Learning the Structure of Auto-Encoding Recommenders

Farhan Khawar
The Hong Kong University of Science and Technology
fkhawar@connect.ust.hk

Leonard Poon
The Education University of Hong Kong
kmpoon@eduhk.hk

Nevin L. Zhang
The Hong Kong University of Science and Technology
lzhang@cse.ust.hk

ABSTRACT

Autoencoder recommenders have recently shown state-of-the-art performance in the recommendation task due to their ability to model non-linear item relationships effectively. However, existing autoencoder recommenders use fully-connected neural network layers and do not employ structure learning. This can lead to inefficient training, especially when the data is sparse as commonly found in collaborative filtering. The aforementioned results in lower generalization ability and reduced performance. In this paper, we introduce structure learning for autoencoder recommenders by taking advantage of the inherent item groups present in the collaborative filtering domain. Due to the nature of items in general, we know that certain items are more related to each other than to other items. Based on this, we propose a method that first learns groups of related items and then uses this information to determine the connectivity structure of an auto-encoding neural network. This results in a network that is sparsely connected. This sparse structure can be viewed as a prior that guides the network training. Empirically we demonstrate that the proposed structure learning enables the autoencoder to converge to a local optimum with a much smaller spectral norm and generalization error bound than the fully-connected network. The resultant sparse network considerably outperforms the state-of-the-art methods like Mult-vae/Mult-dae on multiple benchmarked datasets even when the same number of parameters and flops are used. It also has a better cold-start performance.

KEYWORDS

Structure Learning, Collaborative Filtering, Sparse Autoencoder, Wide Autoencoder, Shallow Networks.

ACM Reference Format:
Farhan Khawar, Leonard Poon, and Nevin L. Zhang. 2020. Learning the Structure of Auto-Encoding Recommenders. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3366423.3380135

1 INTRODUCTION

Collaborative filtering (CF) uses the past behavior of users to recommend items to users[10, 11, 14, 17]. This past behavior is given in the form of a user-item matrix, where each row represents a user and each element in the row indicates whether the user has consumed[1] the corresponding item or not.

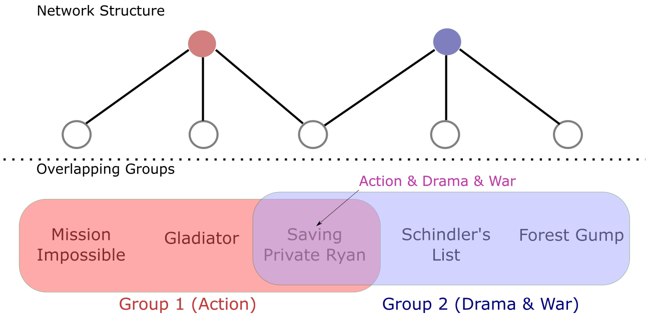

Figure 1: Sample clusters from the Movielens20M dataset are shown. Items are related to each other. Each item can be related to more than one group of items. Neural networks are built based on the overlapping groups of items.

Recently, autoencoder recommenders [16, 27] have been introduced to learn the user/item representations in a non-linear fashion and have shown to outperform conventional methods [16, 27]. These existing methods use neural networks with fully-connected (FC) layers where each neuron of the previous layer is connected to all the neurons of the next layer. An FC network structure is general purpose and carries a high modeling capacity, therefore it is a good first choice for application in any domain. However, more often than not, each domain has a certain structure that can be exploited to design neural network architectures that are less general but nevertheless more suitable for that particular domain. For example, convolution neural networks exploit the spatial invariance property of images to reduce the network size and increase the performance compared to the FC network. Similarly, recurrent neural networks are another example that exploits the sequential nature of the text data. Surprisingly, existing autoencoder recommenders (or even other neural network recommenders) use FC networks and have not explored structure learning in general and the use of domain-specific information to learn the structure of the neural network in particular.

The data in the CF domain has an inherent structure, that is, certain items tend to be more related to each other than to other items. As an example, consider the groups of items from the Movielens20M data shown in Figure 1. We can see that groups of thematically similar items exist. Moreover, we also note that an item can belong to multiple groups, for example, the movie Saving Private Ryan is both an action movie and a drama & war movie, so it is similar to movies in both groups.

[1] Consumed may refer to bought, viewed, clicked, liked or rated, etc.

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380135

With this premise, the attempt of the FC layer to use each neuron to model the interactions of *all* items can lead to modelling unnecessary/noisy item interactions which can harm the generalization ability of the autoencoder. This phenomenon of learning unnecessary parameters can be more detrimental when the data is sparse. Motivated by this, we propose to use structure learning techniques to determine the connections between the layers of the autoencoder recommender.

Common structure learning techniques like pruning connections with small weights [5] and using ℓ_1-regularization on neuron activation [9] are a natural choice and can easily be adapted for recommendation. However, they first start with an overcomplete FC network and then contract the network by retaining useful connections. While this approach is promising as it will be better than asking each neuron to model all item interactions, it does not take advantage of the prior information available to us regarding the tendency of items to be related to each other. Also, by staring with an FC network these approaches rely on the neural network's ability to decide during training which connections are unnecessary. Instead, a better approach would be to use a disjoint two-stage method, where we first fix the network structure and remove the "unnecessary" connections and then train only these connections. This can be thought of as pointing the auto-encoder in the right direction before beginning the training. Thus, in a disjoint two-stage method, the first stage acts as prior that incorporates our domain knowledge about item groups and fixes the structure and the second stage trains only these connections of the network. This enables the network to train efficiently and have both a higher validation accuracy and a higher test accuracy compared to the alternatives.

In this paper, we propose a simple two-stage scheme to incorporate this domain knowledge into the autoencoder structure by presenting a structure learning method that determines the connectivity structure of the autoencoder based on item groups. Given the fact that some items tend to be related to each other and many items tend not to be related to each other, we can introduce a neuron for each group of related items. These neurons model the interactions between the related items and, unlike the FC networks, do not model the unnecessary interactions between unrelated items. We first find *overlapping* groups of related items from the data and for each group, we introduce a neuron to model their interactions. The same structure is inverted and used for the decoder. Then, in the second step, we train this autoencoder to learn the weights of these connections.

Building a network structure in such a fashion allows the connectivity to be sparse as each item is only connected to a subset of neurons of the next layer. As a result, the number of parameters is reduced and the model has a smaller memory footprint than the corresponding FC networks. Since our structure is sparse we can make it wider compared to the FC network in the same memory footprint. Having a wide structure allows us to use more neurons to model the interactions of items. As a result, each neuron models the interaction of a few closely related items. We name the final structure as a sparse and wide (Sw) network.

We demonstrate the benefit of the Sw structure by using a *denoising autoencoder* (Dae) [26]. FC and non-wide Daes have been used successfully in the past for the recommendation task[16, 27], and have shown state-of-the-art performance[16]. By utilizing the item group structure of the data to make the network sparse and wide, the Sw-dae is able to considerably outperform existing methods on several benchmarked datasets. We also show that for the recommendation task, Sw-dae outperforms other common methods of neural network structure learning. Also, it exhibits superior performance compared to the state-of-the-art baseline in the cold-start scenario.

The main contributions of this paper are:

- We introduce the idea of structure learning for the recommendation task and show that by incorporating existing structure learning techniques we can outperform the state-of-the-art deep learning recommenders.
- We then present a simple two-stage technique of learning the autoencoder connectivity structure based on item groups. Which, as we show, is better for the recommendation task than existing structure learning techniques.
- We demonstrate that this performance gain is due to the lower spectral norm of the weight matrices and hence a lower generalization error bound.
- Via empirical evaluation, we show that Sw-dae exhibits the state-of-the-art performance even when it uses the same number of parameters/flops as the best baseline. Moreover, it is also superior in cold-start performance.

2 RELATED WORK

De-noising Autoencoders. Autoencoders can be seen as a special case of feed-forward neural networks that attempt to recreate the input at their output. This is done by learning the hidden encoding of the input and using this encoding to recreate the input at the output. De-noising autoencoders [26] are a type of autoencoders that receive the input data in a *noisy* form and attempt to recreate the original *clean* data at their output.

Autoencoders and Neural Networks for Recommendation. Autoencoders have been used in the past for recommendations. Two such methods that have received attention are Mult-vae [16] and Cdae [27]. In [27] the authors extend the standard denoising autoencoder by using a specific neuron to model each user, whereas in [16] the authors introduce variational autoencoders (VAE) with the multinomial likelihood (Mult-vae) and a partially regularized objective function for recommendation. The Mult-vae represents the state-of-the-art performance on large scale real-world datasets.

Another popular methods that extends linear factor methods for recommenders is neural collaborative filtering (Ncf) [6]. Ncf introduces non-linear interactions between user and item latent vectors via a neural network. The number of parameters in Ncf increases linearly with the number of users and items and this can lead to degraded performance on larger datasets common in CF.

Unlike Sw-dae, all the aforementioned neural network based methods use fully-connected networks that do not incorporate the item group structure in their networks.

Structure Learning and Sparsity. Contraction approaches have been proposed to learn the structure and introduce sparsity in the neural network structure. They start by a larger than required network and then either prune the connections/neurons or introduce a penalty that forces the network to be sparsely activated. In [5]

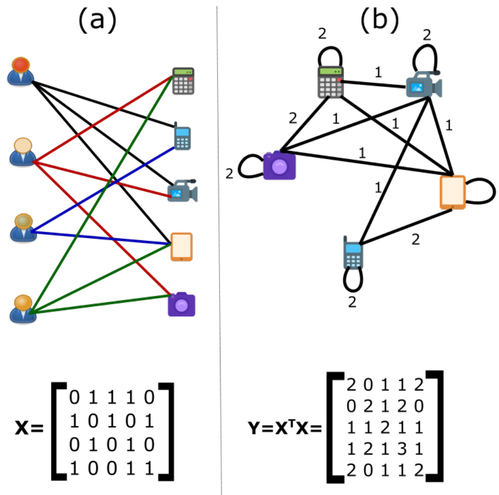

Figure 2: A toy user-item bipartite graph and its corresponding user-item input matrix X are shown in (a). It contains 4 users and 5 items. The corresponding item-item graph and its adjacency matrix Y are shown in (b).

a popular method of pruning is presented that prunes all the network connections with weights less than a predefined threshold. For making the de-noising autoencoder sparsely activated, in [9] an ℓ_1 penalty is introduced on the activation of the hidden layer. Unlike Sw-DAE, both these approaches of introducing learning the connectivity structure start with a complex model as the input and do not explicitly model the cluster structure.

Clustering. Clustering has been traditionally used for recommendations [25]. User and item-based CF methods have high computational and space complexity. Clustering algorithms like K-means and hierarchical clustering have been applied to CF to improve the computational efficiency of these methods [1, 2]. But such methods generally fail to capture global patterns in data and trade accuracy for scalability [2, 23, 24]. Other CF methods have also relied on clustering to aid in recommendation by clustering users/items based on the side information commonly found in heterogeneous networks [29]. Yet other methods like [7, 12, 28] have used co-clustering to get user/item latent factors. However, none of the above methods have used the item cluster information as a structural prior for the neural network and all of these methods exhibit inferior performance compared to the state-of-the-art deep-learning based methods.

3 LEARNING THE SPARSE STRUCTURE

The proposed method for learning the connectivity structure between two layers has the following steps: (i) group the input variables (items) into overlapping clusters, (ii) for each overlapping group introduce a neuron, (iii) learn the parameters of these neurons. We now describe each part of the structure learning procedure in detail.

3.1 Getting the Overlapping Item Groups

In this section, we consider the problem of partitioning the input variables into K overlapping groups. This involves two steps: (1) getting item embeddings and (2) grouping the items based on these embeddings.

3.1.1 Getting the Item Embeddings. We would like to obtain item embeddings that (a) preserve the relationships between items and (b) discount for the effect of popular items. To achieve this we first use the user-item feedback matrix to obtain an item-item graph, then construct a graph laplacian that embodies our requirements, and finally get a low-dimensional representation of this laplacian to get our low-dimensional item embeddings.

Let $\mathbf{X}$ be the $n \times m$ input matrix with n users and m items. We can view this matrix as a user-item bipartite graph, as shown in Figure 2(a), where each edge denotes that a user consumed an item. Since this is a bipartite graph, the items are related to each other via users. To get the item embeddings we first transform this into an item-item graph and then get the embedding of each node. This new item-item graph G (shown in Figure 2(b)) can be easily obtained from the input matrix as its adjacency matrix is defined as $\mathrm{Y} = \mathbf{X}^T\mathbf{X}$.

Each edge in G now denotes the existence of a co-occurrence relationship between items. In addition, G is a weighted item-item graph where the weight of the edge denotes the strength of co-occurrence between items. This weight is equal to the number of times the two items were co-consumed by the users. From Figure 2(b) we can see that the item-item graph G involves fewer nodes (items only) and the corresponding adjacency matrix representation is less sparse than the original bipartite matrix $\mathbf{X}$. These observations are true in general and are the reason we chose to transform the input graph to G.

Given the item-item graph G we would like to obtain node embeddings that (a) preserve the relationships between nodes in graph i.e., node embeddings should be influenced more by neighboring nodes with large edge weights and (b) account for the variation of node degrees in the graph i.e., a popular item will have a high co-occurrence with most items and may influence their neighbors more. Both these requirements can be fulfilled if we consider $\mathbf{X}^T\mathbf{X}$ as the adjacency matrix of G and form the graph laplacian as:

$$\Delta = D^{-1/2}\mathbf{X}^T\mathbf{X}D^{-1/2} = D^{-1/2}\mathrm{Y}D^{-1/2}, \tag{1}$$

where, D is the diagonal degree matrix and $D_{ii} = \deg(v_i) = \sum_j \mathrm{Y}_i$. Y_i is the $i-th$ row of Y and $\deg(v_i)$ is the degree of node i.

The elements of Δ are given by:

$$\Delta_{ij} := \begin{cases} \frac{|\mathbf{x}_i|}{\deg(v_i)} & \text{if } i = j \text{ and } \deg(v_i) \neq 0 \\ \frac{\mathbf{x}_i.\mathbf{x}_j}{\sqrt{\deg(v_i)\deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Where, $\mathbf{x}_i$ denotes the $i-th$ column of $\mathbf{X}$.

To see how this laplacian fulfills our requirements, consider each row of the laplacian which corresponds to an item:

$$\Delta_i = \sum_{j:(i,j)\in\mathcal{E}} \frac{\mathbf{x}_i.\mathbf{x}_j}{\sqrt{\deg(v_i)\deg(v_j)}}, \tag{3}$$

where, $\mathcal{E}$ is the edge set of G. We see that the relationship between item i and j is proportional to their co-occurrence but inversely proportional to their popularity (node degrees). Thus, if we consider Δ_i to be an embedding of item i then both our requirements (a) and (b) are fulfilled.

However, Δ_i is an m-dimensional vector which can be large and as such we are interested in a low-dimensional embedding. To get this low-dimensional vector we perform the eigenvalue decomposition of Δ and pick the top F eigenvectors. Briefly, the reason for picking the eigenvectors is that the eigenvectors of Δ can be seen as a smooth basis of the graph G (i.e., corresponding to minimum Dirichlet energy) and smoothness in the frequency domain corresponds to spatial localization in the graph[22, 30]. Thus, each eigenvector can be seen as partitioning G. We refer the reader to [22, 30] for more details on the relationship between smoothness and spatial localization.

Having found the Laplacian Δ, our task has now reduced to finding the eigenvectors of Δ. First, the eigendecomposition of $\Delta = \mathbf{V}\Lambda\mathbf{V}^T$ is performed, where the columns of $\mathbf{V}$ are the eigenvectors and Λ is a diagonal matrix that contains the corresponding eigenvalues. Second, the largest F ($F \ll m$) eigenvalues retained, since we want a low dimensional embedding, where F is a user-provided parameter. Denote this matrix by $\mathbf{V}_F$. The rows of $\mathbf{V}_F$ represent the item embeddings in this low-dimensional space.

3.1.2 *Grouping.*

After getting the item embeddings we turn our attention to grouping the items to obtain overlapping groups. We first start by clustering the items based on their embeddings using K-means[2]. This gives us K cluster centroids. Since each item can belong to more than one item clusters we connect each item to its R ($R < K$) nearest centroids. This results in a simple and scalable approach to obtain the K overlapping clusters.

The complete procedure is given in Algorithm 1. First compute Δ using Equation 2 and compute the top F left eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_F$ of Δ with the largest eigenvalues. Second, stack them column-wise to form the matrix $\mathbf{V}_F$. Third, normalize each row of the matrix $\mathbf{V}_F$ so that the sum is 1. Following [20] this is done as a preprocessing step for clustering. Fourth, regard the rows of $\mathbf{V}_F$ as points in the F-dimensional Euclidean space and use the K-means algorithm to get K cluster centroids. Note that each row of $\mathbf{V}_F$ corresponds to an item. In practice an item can belong to multiple clusters, therefore in step five, we connect each item to R ($R < K$) nearest centroids. A partition of the items into K overlapping clusters is therefore obtained.

Algorithm 1 ItemGrouping($\mathbf{X}$, K, R, F)

```
Inputs: $\mathbf{X}$ — an $n \times m$ user-item matrix, $K$ — number of clusters,
    $R$ — degree of overlap, $F$ — number of singular vectors.
Outputs: $K$ overlapping item clusters.
1: Compute the $m \times m$ Laplacian matrix of $G$ using Equation 2.
2: Find the $F$ largest eigenvectors of $\Delta$ via Lanczos Bidiagonalization: $\{\mathbf{v}_1 \ldots \mathbf{v}_F\} = \mathrm{evd}(\Delta, F)$.
3: Stack the eigenvectors column wise to form the matrix $\mathbf{V}_F = [\mathbf{v}_1 \ldots \mathbf{v}_F] \in \mathbb{R}^{m \times F}$.
4: Normalize $\mathbf{V}_F$ row-wise such that each row sums to one.
5: Run K-means on the rows of $\mathbf{V}_F$ (i.e. the items) to get $K$ cluster centroids.
6: Associate each item with $R$ nearest centroids to get $K$ overlapping item clusters.
```

3.1.3 *Discussions.*

We compute the eigenvalue decomposition of Δ with the Lanczos bidiagonalization algorithm [3]. The Lanczos algorithm is fast for sparse matrices and its complexity is generally $O(n_{nz}F)$ [15], where n_{nz} are the number of non-zero entries of Δ. Since our laplacian is based on the co-occurrence matrix $\mathbf{Y}$, which is inherently sparse, getting the top F eigenvectors is quite fast. This sparsity is the key to its scalability. Also, since each item is now an F-dimensional vector, the complexity of K-means would be $O(mlKF)$, where l is the maximum number of iterations of K-means, and this is linear in the input size. We present more details on the scalability of Algorithm 1 in the section 5.7.

Instead of forming the laplacian, we can get the eigenvectors of the co-occurrence matrix and project the items in the low dimensional space spanned by the eigenvectors. Since we don't form the laplacian, we can get the eigenvectors of $\mathbf{Y}$ directly from $\mathbf{X}$. This is because $\mathbf{X} = U\Sigma V^T$, and $\mathbf{Y} = \mathbf{X}^T\mathbf{X} = V\Sigma U^T U\Sigma V^T = V\Sigma^2 V^T$. Therefore, by performing the singular value decomposition of $\mathbf{X}$ (using Lanczos bidiagonalization algorithm) we are able to get the eigenvectors of $\mathbf{Y}$. We can then follow a similar procedure to before, but now we operate on $\mathbf{X}^T$ instead of the laplacian, perform its singular-value decomposition, and project the items in the low dimensional space and cluster them. The complete procedure is given in Algorithm 2. Unlike the laplacian, this method will have the disadvantage of allowing higher degree nodes to dominate more. However, since we operate on a more sparse matrix $\mathbf{X}$, getting the top eigenvectors will be faster.

Algorithm 2 ItemGrouping2($\mathbf{X}^T$, K, R, F)

```
Inputs: $\mathbf{X}^T$ — an $m \times n$ item by user implicit feedback matrix,
    $K$ — number of clusters, $R$ — degree of overlap, $F$ — number
    of singular vectors.
Outputs: $K$ overlapping item clusters.
1: Find the $F$ largest left singular vectors and associated singular values of $\mathbf{X}^T$ via Lanczos Bidiagonalization: $\{\mathbf{u}_1 \ldots \mathbf{u}_F, \sigma_1 \ldots \sigma_F\} = \mathrm{svd}(\mathbf{X}^T, F)$.
2: Stack the singular vectors column wise to form the matrix $\mathbf{U}_F = [\mathbf{u}_1 \ldots \mathbf{u}_F] \in \mathbb{R}^{m \times F}$ and form the diagonal matrix $\Sigma_F$.
3: Project the items in this space by $\mathbf{P} = \mathbf{U}_F \Sigma_F$.
4: Normalize $\mathbf{P}$ row-wise such that each row sums to one.
5: Run K-means on the rows of $\mathbf{P}$ (i.e. the items) to get $K$ cluster centroids.
6: Associate each item with $R$ nearest centroids to get $K$ overlapping item clusters.
```

We note that Algorithm 1 is similar to spectral clustering[3] where it tries to embed items in a smooth space and Algorithm 2 is like principal components analysis where the items are projected (line 3) in the space spanned by the top eigenvectors of the co-occurrence matrix[4]. We compare the performance of these methods in section 5.7.

Another alternate procedure to get the overlapping clusters would be to cluster the columns of $\mathbf{X}$. However, $\mathbf{X}$ does not possess the desirable properties of Δ. Also, due to the inherent sparsity of $\mathbf{X}$, applying K-means directly on $\mathbf{X}$ might lead to unsatisfactory performance. Finally, if applied directly on $\mathbf{X}$, K-means will have a

[2] Any distance-based clustering can be used, we leave this exploration as future work.

[3] However, it is much faster due to operating on the sparse co-occurrence matrix.

[4] PCA uses the dense co-variance matrix with expensive eigendecomposition.

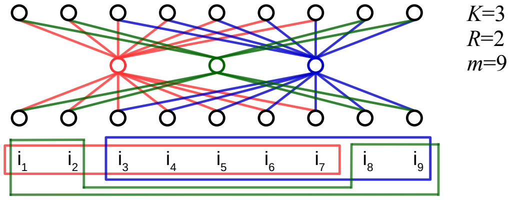

Figure 3: Illustration of network construction. Suppose the items can be grouped into three overlapping clusters $\{i_1, i_2, \ldots, i_7\}$, $\{i_3, i_4, \ldots, i_9\}$ and $\{i_1, i_2, i_8, i_9\}$. In the resulting structure, there is one neuron for each cluster (colored nodes). Each neuron is connected to the input and output variables for all the items belonging to its cluster.

complexity of $O(mnKl)$. When m and n are both large this becomes computationally infeasible.

3.2 Building the Connectivity Structure

To determine the connectivity structure between two neural network layers we use the overlapping clusters of the items. Items in one cluster are more related to each other compared to items that are not in the same cluster. This is because each cluster represents items that are close to each other in the low-dimensional subspace. Therefore, we introduce a latent node (neuron) to model the interactions between them. This neuron connects to all the items in the overlapping cluster. Figure 3 shows an illustration of the network structure.

By forming the structure in such a manner we ensure that (i) the connectivity is sparse as $R < K$, (ii) the interactions between related items are captured, and (iii) the aspect of an item being related to multiple item groups is also modeled. The intuition of forming such a connectivity structure is that an item is related to a few concepts (represented by neurons of the hidden layer) rather than to all concepts.

3.3 Learning Parameters

Once the connectivity structure is learned we can use it to replace the FC layers in the denoising autoencoder. The denoising autoencoder consists of an input layer, a hidden layer, and an output layer. We use our learned structure to replace the FC connections between the input and the hidden layer, and the same connectivity structure is inverted to replace the connections between the hidden and the output layer. This results in a sparse and wide denoising autoencoder (Sw-dae).

Let $\mathbf{x} = [x_i], i = 1 \ldots m$ denote the vector of input units, $\mathbf{h} = [h_j], j = 1 \ldots K$ denote the vector of the hidden units, $\mathbf{b} = [b_j]$ denote the bias vector of the hidden units, $\mathbf{W} = [w_{ij}]$ denote the weights of the connectivity structure such that $w_{ij} = 0$ if input variable x_i is not connected to the hidden variable h_j and $\mathbf{W}^T$ denoting the transposition of $\mathbf{W}$. The Sw-dae then defines the probability distribution $p(\mathbf{h}|\mathbf{x})$ of the hidden units given the observed variables and the probability distribution $p(\mathbf{x}|\mathbf{h})$ of the observed variables given the hidden units as follows:

$$\begin{aligned} p_{encoder}(\mathbf{h}|\mathbf{x}) &= f(x) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}), \\ p_{decoder}(\mathbf{x}|\mathbf{h}) &= g(h) = \sigma(\mathbf{W}'\mathbf{h} + \mathbf{b}'), \end{aligned} \tag{4}$$

Table 1: Statistics of the datasets.

	ML20M	Netflix	MSD
# of users	136,677	463,435	571,355
# of items	20,108	17,769	41,140
# of interactions	10.0M	56.9M	33.6M
% of interactions	0.36%	0.69%	0.14%
# of val./test users	10,000	40,000	50,000

where, σ represents the sigmoid function and $\mathbf{W}'$ represents the decoder weight matrix which has the same connectivity structure as $\mathbf{W}^T$.

To learn the Sw-dae parameters i.e., the weights of the connections and biases of the neurons, we use the stochastic gradient descent with the denoising criterion [26]. Let $C(\tilde{\mathbf{x}}|\mathbf{x})$ be a random corruption process, which is a conditional distribution of the noisy sample $\tilde{\mathbf{x}}$ given the original data sample $\mathbf{x}$. Given the noisy sample $\tilde{\mathbf{x}}$ from $C(\tilde{\mathbf{x}}|\mathbf{x})$ as the input, the job of the Sw-dae is to learn the function to reconstruct the original data sample $\mathbf{x}$ i.e., learn the reconstruction distribution $p(\mathbf{x}|\tilde{\mathbf{x}}) = p_{decoder}(\mathbf{x}|\mathbf{h} = f(\tilde{\mathbf{x}}))$. This can be achieved by performing stochastic gradient descent on the negative log-likelihood $-\log p_{decoder}(\mathbf{x}|\mathbf{h} = f(\tilde{\mathbf{x}}))$. This is equivalent to performing stochastic gradient descent to minimize the following expected loss [4]:

$$J = -\mathbb{E}_{x \sim \hat{p}_{data}(\mathbf{x})} \mathbb{E}_{\tilde{\mathbf{x}} \sim C(\tilde{\mathbf{x}}|\mathbf{x})} \log p_{decoder}(\mathbf{x}|\mathbf{h} = f(\tilde{\mathbf{x}})), \tag{5}$$

where $\hat{p}_{data}$ is the empirical probability distribution of the data.

3.4 Going Deeper

The proposed method learns a single hidden layer autoencoder. In our experiments, this architecture gave the best results. However, for completeness, we also present how our structure learning technique can be used to go deeper. We propose a greedy layer-wise fashion to build a stacked Sw-dae structure and to get its pretrained weights. This is achieved by repeating the procedures of clustering the observed variables, building the connectivity structure and learning parameters until the desired depth is reached.

The first step is to turn the hidden variables (neurons) to observed variables. This can be done by doing a forward pass for each data point $\mathbf{x}$, getting its hidden encoding and treating this as the new input vector. This results in the new data of dimension $n \times K$, where now we have K observed variables. Then, in the second step, we perform the input variable clustering of Algorithm 1, followed by building the structure (section 3.2) and learning the parameters (section 3.3). This results in a new Sw-dae, which in the third step is stacked with the previous Sw-dae. We can repeat these three steps until the desired depth is reached.

The Sw-daes can be stacked by connecting the output of the encoder/decoder of each Sw-dae to the input of the successive Sw-dae's encoder/decoder and copying the respective parameters. These copied parameters serve as pretrained weights. Formally, consider a stacked Sw-dae with d hidden layers in its encoder/decoder, then the encoding $f(\tilde{\mathbf{x}})$ of the stacked Sw-dae is obtained by the successive encoding of each encoding layer $l = 1 \ldots d$ as follows:

$$\begin{aligned} \mathbf{z}^l &= \mathbf{W}^{l-1}\mathbf{a}^{l-1} + \mathbf{b}^{l-1}, \\ \mathbf{a}^l &= \sigma(\mathbf{z}^l), \end{aligned} \tag{6}$$

where $\mathbf{W}^l$, $\mathbf{a}^l$ and $\mathbf{b}^l$ denote the weight, activation and bias of the l-th layer respectively and $\mathbf{a}^0 = \tilde{\mathbf{x}}$. The decoding $g(\mathbf{h})$ is also obtained by the successive decoding of the decoding layers but in the reverse order as follows:

$$\begin{aligned} \mathbf{z}^{d+l+1} &= \mathbf{W}^{d-l'}\mathbf{a}^{d+l} + \mathbf{b}^{d+l}, \\ \mathbf{a}^{d+l} &= \sigma(\mathbf{z}^{d+l}). \end{aligned} \tag{7}$$

We note that $\mathbf{a}^d = f(\tilde{\mathbf{x}}) = \mathbf{h}$ and $\mathbf{a}^{2d} = g(\mathbf{h}) = p_{decoder}(\mathbf{x}|\mathbf{h} = f(\tilde{\mathbf{x}})$. Then the stacked Sw-dae can be trained using the objective of Equation 5 to fine tune the parameters.

4 EMPIRICAL STUDY

We perform an empirical evaluation of Sw-dae for the recommendation task. To perform the recommendation for a user, we input the user's binary consumption history (corresponding row of matrix $\mathbf{X}$) to the trained Sw-dae. Then we perform forward pass through the Sw-dae to get the output at the decoder before the softmax. The values in the output vector are treated as the score. We then rank the unconsumed items based on this score.

4.1 Datasets

For a direct performance comparison, we use the same datasets, data splits (using the same random seed) and pre-processing steps as [16] during our evaluation. Details of the three datasets are below:

- Movielens20M (ML20M): is a dataset of users rating movies. The ratings were binarized and a rating value greater or equal to four was interpreted as a positive signal. Users who rated less than five movies were filtered out.
- Netflix: is also a movie rating dataset from the Netflix prize[5]. The ratings were binarized and the users with less than five movie ratings were removed and a rating values greater or equal to four was taken as positive.
- Million Song Dataset (MSD) [18]: is a dataset that contains the playing counts of users for songs. The counts were binarized and a user listening to a song was taken as a positive signal. Users who listened to less than twenty songs or songs that were listened to by less than two hundred users were removed.

We use the strong generalization experimental setup [16, 17] in our experiments. The datasets were split into training, validation and test users resulting in three subsets. The details of the splits are shown in Table 1. The models were trained on the training users. A randomly chosen 80% subset of the click history of the validation/test users was used to learn their necessary representations and the remaining 20% of the click history was used for evaluation.

4.2 Metrics

To get the ranked list of the unconsumed items of the validation/test users from Sw-dae, we ranked the items based on the un-normalized probability score at the decoder. We then used two top R ranking based metrics, namely, Recall@R and the normalized discounted cumulative gain NDCG@R for evaluation. Formally, Recall@R for a user u is defined as:

$$\text{Recall@}R(u, \rho) := \frac{\sum_{r=1}^{R} \mathbb{I}[\rho(r) \in I_u]}{\min(R, |I_u|)},$$

where ρ is the ranked list, $\rho(r)$ is the item at position r in the list, $\mathbb{I}$ is the indicator function, and I_u is the set of items consumed by user u. The term is the denominator ensures that Recall@R has a maximum value of 1 which corresponds to raking all relevant items of the user in the top R list.

Table 2: The number of flops and parameters in millions. Even with the same number of parameters/flops Sw-dae-p provides considerable improvement in performance.

ML20M	Parameters (M)	Flops (M)	NDCG@100	% Improvement
Sw-dae	60.324	120.645	0.442	3.76
Sw-dae-p	24.129	48.258	0.437	2.58
Mult-vae	24.193	48.738	0.426	-
Netflix				
Sw-dae	46.199	92.3962	0.404	4.66
Sw-dae-p	21.322	42.644	0.398	3.11
Mult-vae	21.386	43.124	0.386	-
MSD				
Sw-dae	74.052	148.102	0.372	17.72
Sw-dae-p	49.368	98.734	0.367	16.77
Mult-vae	49.432	99.214	0.316	-

The NDCG@R is the DCG@R divided by the best possible DCG@R. The best DCG@R corresponds to ranking all the relevant items of the user at the head of the top R list. Formally, DCG@R is defined as:

$$\text{DCG@}R(u, \rho) := \sum_{r=1}^{R} \frac{\mathbb{I}[\rho(r) \in I_u]}{\log(r+1)}.$$

We note that unlike Recall@R which gives equal importance to relevant items in the ranked list, NDCG@R gives more importance to correctly ranked items at the head of the list than those lower down the list.

4.3 Experimental Setup

The hyperparameters and architecture were chosen based on the NDCG@100 performance on the validation set. The architecture of Sw-dae was symmetric for both the encoder and decoder. We used the sigmoid activation function as the non-linear activation function. For the corruption process, we applied a random dropout with a probability of 0.6 at the input. During training, we also applied a dropout with a probability of 0.2 at the hidden layer. We used a batch size of 500 and trained using the Adam optimizer [13] with a learning rate of 0.001 for 100 epochs. K was searched in multiples of 1000 up to the maximum value determined by the GPU memory for each dataset. For all our experiments, we set F at 50 and set R to keep the network at 10% sparsity (i.e., $R = 0.1K$) compared to the fully connected network.

4.4 Baselines

The following non-linear and linear state-of-the-art collaborative filtering methods were used as baselines in our experiments:

Mult-vae [16]: is a non-linear recommender that uses a VAE with a partially regularized objective function and uses a multinomial loss function. The hyperparameters including the regularization parameter β were set using the strategy of the original paper

[5] http://www.netflixprize.com/

[16] that gave its best NDCG@100 performance. The Adam optimizer was used with a batch size of 500. Mult-dae also uses the multinomial likelihood but uses a DAE instead of a VAE.

Wmf [8]: is a linear low-rank matrix factorization model. The weight of each consumption event was searched over the set $\{2, 5, 10, 30, 50, 100\}$ and the size of latent dimension K was searched over $\{100, 200\}$ based on the NDCG@100 performance on validation.

Slim [21]: is also a linear item based recommender which solves an ℓ_1-regularized constrained optimization problem to learn a sparse item-item similarity matrix. The regularization parameters were searched over $\{0.1, 0.5, 1, 5\}$ based on the NDCG@100 performance on validation users.

Cdae [27]: enhances an autoencoder by adding a latent factor for each user. Unlike the DAE, this results in the number of parameters growing linearly with the number of users and items and makes it prone to overfitting. The settings of [16] were used to train the model. The weighted square loss and Adam with a batch size of 500 was used for optimization.

We do not report results on Ncf since its performance was not competitive on the larger datasets used in this paper. This is consistent with the findings of [16].

5 RESULTS

In this section, we provide the quantitative results for our proposed approach compared with the baselines. We will look at the performance in order to answer the following:

- How do autoencoders with learned structure compare with baselines in terms of recommendation accuracy?
- Is the Sw-dae gain due to more parameters or more neurons?
- Does the item group structure in the autoencoder help?
- How does the proposed structure learning approach compare with other structure learning techniques?

5.1 Learned Structures Compared to Baselines

In Table 3 we show the performance in terms of NDCG@100, Recall@20 and Recall@50. For each dataset, the structure learning methods are above the solid line and the baselines are below it. Fc-prune and Fc-Reg represent the use of pruning [5] and regularization [9] to learn the connectivity structure of the autoencoder respectively. We see that the standard structure learning methods outperform the baselines except for Recall@50 on ML20M. However, we see that the Sw-dae outperforms both the linear and non-linear baselines on all the datasets by a considerable margin. The largest performance improvement is observed on the sparsest MSD dataset. We conjecture that the addition of the cluster structure assists the network training when there are lower number of observations.

5.2 Sw-dae with the Same Number of Parameters

The results of the baseline neural network methods reported in Table 3 are based on the architecture with one or two hidden layers of 600 or 200 neurons since they gave the best validation NDCG@100 performance [16]. These architectures are FC and not as wide as Sw-dae, therefore, we also report the results of our method when the same number of parameters as the best baseline i.e., Mult-vae are used. We label this variant as Sw-dae-p and we can see that even with the same number of parameters Sw-dae-p outperforms the baselines. In Table 2 we show the comparison in terms of flops and parameters along with the percentage improvement in NDCG@100 of Sw-dae/Sw-dae-p over Mult-vae. We see that Sw-dae is wider and has more parameters[6] and flops but gives the best performance. However, Sw-dae-p with the same number of parameters and flops as Mult-vae is still able to considerably outperform it on all datasets.

Table 3: Comparison between Sw-dae with various baselines on the test set. Sw-dae outperforms the baselines considerably on all datasets. Slim did not finish within a reasonable amount of time on MSD.

(a) ML-20M

	Recall@20	Recall@50	NDCG@100
Sw-dae	**0.410**	**0.549**	**0.442**
Sw-dae-p	0.406	0.542	0.437
Fc-Prune-50	0.399	0.534	0.431
Fc-reg	0.399	0.535	0.431
Mult-vae	0.395	0.537	0.426
Mult-dae	0.387	0.524	0.419
Wmf	0.360	0.498	0.386
Slim	0.370	0.495	0.401
Cdae	0.391	0.523	0.418

(b) Netflix

	Recall@20	Recall@50	NDCG@100
Sw-dae	**0.370**	**0.458**	**0.404**
Sw-dae-p	0.364	0.453	0.398
Fc-Prune-50	0.355	0.445	0.390
Fc-reg	0.355	0.444	0.389
Mult-vae	0.351	0.444	0.386
Mult-dae	0.344	0.438	0.380
Wmf	0.316	0.404	0.351
Slim	0.347	0.428	0.379
Cdae	0.343	0.428	0.376

(c) MSD

	Recall@20	Recall@50	NDCG@100
Sw-dae	**0.317**	**0.416**	**0.372**
Sw-dae-p	0.313	0.413	0.369
Fc-Prune-50	0.304	0.397	0.356
Fc-reg	0.300	0.393	0.352
Mult-vae	0.266	0.364	0.316
Mult-dae	0.266	0.363	0.313
Wmf	0.211	0.312	0.257
Slim	—	—	—
Cdae	0.188	0.283	0.237

5.3 Is the Gain Due to More Neurons?

We can also make the baseline structures wide and examine their performance in relation to Sw-dae. Figure 5 shows this comparison for the two best baselines Mult-dae/vae on the ML20M and MSD datasets. In addition, we also make a comparison with a Fc-dae that uses the same dropout, activation and loss functions as Sw-dae. All the methods used the same structure of one hidden layer and the width of the models was increased until the GPU memory capacity was reached.

[6] These parameters are just 10% of the parameters of the original FC network.

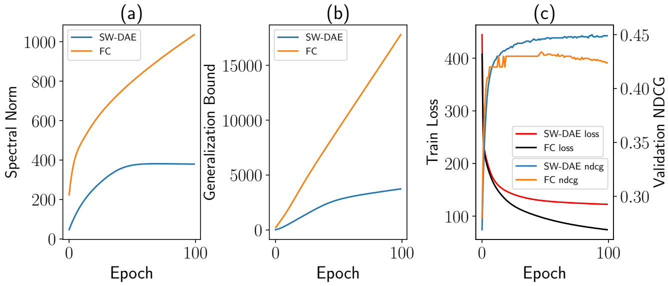

Figure 4: (a) The spectral norm of Sw-DAE and Fc-DAE, (b) the generalization error upper bound and (c) the training loss and validation NDCG for the ML20M dataset. Sw-DAE has a lower spectral norm and hence a lower generalization error bound. This manifests as a higher validation NDCG even when the training loss is more than Fc-DAE .

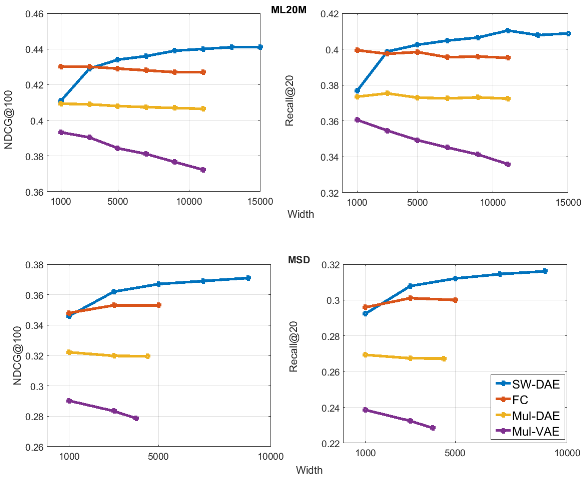

Figure 5: The performance of Sw-DAE improves with the width (K) of the hidden layer, unlike the baselines, due to better modeling of the item relationships. The performance gain of Sw-DAE is due to incorporating the item group information in its structure.

From Figure 5 we can see that, unlike the baselines, the NDCG@100 and Recall@20 performance of Sw-DAE increases with the number of neurons and finally plateaus. Increasing the number of neurons allows better and more fine-grained modeling of item interactions as each neuron models the interactions of a smaller group of related items.

We see that for Mult-DAE/VAE and Fc-DAE increasing the number of neurons does not help in performance. In fact the performance of narrower Mult-DAE/VAE shown in Table 3 was better than their wider counterparts shown in Figure 5.

5.4 Does the Item Group Structure Help?

This brings us to the utility of the sparse and wide cluster structure. To isolate the performance gain due to the cluster structure, we can compare the results of Sw-DAE and Fc-DAE in Figure 5. Since the corresponding Fc-DAE uses the same number of neurons, activation function, and parameter and learning settings as Sw-DAE, this improved performance can be attributed to better modeling of item relations as a result of the introduction of the item group structure.

5.5 Stable Rank and Spectral Norm

The item group structure removes unnecessary item interaction in the first phase by fixing the structure, and in the second phase Sw-DAE only learns useful parameters from the sparse data. Intuitively this can be the rationale for the improved generalization. To understand the generalization ability of Sw-DAE more formally, we investigate the spectral norm and stable rank of the weight matrix (either encoder or decoder) of the auto-encoder.

The stable rank srank($\mathbf{W}$) of a matrix $\mathbf{W}$ is defined as srank($\mathbf{W}$) := $||\mathbf{W}||_F^2/||\mathbf{W}||_2^2$. It is the ratio of its squared Frobenius norm and squared spectral norm. Clearly, for any matrix $\mathbf{W}$, we have $1 \leq$ srank($\mathbf{W}$) $\leq$ rank($\mathbf{W}$). The stable rank is more stable than the rank because it is largely unaffected by tiny singular values. More importantly, it has recently been shown [19, Theorem 1] that the generalization error of the neural network depends on the spectral norm and the stable rank of the weight matrices of the network. Specifically, it can be shown that the generalization error is upper bounded by $O\left(\sqrt{\prod_{j=1}^{L} ||\mathbf{W}_j||_2^2 \sum_{j=1}^{L} \mathrm{srank}(\mathbf{W}_j)/n}\right)$, where L is the number of layers in the network. This upper bound suggests that a smaller spectral norm (or smoother function mapping) and smaller stable rank lead to better generalization. We note that this upper bound depends on the product of the square of the spectral norms of all the layers, therefore, a smaller spectral norm is highly desirable.

We can compare the spectral norm and generalization bound of Sw-DAE with the corresponding Fc-DAE compare which is identical to Sw-DAE in every respect except the fact that Fc-DAE does not have the item group structure. In Figure 4 (a) we plot the spectral norm of weight matrix for the encoder for Sw-DAE and Fc-DAE[7] on the ML20M dataset. We see that Sw-DAE has a much lower spectral norm then Fc-DAE throughout the training. In Figure 4 (b) we plot the generalization error upper bound for both Sw-DAE and Fc-DAE and again we see that Sw-DAE has a much lower generalization error upper bound. We also note that, unlike Fc-DAE , by the end of the 100-th epoch the spectral norm and generalization error stabilize for Sw-DAE . Therefore, as expected, in Figure 4 (c) we see that as the training progresses Sw-DAE has a much better validation NDCG even though the training error is higher than Fc-DAE . Consequently, the effect of introducing the item group structure is the reduction of the spectral norm of network weight matrices and this results in better generalization.

5.6 Other Methods of Structure Learning

Pruning the network connections after training followed by retraining is one popular way of learning the connectivity structure. In Table 4 we show the behavior of the pruning method (Fc-PRUNE) on the ML20M dataset. We used the best FC model based on validation, pruned it following [5] and then retrained it. As we prune more connections, the network becomes sparser, however, its performance

[7] Similar results are observed for the decoder and on other datasets.

Table 4: The best FC architecture was chosen for pruning. The reported results are on the ML20M data. Prune-90 has the same sparsity level of 10% as Sw-dae but Sw-dae performs much better.

Method	NDCG@100	Recall@20	Recall@50
Sw-dae	**0.442**	**0.410**	**0.549**
Fc-Prune-90	0.415	0.385	0.521
Fc-Prune-80	0.424	0.392	0.531
Fc-Prune-70	0.427	0.395	0.531
Fc-Prune-60	0.431	0.399	0.534
Fc-Prune-50	0.430	0.399	0.535

Table 5: The best FC architecture on ML20M dataset was chosen for regularization. Sw-dae outperforms the sparsely activated regularized versions of DAE.

Method	NDCG@100	Recall@20	Recall@50
Sw-dae	**0.442**	**0.410**	**0.549**
Fc-reg $\lambda_1 = 10^{-5}$	0.429	0.396	0.533
Fc-reg $\lambda_1 = 10^{-4}$	0.430	0.398	0.534
Fc-reg $\lambda_1 = 10^{-3}$	0.431	0.399	0.535
Fc-reg $\lambda_1 = 10^{-2}$	0.425	0.395	0.533
Fc-reg $\lambda_1 = 10^{-1}$	0.376	0.343	0.476

Table 6: The comparison of the running time of the components of two item grouping methods (Algorithm 1 & 2) in seconds along with their respective accuracies.

ML20M	Spectrum	K-means	Total	Recall@50	NDCG@100
Algorithm 1	155 sec.	528.6 sec.	683.6 sec.	0.549	0.442
Algorithm 2	41.3 sec.	833.1 sec.	874.4 sec.	0.545	0.441
Netflix					
Algorithm 1	238.5 sec.	1591.1 sec.	1829.6 sec.	0.458	0.404
Algorithm 2	176.8 sec.	400.9 sec.	577.7 sec.	0.457	0.403
MSD					
Algorithm 1	1229.8 sec.	1068.9 sec.	2298.7 sec.	0.416	0.372
Algorithm 2	273.5 sec.	1267.8 sec.	1541.3 sec.	0.414	0.371

also drops. Prune-90 has the same 10% sparsity level as Sw-dae, but its performance is much lower than Sw-dae.

We can also make the activation of the hidden layer sparse by introducing a ℓ_1 penalty (λ_1) on the activation of the hidden layer. Again we use the best Fc-dae based on the validation performance for the experiments and compare its performance with Sw-dae. Table 5 shows this comparison on the ML20M dataset. Sw-dae is considerably better than any regularized model. We can see that too little or too much regularization are both undesirable and there exists an optimal level in-between (around $\lambda_1 = 10^{-3}$) that gives the best performance.

5.7 Analysis of Item Grouping

Table 6 shows the comparison of Algorithm 1 and Algorithm 2 in terms of the running time and the recommendation performance. We see that both the algorithms are reasonably fast even on the lager Netflix and MSD datasets. Also, Algorithm 2 is faster on all datasets in getting the top F eigenvectors due to operating on the more sparse matrix $\mathbf{X}$ directly. We also see that the time taken by K-means to cluster the projected items is not a lot. This is due to the items being F-dimensional vectors where F is small. As a result, both the algorithms scale well to large and sparse data. Finally, we note that, as expected, Algorithm 1 provides better accuracy on all three datasets due to discounting for item popularity by normalizing the Laplacian of Equation 1.

Table 7: Sw-dae outperforms the state-of-the-art Mult-dae/vae in the cold-start scenario on the ML20M dataset.

	Recall@20	Recall@50	NDCG@100
Sw-dae	**0.460**	**0.619**	**0.343**
Mult-vae	0.456	0.617	0.337
Mult-dae	0.448	0.614	0.328

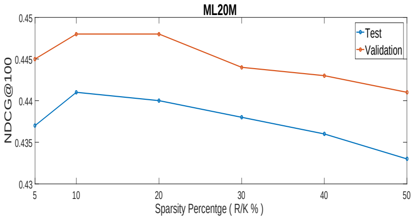

Figure 6: Effect of sparsity on validation and test NDCG@100 on ML20M dataset. X-axis denotes the sparsity level: higher means more dense.

5.8 Cold Start

We compare the performance of Sw-dae in the cold-start scenario with Mult-dae/vae as they represent the best two baselines. As before we first sample the test users and then we randomly sample 80% of the events (fold-in set) of the test users for learning their representations and use the remaining 20% events of the test users for testing. From the test users, we select the cold-start users based on their activity in the fold-in set. The activity is defined by the number of events of each user in the fold-in set. Testing is then done only on these cold-start users. Table 7 shows this comparison for the ML20M dataset. Consistent with [16] we find that Multi-vae performs better than Multi-dae for the user cold-start. However, we see that Sw-dae outperforms Multi-vae. We conjecture that the addition of cluster structure acts as an additional source of information that enables better performance when the user data is scarce. A similar trend is observed on the other datasets.

5.9 Effect of Sparsity: R/K

In our experiments, we set the sparsity level to 10% i.e., $R/K = 0.1$ for all the datasets. In this section we empirically investigate the effect of the sparsity level (R/K) on the performance of Sw-dae. Figure 6 shows the effect of the sparsity level on the validation and test NDCG@100 on ML20M and similar trends were observed for other datasets. The x-axis denotes the sparsity level where larger values denote a more dense network. We can see that the best validation performance was obtained for 10% and 20% sparsity. Based on this the 10% sparsity was chosen for the experiments in the paper. We also note that very sparse and very dense networks both lead to unsatisfactory performance and there exist optimal values of sparsity between 10% and 20%.

Figure 7 shows the effect of sparsity on the test Recall@20 and test Recall@50. We observe a similar trend as the NDCG@100 graph. There exists an optimal level of sparsity between 10% and 20% that gives the best performance.

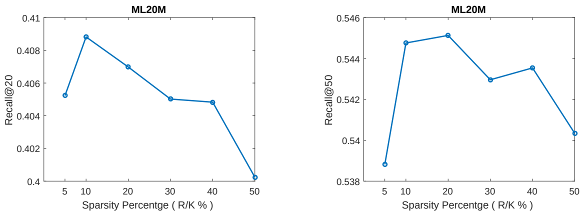

Figure 7: Effect of sparsity on test Recall@20 and Recall@50 on ML20M dataset.

Table 8: Sw-dae detects a variety of related items. Below we see four sample neurons that detect documentary, horror, drama, and children genres on the ML20M dataset.

Neuron ID	Top three genres
2785	Documentary(35%), Drama(39%), Comedy(32%)
691	Horror(73%), Thriller(37%), Sci-fi(17%)
1247	Drama(88%), Crime (25%), Action (25%)
387	Children(42%), Comedy(44%), Drama(30%)

5.10 Qualitative Analysis

We also examine the resulting Sw-dae structure qualitatively. For this, we obtained the genre information of each movie in the ML20M dataset from IMDB. In Table 8 we show four sample neurons where each neuron represents an overlapping cluster. The top three genres for each overlapping cluster are shown along with the percentage of movies in the cluster that possess this genre. Each movie can possess more than one genre. We see that neuron 691 primarily detects the genres associated with horror i.e., horror-thriller-scifi movies, similarly, other neurons detect the documentary (neuron 2785), drama(neuron 1247) and children (neuron 387) genres. Thus, the overlapping clusters can pick up related items.

In Figure 8 we pick the top 25 neurons possessing a specific genre and then examine the other genres present in the overlapping cluster represented by this genre. Specifically, we first select the top 25 neurons with respect to the percentage of movies of a specific genre present in them. Then for each of these neurons, we calculate the percentage of other genres present. Figure 8 shows the heat maps for three genres: musical, romance, and sci-fi. The y-axis represents the neuron IDs and the x-axis represents the various genres. The color denotes the degree to which a genre is present in the corresponding cluster. We can see that the neurons possessing the musical genre also possess a high percentage of comedy, drama, children genres but do not possess the movies with mystery, western, noir genres. Similarly, the neurons that possess the romance genres also possess movies with drama and comedy genres but rarely possess the horror, mystery and war genres. Finally, the sci-fi movies are generally not grouped with IMAX, musical and children movies but are frequently grouped with thriller, horror, action, etc. genres. This shows that the techniques employed for overlapping clustering result in thematically similar overlapping clusters, and movies with complementary genres are grouped together.

6 CONCLUSION

Existing autoencoders use fully-connected layers for making recommendations. They fail to model the tendency of items to be related to only a subset of all items. Rather they connect each neuron to all items and rely on the network to automatically determine which interactions to model. This becomes especially difficult when the data is sparse. To overcome this we proposed the use to structure learning to decide the connectivity pattern of the neurons. We showed that existing structure learning methods can be adopted for recommendation and they outperform the state-of-the-art methods. We then presented a two-stage method that first fixes the structure based on item groups and then trains only the required connections. We conducted extensive experiments to show that our proposed structure learning technique considerably outperforms the baselines, and also performs better in the cold-start scenario. We showed that this improvement is due to having a smaller spectral norm and a lower generalization error upper-bound. Moreover, considerable improvements can be seen even when the same number of flops/parameters are used as the baselines. We also showed that the item grouping phase is fast and scalable for sparse data and the learned overlapping clusters have thematically similar items.

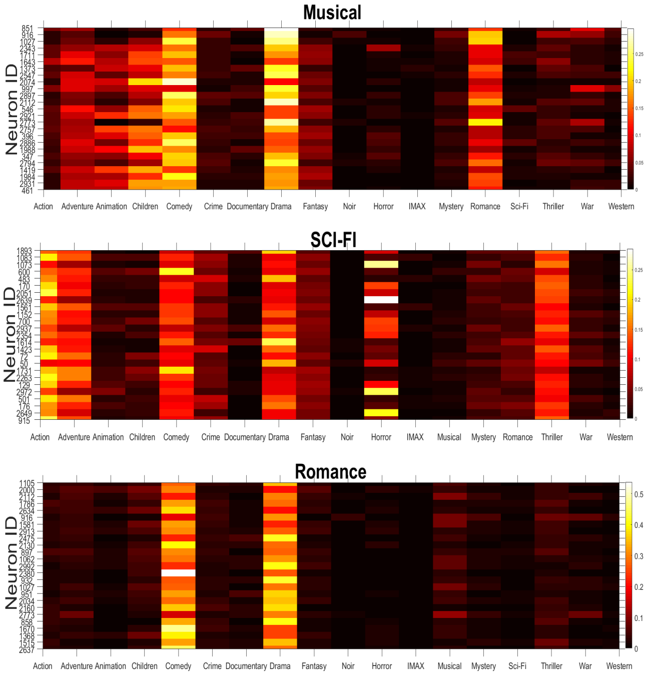

Figure 8: We pick the top 25 neurons possessing a specific genre and then examine the other genres present in the overlapping cluster represented by this genre. The y-axis denotes the neuron ID and the x-axis denotes the other genres. The color denotes the degree to which a genre belongs to a cluster. We can see that the neurons possessing the musical genre also possess a high percentage of comedy, drama, children genres. Similarly, the neurons that possess the romance genres also possess movies with drama and comedy genres but rarely possess the horror, mystery and war genres.

ACKNOWLEDGMENTS

Research on this article was supported by Hong Kong Research Grants Council under grants 16202118 and 18200519.

REFERENCES

[1] Charu C. Aggarwal. 2016. *Recommender Systems: The Textbook* (1st ed.). Springer Publishing Company, Incorporated.

[2] Xavier Amatriain, Alejandro Jaimes, Nuria Oliver, and Josep M. Pujol. 2015. *Recommender Systems Handbook*. Springer US, Boston, MA, Chapter Data Mining Methods for Recommender Systems.

[3] James Baglama and Lothar Reichel. 2005. Augmented implicitly restarted Lanczos bidiagonalization methods. *SIAM Journal on Scientific Computing* 27, 1 (2005), 19–42.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[5] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.

[6] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 173–182.

[7] Reinhard Heckel, Michail Vlachos, Thomas Parnell, and Celestine Dünner. 2017. Scalable and interpretable product recommendations via overlapping co-clustering. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*. IEEE, 1033–1044.

[8] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM '08)*. IEEE Computer Society, Washington, DC, USA, 263–272. https://doi.org/10.1109/ICDM.2008.22

[9] Xiaojuan Jiang, Yinghua Zhang, Wensheng Zhang, and Xian Xiao. 2013. A novel sparse auto-encoder for deep unsupervised learning. *2013 Sixth International Conference on Advanced Computational Intelligence (ICACI)* (2013), 256–261.

[10] Farhan Khawar and Nevin L Zhang. 2019. Conformative filtering for implicit feedback data. In *European Conference on Information Retrieval*. Springer, 164–178.

[11] Farhan Khawar and Nevin L Zhang. 2019. Modeling Multidimensional User Preferences for Collaborative Filtering. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1618–1621.

[12] Mohammad Khoshneshin and W Nick Street. 2010. Incremental collaborative filtering via evolutionary co-clustering. In *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 325–328.

[13] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Yehuda Koren and Robert Bell. 2015. *Recommender Systems Handbook*. Springer US, Boston, MA, Chapter Advances in Collaborative Filterin, 77–118.

[15] R. Lehoucq and D. Sorensen. 2000. *Templates for the solution of algebraic eigenvalue problems: a practical guide*. SIAM, Philadelphia, Chapter Implicitly Restarted Lanczos Method (Section 4.5).

[16] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational Autoencoders for Collaborative Filtering. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 689–698.

[17] Benjamin Marlin. 2004. *Collaborative filtering: A machine learning perspective*. University of Toronto.

[18] Brian McFee, Thierry Bertin-Mahieux, Daniel PW Ellis, and Gert RG Lanckriet. 2012. The million song dataset challenge. In *Proceedings of the 21st International Conference on World Wide Web*. ACM, 909–916.

[19] Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. 2017. A pac-bayesian approach to spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1707.09564* (2017).

[20] Andrew Y Ng, Michael I Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*. 849–856.

[21] Xia Ning and George Karypis. 2011. Slim: Sparse linear methods for top-n recommender systems. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. IEEE, 497–506.

[22] Braxton Osting, Chris D White, and Édouard Oudet. 2014. Minimal Dirichlet energy partitions for graphs. *SIAM Journal on Scientific Computing* 36, 4 (2014), A1635–A1651.

[23] Mark O'Connor and Jon Herlocker. 1999. Clustering items for collaborative filtering. In *Proceedings of the ACM SIGIR workshop on recommender systems*, Vol. 128. UC Berkeley.

[24] Badrul M Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2002. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the fifth international conference on computer and information technology*, Vol. 1. 291–324.

[25] Lyle H Ungar and Dean P Foster. 1998. Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems*, Vol. 1. 114–129.

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

[27] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 153–162.

[28] Yao Wu, Xudong Liu, Min Xie, Martin Ester, and Qing Yang. 2016. CCCF: Improving Collaborative Filtering via Scalable User-Item Co-Clustering. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining (WSDM '16)*. ACM, New York, NY, USA, 73–82. https://doi.org/10.1145/2835776.2835836

[29] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. 2014. Personalized entity recommendation: A heterogeneous information network approach. In *Proceedings of the 7th ACM international conference on Web search and data mining*. ACM, 283–292.

[30] Dominique Zosso, Braxton Osting, and Stanley J Osher. 2015. A dirichlet energy criterion for graph-based image segmentation. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. IEEE, 821–830.