this subsection, we study these variants on the 3 data sets.

For the mapping function, we show results for the *identity* function and *sigmoid* function on the hidden layer and the output layer. (Results for the *tanh* function are similar to those of the *sigmoid* function and hence omitted.) There are $2^3 = 8$ total combinations of choices for the mapping functions (on both layers) and the loss function. Among them, the logistic loss function requires $\hat{y}$ to be a value between 0 and 1, so it must be associated with a sigmoid function on the output layer. Note that the combination of the sigmoid function and the logistic loss is equivalent to the cross entropy loss discussed in Section 2.1. Therefore, we study 4 variants[10] of our model in this subsection. Table 4 describes the function choices for each variant.

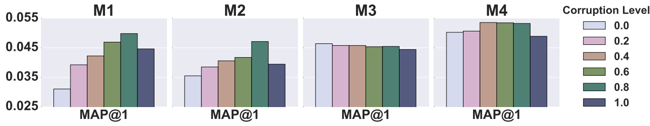
Table 4: Four possible variants of the CDAE model.

	Hidden Layer	Output Layer	Loss Function
M1	Identity	Identity	Square
M2	Identity	Sigmoid	Logistic
M3	Sigmoid	Identity	Square
M4	Sigmoid	Sigmoid	Logistic

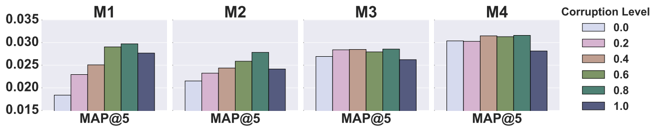
In our extensive experiments, we observed that the pair-wise objective function did not perform much better than point-wise objectives for CDAE. One possible cause is that for the implicit feedback with binary ratings, point-wise loss functions are sufficient for separating those items preferred by the user and those not preferred. In other words, a well-designed point-wise loss can be discriminating enough to model the user's preference in these datasets, and the pair-wise loss is not needed. For this reason, results on pair-wise objective functions are omitted in the rest of this paper. On a related note, as we show in section 5.5, the BPR model, which uses pair-wise loss, does not perform better than MF, which uses point-wise

[7]http://grouplens.org/datasets/movielens
[8]http://www.netflixprize.com
[9]http://www.yelp.com/dataset_challenge

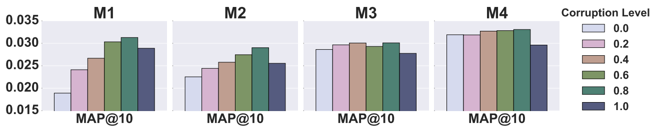
[10]We omit the results of another 2 variants (replacing the loss functions of M2 and M4 with square loss) since their performances are similar to those of M2 and M4 respectively.



(a) MAP@1

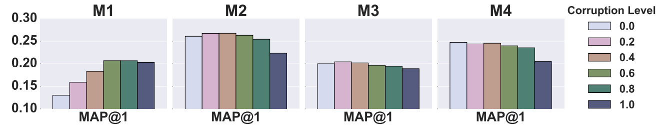


(b) MAP@5

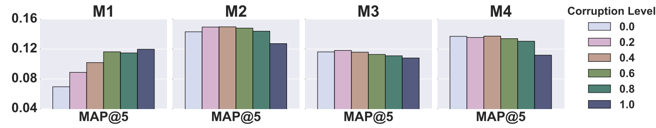


(c) MAP@10

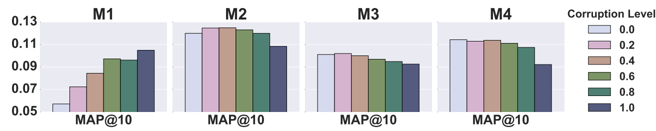
Figure 2: Model performance comparison on Yelp data.



(a) MAP@1



(b) MAP@5



(c) MAP@10

Figure 3: Model performance comparison on Netflix data.

loss. Similar results have also been reported in [7]. This might be due to the same reason as for CDAE. Moreover, BPR is designed to optimize for the AUC, not top-N metrics such as precision, recall, or MAP. Hence, for multiple models for top-N recommendation, pair-wise loss functions may not be necessary for all data sets.

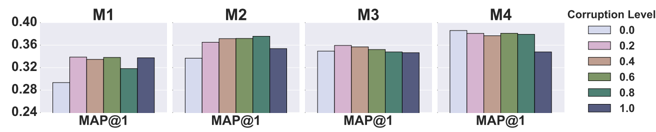
We train each of the four variants under varying corruption levels (q in Equation 9) from $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$. The number of latent dimensions K is set to 50 unless otherwise stated. The results on the three data sets are shown in Figure 2, 3 and 4 respectively.

The general observation is that the best model depends on the data set. No single variant of CDAE always produces the best results. So one should choose the components (mapping function, objective function, loss function, and corruption level) depending on the data. Consider the two different extremes of corruption level: $q = 0$ (no input corruption) and $q = 1$ (complete input corruption). For variant M1, the results of $q = 0$ are much worse than those of $q = 1$. This indicates that simply summing up all the input vectors ($q = 0$) is insufficient for learning good representations, and is in fact worse than dropping out all of them ($q = 1$). Note that introducing non-linear functions can largely alleviate the problem, as evidenced in the results for the other three variants with $q = 0$. Adding noise on the input can also prevent this problem (see the results of M1 with various corruption levels), which means that the denoising technique can help with learning more robust representations.

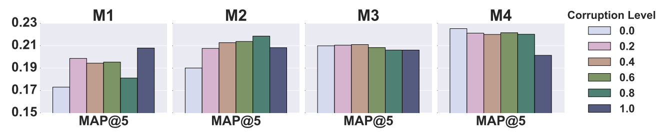
The denoising technique appears beneficial especially for variants M1 and M2. On the Yelp data set, all four variants can be improved by adding relatively higher levels of noise (e.g., $q = 0.8$). Variant M2 is the best model for the Netflix data set, and setting $q = 0.2$ and $q = 0.4$ can slightly improve the results. On MovieLens data, setting $q = 0.8$ makes M2 almost as good as M4, the best model. However, in some cases, the best results are those without any input corruption ($q = 0$).

In general, M4 produces relatively better results on all three data sets. In particular, it achieves the best MAP scores on Yelp and MovieLens. This indicates that non-linear functions help to increase the representation capability of the model, thus improving the recommendations.

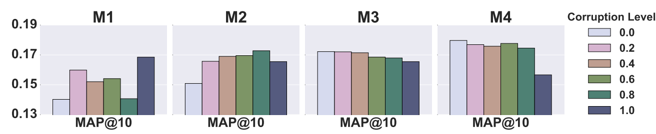
Comparison with DAE. A main difference between CDAE and classical DAE is the user-specific input between the input layer and the hidden layer, namely, the vector $\boldsymbol{V}_u$. We study two cases here – with the user-specific vectors (CDAE) and without the user-specific vectors (DAE). We conduct experiments on the three data sets, and get relatively similar results. We show the results on the Yelp data and on the MovieLens data in Table 5 and 6 respectively. We can see that for models M1 and M2, using user-specific vectors greatly improves the results on the Yelp data. As the performances of the models get better for M3 and M4, the gain becomes relatively smaller. For the MovieLens data, using user-specific vectors consistently outperforms the alternative choice.



(a) MAP@1



(b) MAP@5



(c) MAP@10

Figure 4: Model performance comparison on MovieLens data.

Table 5: Comparison with DAE on the Yelp data.

	MAP@1		MAP@5		MAP@10	
Model	**DAE**	**CDAE**	**DAE**	**CDAE**	**DAE**	**CDAE**
M1	0.0275	0.0327	0.0164	0.0201	0.0172	0.0210
M2	0.0369	0.0420	0.0225	0.0241	0.0236	0.0254
M3	0.0443	0.0460	0.0270	0.0285	0.0289	0.0303
M4	0.0529	0.0528	0.0315	0.0319	0.0329	0.0334

Table 6: Comparison with DAE on the MovieLens data.

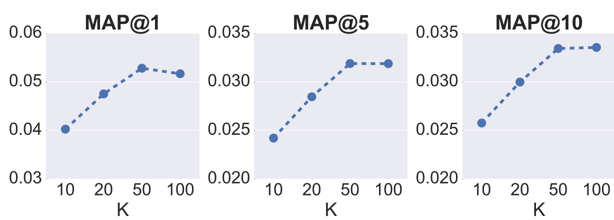
	MAP@1		MAP@5		MAP@10	
Model	**DAE**	**CDAE**	**DAE**	**CDAE**	**DAE**	**CDAE**
M1	0.2807	0.2933	0.1619	0.1730	0.1278	0.1402
M2	0.3134	0.3368	0.1785	0.1900	0.1408	0.1509
M3	0.3270	0.3494	0.1953	0.2099	0.1579	0.1722
M4	0.3625	0.3860	0.2123	0.2252	0.1684	0.1797

Table 7: Effects of using tied weights on MovieLens data. "TW" means using tied weights, while "NTW" means no tied weights.

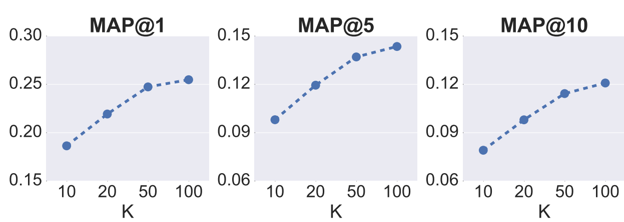
	MAP@1		MAP@5		MAP@10	
Model	**TW**	**NTW**	**TW**	**NTW**	**TW**	**NTW**
M1	0.1739	0.2933	0.0983	0.1730	0.0763	0.1402
M2	0.3707	0.3368	0.2086	0.1901	0.1643	0.1509
M3	0.3482	0.3494	0.2044	0.2099	0.1669	0.1722
M4	0.3530	0.3860	0.2007	0.2252	0.1609	0.1797

Table 8: Effects of using tied weights on Netflix data. "TW" means using tied weights, while "NTW" means no tied weights.

	MAP@1		MAP@5		MAP@10	
Model	**TW**	**NTW**	**TW**	**NTW**	**TW**	**NTW**
M1	0.1172	0.1301	0.0551	0.0695	0.0428	0.0571
M2	0.2567	0.2608	0.1418	0.1431	0.1177	0.1199
M3	0.1172	0.2000	0.0551	0.1162	0.0428	0.1011
M4	0.2287	0.2474	0.1260	0.1370	0.1066	0.1143



(a) Results on Yelp data.



(b) Results on Netflix data.

Figure 5: The effects of the number of latent dimensions.

Tied weights. We study the effects of using tied weights (TW) for the weight matrices, where we force $\boldsymbol{W} = \boldsymbol{W}'$. Results on MovieLens and Netflix data sets are shown in Table 7 and 8, respectively. We refer to the cases of "no tied weights" as NTW. The results on Yelp data are similar to those on Netflix data, so we omit them here. Other than variant M2 on MovieLens data, the results of NTW are much better than TW. On Netflix data, NTW consistently outperforms TW by at least 10%. On both data sets, the best MAP scores are from models with NTW (M4+NTW in Table 7 and M2+NTW in Table 8, respectively). Thus we do not recommended using tied weights for CDAE.

The number of latent dimensions. We study the effects of the number of latent dimensions K. Results on Yelp data and Netflix data are shown in Figure 5. From the figures, we can see that the performance increases with larger K, but only up to a point. When K becomes large enough, the performance no longer improves and can in fact decrease due to overfitting.

5.5 Experimental Comparisons with Previous Models

In this section, we compare CDAE with a number of popular top-N recommendation methods. Note that comparisons against Denoising Auto-Encoder (DAE) and its non-denoising variant (when $q = 0$) are already discussed in Section 5.4.

These baseline methods are:

- **POP**: Items are recommended based on how many users have rated them.
- **ITEMCF** [19]: We use Jaccard similarity measure and set the number of nearest neighbor to 50.
- **MF (with negative sampling)** [9]: We train the Latent Factor Model with point-wise objective functions (with square, log, hinge and cross entropy losses) and negative sampling[11], and report the best results.
- **BPR** [15]: BPR is the state-of-the-art method for recommendation based on implicit feedback. As discussed in Section 2.1, BPR-MF is a Latent Factor Model with the pair-wise log loss function. In addition to the log loss used in the original paper [15], we also experiment with square and hinge loss functions and report the best results.
- **FISM** [7]: FISM is a variant of FSM with point-wise square loss function[12]. We also test log loss and hinge loss and report the best results.

For all the baseline methods, we carefully choose the hyperparameters by cross validation to ensure fair comparisons. We train the 4 variants of CDAE discussed in Table 4 with different corruption levels, and report the best results. For all the latent factor models (including MF, BPR, FISM and CDAE), we set the number of latent dimensions to 50 and use an additional dimension to model the bias. Other implementation details are as discussed in Section 5.2.

Figure 6, 7 and 8 show the MAP@N scores of all models on Yelp, MovieLens and Netflix, respectively. Since Recall is another widely used metric for comparing top-N recommendation models, we also include plots of the Recall scores on the three data sets in Figure 9, 10 and 11.

In general, the results of MAP and Recall are consistent, *i.e.*, the performance orderings of models are almost the same. One exception is on the Yelp data, where MF gets better MAP@N scores than BPR, but BPR has better Recall@N scores.

According to the results of MAP@10 and Recall@10, CDAE consistently outperforms other compared methods. On the Yelp data, CDAE outperforms the other methods with a large margin on all the evaluation metrics. The MAP@10 score and Recall@10 score of CDAE are at least 15% better than those of the second best model MF. For the Netflix data set, ITEMCF achieves much better results than other methods such as MF, BPR and FISM, particularly on the metrics MAP@1 and Recall@1. CDAE is the only model that can beat ITEMCF on metrics MAP@10 and Recall@10, where CDAE outperforms ITEMCF by around 10%.

It is surprising to see that BPR and FISM achieve lower MAP scores than MF on Yelp and Netflix data sets. The only data set on which they achieve better results is MovieLens, but the performance gains are not significant.

[11] We note that, for implicit feedback data, MF with negative sampling has the same objective function with WRMF [6] – both of them assign high confidence on the observed/positive feedback and low cofidence on the missing/negative feedback. We do not compare with WRMF because its computational speedup trick for ALS only works with squared loss.

[12] We also tried the pair-wise loss functions, but the results are not as good as for the point-wise functions. The same observation is reported in the original paper.

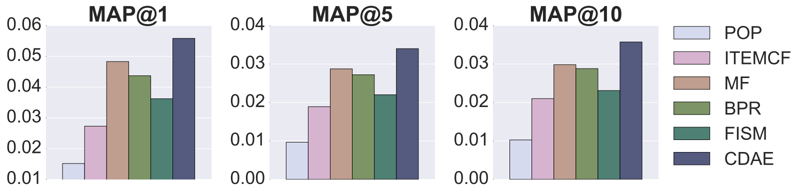


Figure 6: MAP scores with different N on the Yelp data set.

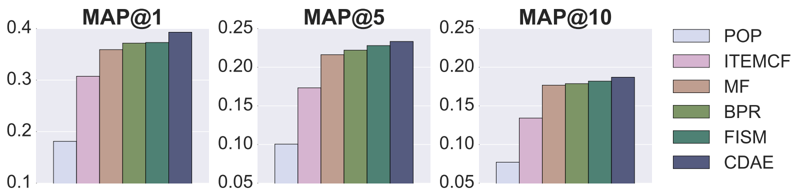


Figure 7: MAP scores with different N on the MovieLens data set.

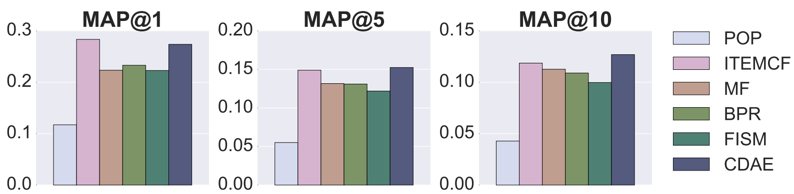


Figure 8: MAP scores with different N on the Netflix data set.

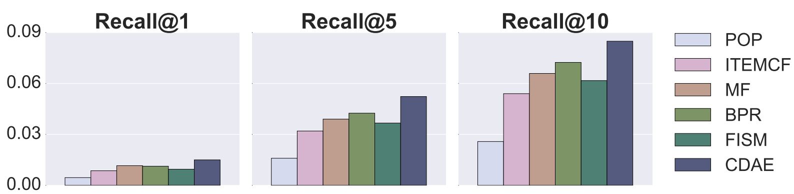


Figure 9: The Recall scores with different N on the Yelp data set.

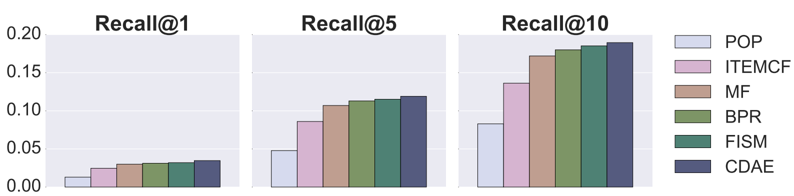


Figure 10: The Recall scores with different N on the MovieLens data set.

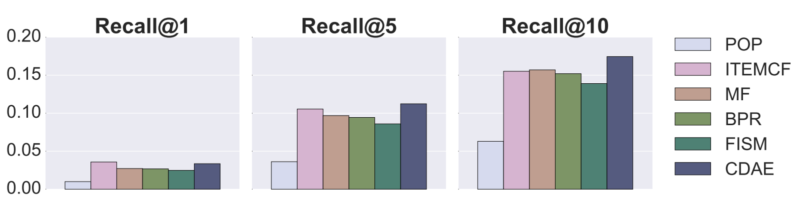


Figure 11: The Recall scores with different N on the Netflix data set.

6. CONCLUSION

In this paper, we presented the Collaborative Denoising Auto-Encoder (CDAE) for the top-N recommendation problem. CDAE learns distributed representations of the users and items via formulating the user-item feedback data using a Denoising Auto-Encoder structure. Several previous work can be seen as special cases of the proposed model. We conducted a comprehensive set of experiments on three data sets to study how the choice of the model components impacts the performance. We also compared CDAE against several other state-of-the-art top-N recommendation methods and the results show that CDAE outperforms the rest of the methods by a large margin.

The proposed model enables a wide range of future work on applying neural networks to recommender systems. Here we list some potential directions.

Deep Neural Network. The neural network structure used in our paper is shallow. A straightforward extension is to stack the model as done in the stacked DAE [25]. Our preliminary experiments on the stacked CDAE do not show significant improvement over CDAE. We plan to investigate the reason and try to improve it. Also, the idea of marginalized DAE [3] might be able to speed up the training and improve the performance. It would also be interesting to consider applying other neural network structures such as Convolutional Neural Networks and Recurrent Neural Networks to this framework.

Feature-aware Recommendation. User and item features can be important for producing semantically meaningful models and dealing with the *cold-start* problem. It is worth exploring how to incorporate user and item features to improve the proposed model.

Context-aware Recommendation. In many applications, one might benefit from incorporating contextual information (such as time, location, browser session, etc.) into the recommendation process in order to recommend items to users in certain circumstances. We leave such extensions as future work.

7. REFERENCES

[1] D. Agarwal and B. Chen. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 19–28, 2009.

[2] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, pages 153–160, 2006.

[3] M. Chen, K. Q. Weinberger, F. Sha, and Y. Bengio. Marginalized denoising auto-encoders for nonlinear representations. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1476–1484, 2014.

[4] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[5] X. Glorot, Y. Bengio, and Y. N. Dauphin. Large-scale learning of embeddings with reconstruction sampling. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 945–952, 201.

[6] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the Eighth IEEE International Conference on Data Mining*, pages 263–272, 2008.

[7] S. Kabbur, X. Ning, and G. Karypis. FISM: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 659–667. ACM, 2013.

[8] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 426–434. ACM, 2008.

[9] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[10] S. Lauly, H. Larochelle, M. Khapra, B. Ravindran, V. C. Raykar, and A. Saha. An autoencoder approach to learning bilingual word representations. In *Advances in Neural Information Processing Systems*, pages 1853–1861, 2014.

[11] J. Lee, S. Bengio, S. Kim, G. Lebanon, and Y. Singer. Local collaborative ranking. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 85–96. ACM, 2014.

[12] X. Ning and G. Karypis. SLIM: Sparse linear methods for top-N recommender systems. In *Proceedings of the 11th IEEE International Conference on Data Mining*, pages 497–506, 2011.

[13] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *Proceedings of the Eighth IEEE International Conference on Data Mining*, pages 502–511, 2008.

[14] S. Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.

[15] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 452–461, 2009.

[16] J. D. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM, 2005.

[17] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2007.

[18] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

[19] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[20] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web Companion*, WWW'15 Companion, pages 111–112, 2015.

[21] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 139–146. ACM, 2012.

[22] Y. Shi, M. Larson, and A. Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 269–272. ACM, 2010.

[23] H. Steck. Training and testing of recommender systems on data missing not at random. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 713–722. ACM, 2010.

[24] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[25] 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. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

[26] C. Wang and D. M. Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 448–456. ACM, 2011.

[27] H. Wang, N. Wang, and D.-Y. Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1235–1244. ACM, 2015.

[28] M. Weimer, A. Karatzoglou, Q. V. Le, and A. J. Smola. COFI RANK-maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems*, pages 1593–1600, 2007.

[29] J. Weston, S. Bengio, and N. Usunier. WSABIE: scaling up to large vocabulary image annotation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 2764–2770, 2011.

[30] J. Weston, C. Wang, R. Weiss, and A. Berenzweig. Latent collaborative retrieval. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 9–16, 2012.

[31] S.-H. Yang, B. Long, A. J. Smola, H. Zha, and Z. Zheng. Collaborative competitive filtering: learning recommender using context of user choice. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 295–304. ACM, 2011.

[32] L. Zhang, D. Agarwal, and B.-C. Chen. Generalizing matrix factorization through flexible regression priors. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 13–20. ACM, 2011.