Movie Recommedation System Enhanced by Natural Language Processing

Jiaxi Chen
Department of Computer Science
Stanford University
jiaxi@stanford.edu

Ziran Zhang
Department of Computer Science
Stanford University
zirzhang@stanford.edu

Abstract

The online recommendation system often suffers from sparsity problem, which is also known as the "cold start" issue. Sparsity problem means that the recommendation system is unable to properly predict users preference without sufficient data on either user side or item side. In this project, the main problem we investigated is how to build a movie recommendation system using deep learning and natural language processing approach overcoming the "cold start" issue. We tried two approaches to build the system. The first approach is utilizing autoencoder with the use of movie data. The second approach is using a score model inspired by word2vec by training embedding matrixes for both users and movies, and then output a confident score associate with predicted ratings. The major findings in this project are the denoising autoencoder has the potential to build an effective movie recommendation system, and the natural language processing techniques making use of movie metadata (e.g. genre, overview, keyword) is a good way to alleviate the cold start issue as it regularizes the model.

1 Introduction

The problem we will be investigating is to give movie recommendation to user based on his/her previous movie ratings and movie contents. It is interesting because watching a movie usually takes $2 \sim 3$ hours, which is a long time duration. We hope that our recommendation system would provide users with movies they are likely to enjoy rather than wasting time on movies they don't like. The challenge is that each movie contains tons of metadata (e.g. genre, overview, languages, credits...), and we are not sure which types of the data are helpful during our model training. In addition, it is difficult to integrate this metadata into traditional recommendation system built on network structure. The dataset we plan to use is 'The Movie Dataset' [Fabien D(2017)] available publicly from kaggle.com.

Our overall approaches are using an autoencoder and a score model. For the first approach we predict users' ratings on movies they haven't rated using an autoencoder. Our goal is for every incomplete $R_{\cdot j} \in \mathbb{R}^N$ vector of j^{th} movie, predict full vector $\hat{R}_{\cdot j}$. For the second approach we learn the embeddings for both movies and users in our dataset. The model will generate a set of vector representations of movies and users during training, and use these vectors to predict user's preference on the corresponding movie.

2 Related Work

There are plenty of research work and existing frameworks for recommendation systems. Collaborative Filtering [Hill et al.(1995)Hill, Stead, Rosenstein, and Furnas] [Konstan et al.(1997)Konstan, Miller, Maltz, Herlocker, Gordon, and Riedl] is a popular

and powerful technique employed by many recommendation systems, such as Google News Personalization [Das et al.(2007)Das, Datar, Garg, and Rajaram] and Amazon.com [Linden et al.(2003)Linden, Smith, and York]. To explain in our movie recommendation system scenario, the principle of Collaborative Filtering is to predict all the movies a user has not rated based on his/her existing ratings on a set of movies he/she does rate. Network embedding, such as node2vec [Grover and Leskovec(2016)], is another well-known method to represent relationship of nodes in graphs. Recently, as the emerging of deep learning techniques, researchers also did a few attempts on recommendation system based on both Neural Network models [LeCun et al.(2015)LeCun, Bengio, and Hinton] and Recurrent Network models [Wang et al.(2016)Wang, Xingjian, and Yeung].

However, these techniques only focus on analysis of interaction between items, and often suffer from sparsity problem. Inspired by the idea we learned from hybrid recommender system based on autoencoder [Strub et al.(2016)Strub, Gaudel, and Mary], we would like to utilize intrinsic metadata of movies as additional features to learn latent factors of users' preference over movies, and alleviate the 'cold start' issue by making use of the side information.

3 Approach

3.1 Datasets

The dataset we are using is The Movie Dataset [2] available publicly from kaggle.com. It consists of more than 26,000,000 ratings from over 270,000 users on 40,000 movies. By average, each user gives nearly 100 ratings, while each movie receives 650 ratings from users. Given the large quantity of data, this dataset is a great source to train a recommendation system on. In addition to rating, The Movie Dataset also contains side information for each movie, such as an overview of the movie content, several genres of the movie, and some keywords. These features can alleviate the cold start issue mentioned above with the help of NLP techniques.

3.2 Data Preprocessing

The primary issue of the existing dataset is that, if we simply convert the dataset of 40,000 movies and 270,000 users to a 40,000 by 270,000 rating matrix in which $R_{ij} = 0$ if movie i does not receive rating from user j, the size of the matrix is extremely large. This leads to two problems. First, the large size of training data will decrease the training speed. Second, the dimension of input vector will increase the number of weight for input layer. Both will make the training significantly inefficient. Moreover, recommendation system based on Autoencoder only works well on the incomplete matrix with sparsity no less than 5% [Strub et al.(2016)Strub, Gaudel, and Mary]. However, in our dataset, there are only 26,000,000 ratings, while the size of matrix is 10,800,000,000. The sparsity of this matrix is 0.25%, which fails to meet the basic requirement for Autoencoder.

To solve the issue, we set threshold for both user and movie, and only select user and movie that are considered to be valuable for the Autoencoder. Specifically, we first count the number of ratings given by each user, and select users who give more than 300 ratings. This results in 20,133 users out of 270,000. Next, we go over all the movies in the dataset, and select movies which receive more than 300 ratings from the group of users we generated in previous step. By doing so, we decrease the number of movie in the matrix from 40,000 to 6,672. This data preprocessing results in a 6,672 by 20,133 matrix with 13% sparsity.

3.3 Denoising AutoEncoder (DAE) Model

3.3.1 Model Architecture

Figure 1 is the model structure and data processing steps of the DAE model.

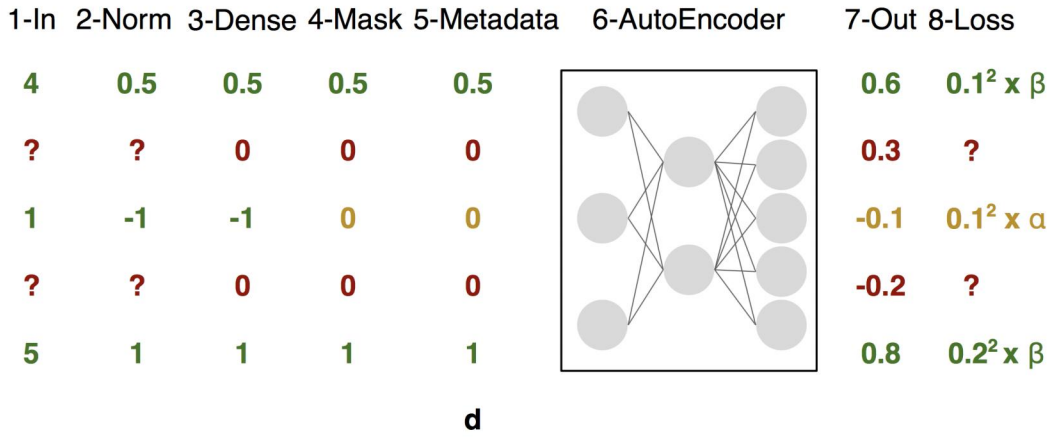

Figure 1: Denoising AutoEncoder model

Given the input movie vector, we first normalize the ratings to a $[-1, 1]$ scale, which allows a tanh activation output layer to output ratings at the same scale without further processing; then we densify the vector by filling unknown ratings with 0's; next, 30% of random known values are masked to 0 to enable the usage of denoising loss function; finally the d-dimensional metadata m_j of the j-th movie is appended to the vector and is fed into the AutoEncoder. We first implement a shallow AutoEncoder with one hidden layer to get a baseline model. The predicting task is achieved by performing a forward pass through our neural network:

$$\hat{R}_{\cdot j} = g^{[2]}\left(W^{[2]} \cdot g^{[1]}\left(W^{[1]} \cdot [R_{\cdot j}, m_j] + b^{[1]}\right) + b^{[2]}\right) \tag{1}$$

where $g^{[1]}, g^{[2]}$ are non-linear activation function for different layers, $W^{[1]} \in \mathbb{R}^{n^{[1]} \times N}, b^{[1]} \in \mathbb{R}^{n^{[1]}}, W^{[2]} \in \mathbb{R}^{M \times n^{[1]}}, b^{[2]} \in \mathbb{R}^{M}$ are the weights and biases to be learnt in the network. Later on we will experiment with deeper models and more complex network architectures for (hopefully) better performance. For our baseline model, we currently choose the activation function $g^{[1]} = ReLu, g^{[2]} = tanh$ and the hidden layer size $n^{[1]} = 750$ but these hyperparameters of model architecture are to be tuned by experiments in the future.

The concerns for incorporating metadata of movies is that it may help alleviate "the cold start" problems when there are very few users rating on particular movies. The intuition is to add the internal information of the movies to *bias* the missing ratings. In our neural network implementation, such method could be applied by appending metadata's vector representations to the original input vectors. For baseline model, we apply the CBOW [Mikolov et al.(2013)Mikolov, Chen, Corrado, and Dean] technique by adding up word vectors of movie keywords as the metadata and only apply the metadata at the input level. Again, later on we will experiment with other metadata representations and applications at different layers to compare the results.

3.3.2 Loss Function

The major challenge of recommendation systems is that the input vector is incomplete and usually very sparse. The training algorithm must treat those missing values differently from existing values with special care throughout the whole process. In our implementation, we distinguish missing values by setting them to zero in the input vector and design the loss function to include errors caused only by existing values.

A vanilla loss function with L2 regularization would be:

$$\mathcal{L}(R_{\cdot j}, \hat{R}_{\cdot j}) = \frac{1}{2} \sum_{i \in \mathcal{E}(R_{\cdot j})} \left(\hat{R}_{ij} - R_{ij}\right)^2 + \frac{\lambda}{2} \|\mathbf{W}\|_2^2 \tag{2}$$

where $\mathcal{E}(R_{\cdot j})$ is the set of indices of existing values in $R_{\cdot j}$.

However, one problem with this loss function is that optimization under such loss function will focus on reconstructing the existing values in the initial input vectors without caring about predicting missing values, i.e. overfitting the existing values. To generalize well to predict missing values, we adopt the Denoising AutoEncoder's (DAE) [Vincent et al.(2008)Vincent, Larochelle, Bengio, and Manzagol] loss function. The idea of DAE is to mask existing values (set values to zero in our case) in input vectors and predict them as missing values. Thus the loss function will emphasize on predicting missing values as well as recover input vectors. The DAE loss function is:

$$\begin{aligned}\mathcal{L}_{\text{DAE}}(R_{\cdot j}, \hat{R}_{\cdot j}) = \frac{\alpha}{2} \sum_{i \in \mathcal{E}(R_{\cdot j}) \cap \mathcal{M}(R_{\cdot j})} \left(\hat{R}_{ij} - R_{ij}\right)^2 + \\ \frac{\beta}{2} \sum_{i \in \mathcal{E}(R_{\cdot j}) \setminus \mathcal{M}(R_{\cdot j})} \left(\hat{R}_{ij} - R_{ij}\right)^2 + \frac{\lambda}{2} \|\mathbf{W}\|_2^2\end{aligned} \tag{3}$$

where $\mathcal{M}(R_{\cdot j})$ is the set of indices of masked elements in $R_{\cdot j}$. Note that α and β are hyperparameters reflecting respectively how much we care about the predicting goal and the reconstructing goal. In our case we choose $\alpha > \beta$ to address the problem mentioned about loss function Equation 2.

3.4 Score Model

Inspired by word2vec [Mikolov et al.(2013)Mikolov, Chen, Corrado, and Dean], we came up with another approach which learns denser representation embedding for movies and users.

3.4.1 Model Architecture

Different from DAE Model, the Score Model will perform a binary classification task. The input of the model is movie i and user j, and the output will be a value range in $(0, 1)$. We only use existing ratings from our dataset and use the true rating as label during training. A prediction value closing to one indicates that the user will like this movie and vice versa.

A. Score Model without Neural Network
We first implemented a baseline model without utilizing a neural network. We initialized two embeddings M and U for both movies and users. The embedding for each movie or user is a $1 \times d$ vector, where d refers to the dimension of vector representation. Then we use the following function to compute a score which indicates the preference of a user on a movie:

$$\hat{P}_{ij} = \sigma(M_i \cdot \mathbf{U_j}^\intercal) \tag{4}$$

where M_i is the movie vector for the ith movie and U_j is the user vector for the jth user. $hatP_{ij}$ is the models prediction on the preference of jth user on ith movie, which ranges in $(0, 1)$.

B. Score Model with Neural Network
We realized that a naive Score Model cannot fully represent the characteristic of movie and user, as well as their relationship. Therefore, we design a shallow neural network with l hidden layers, where each layer contains n neurons. For each layer, we use tanh as activation function. We create such neural network for movie vector and user vector separately. Suppose both movie and user embeddings are $1 \times d$ vectors, the output of the neural they are corresponding to will be a $1 \times n$ vector. Similar to score model without neural network, we will follow Equation 5 to predict the preference of this user on this movie.

Moreover, we can add the metadata information to this neural network as well. To be specific, before we feed a movie vector into its neural network, we append the vector representation of its metadata to the original input vectors, i.e, the average of word vectors in the movie overview. Let's say the shape of word vector is $1 \times w$, the input for movie neural network will become $1 \times (d + w)$, while the output will still be $1 \times n$. By doing so, we make use of the internal information of the movies as bias in the neural network.

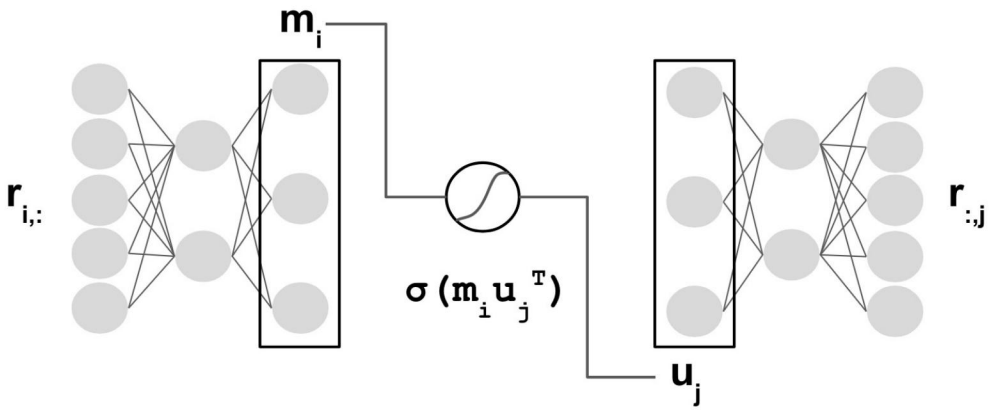

Figure 2: ScoreModel with Neural Network

3.4.2 Loss Function

The Score Model is essentially a binary classification model. Note that weve already transformed the rating matrix to a binary one. As a result, logistic loss is used:

$$\mathcal{L}_{\text{Score}}(R, \hat{R}) = \sum_{i,j \in \mathcal{E}(R)} -y_{i,j} \log \hat{y}_{i,j} - (1 - y_{i,j}) \log (1 - \hat{y}_{i,j}) \tag{5}$$

4 Experiments

We tested both DAE Model and Score Model on our 2,095 by 7,941 matrix. We shuffled the dataset, and then divide it in the following ways: 80% for training set, 10% for development set, and 10% for test set.

To speed up training, we utilized batch gradient descent such that trainable variables will be updated for every 2100 samples for DAE Model and 210 samples for Score Model. The difference between batch size results from different size we used for two models. For DAE Model, we want to go over all the grids inside matrix, including those without ratings. The total number is 16,636,395. For Score Model, however, we only used existing ratings, which only occupies 7% of the entire matrix.

4.1 Evaluation

To evaluate the model performance, we introduce the accuracy metric and coefficient of determination (r^2) metric. We use r^2 as the metric of evaluating our AutoEncoder model instead of Root Mean Square Error (RMSE) used in our milestone. They are both metrics for regression problems but it is hard to interpret the RMSE metric. The r^2 metric is defined as follows:

$$r^2 = 1 - \frac{\sum |y - \hat{y}|^2}{\sum |y - \overline{y}|^2}$$

which lies in range $[-1, 1]$ and indicates the percentage of improvement over a baseline which simply output mean value of input values, as illustrated in Figure 3.

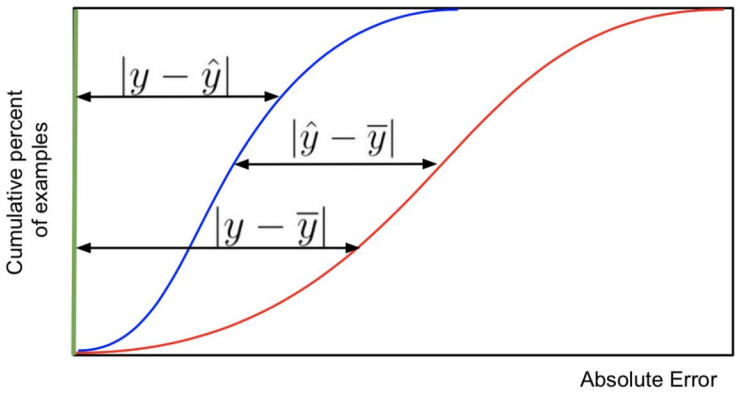

Figure 3: Coefficient of Determination Metric

In contrast, in accuracy metric of DAE model, we set a threshold to decide if a rating indicates like or dislike. By doing so, we convert both actual ratings and predicted ratings to binary form of like and dislike. Then we compute the prediction accuracy by taking the number of correct predictions over the number of ratings.

Since Score Model itself is a binary classification problem, we can only use accuracy metric on it. Given that the output of Score Model range in $0, 1$, we choose 0.5 as the threshold for predicting like or dislike based on the distribution on like and dislike in the converted dataset. Then we compute accuracy for the model.

4.2 Result

We run both models for 100 epochs, with 10^{-3} learning rate, 0.1 l2-regularization rate, and 0.8 dropout rate. We get the following results:

4.2.1 Denoising AutoEncoder (DAE) Model

Figure 4 shows the results we get for different autoencoder architectures. We first use one hidden layer with 750 hidden neurons without metadata. The model overfits quickly and the r^2 metric gets stuck around 0.167. The reason may be that we have a very high dimensional input and output resulting a large amount of parameters. To alleviate this problem, we first try to reduce the dimension of input vector by doing a principal component analysis on the input matrix and apply the autoencoder to the principal components. But this approach makes the performance even worse, the possible reasons are: 1) the model bias is enlarged by trying to recover the imprecise components; 2) the principal components for train/dev/test sets are different. We then try decreasing the first and last hidden layer sizes from 750 to 50 and stack another layer of size 100 in between. This increases the dev r^2 to 0.418. The intuition behind this architecture is that by decreasing the size of first and last hidden layer we could not only decrease the number of parameters but also learn denser features of the movie vector; and by stacking another layer we could learn more complex features with fewer parameters. Lastly, by adding metadata on top of this architecture, the r^2 is increased to 0.496.

4.2.2 Score Model

After tuning hyperparameters, we get the best result when vector dimension is 50, number of hidden layers in neural network is 2, and number of neurons in each layer is 60. The accuracy is shown in Table 1:

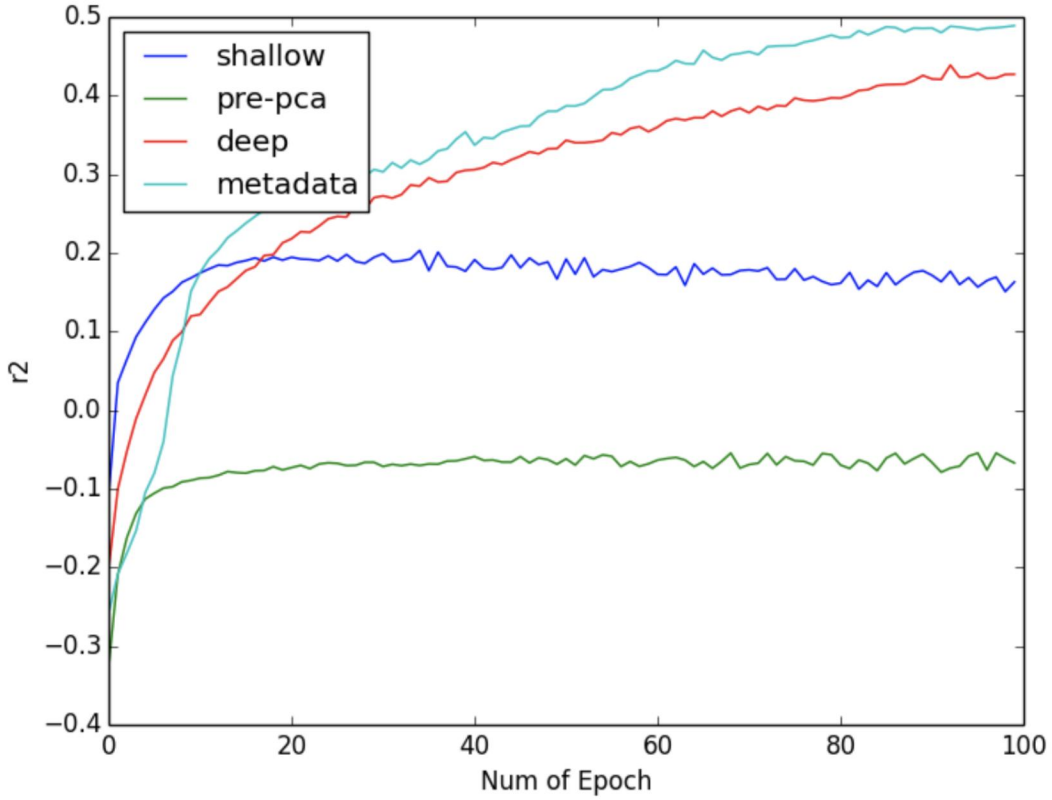

Figure 4: Experiment Results of DAE Model

Table 1: Accuracy on different models

Model	Accuracy
Score Model without neural network	62.9%
Score Model with neural network	70.9%
Score Model with neural network plus metadata	71.1%
DAE Model	70.2%
DAE Model plus metadata	74.3%

We observe that even with a shallow neural network with 2 hidden layers, the Score Model with neural network will greatly outperform the one without hidden layers. The neurons inside such layer enhance the representation of movies and users as a vector. Another observation is that the metadata has a smaller effect on the performance comparing with DAE Model. One possible reason is that the dimension for movie embedding is 50, while the dimension for metadata is also 50. Comparing with DAE Model which concatenates a 1×50 vector to a much larger 1×8000 vector, score model can hardly differentiate the weight for movie embedding versus metadata vector in the neural network. In DAE Model, however, the 1×50 metadata vector is just a small component in the input vector, and can easily trained as regularization factor.

5 Conclusion

In this project, we have learned that in our Denoising Autoencoder Model, we can build an effective movie recommendation system by making use of textual data of movie. By appending genre, overview and keywords information onto movie vector, we improve the Denoising Autoencoder

Model performance comparing the Autoencoder model without metadata. In addition, we used an entirely different model, Score Model, to build the movie recommendation model. Inspired by word2vec, we trained vector representations for every user and movie with neural network, and compute the dot product of those two vectors as our confidence score showing users preference over movies. We also incorporate the textual data of movie into Score Model which achieves the best accuracy among our few tuned score model. It turns out that in our current setting, the Denoising Autoencoder Model still outperforms the Score Model.

In the future, to possibly improve our models we can integrate RNN techniques with our denoising autoencoder model and score model, or even build a completely new model on mainly RNN.

6 Note

This project is shared with final project of CS230 Deep Learning, in which we mainly focused on Denoising Autoencoder Model. In CS224N, thanks to suggestions given by Prof. Socher, we tried Score Model, which is similar to word2vec, and examine its performance on the task. Also, we make use of NLP representation of textual data to improve the performance of both model.

References

[Das et al.(2007)Das, Datar, Garg, and Rajaram] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.

[Fabien D(2017)] Martin E Fabien D, Rounak B. The movie dataset. `https://www.kaggle.com/rounakbanik/the-movies-dataset`, 2017.

[Grover and Leskovec(2016)] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[Hill et al.(1995)Hill, Stead, Rosenstein, and Furnas] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201. ACM Press/Addison-Wesley Publishing Co., 1995.

[Konstan et al.(1997)Konstan, Miller, Maltz, Herlocker, Gordon, and Riedl] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40 (3):77–87, 1997.

[LeCun et al.(2015)LeCun, Bengio, and Hinton] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[Linden et al.(2003)Linden, Smith, and York] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7 (1):76–80, 2003.

[Mikolov et al.(2013)Mikolov, Chen, Corrado, and Dean] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[Strub et al.(2016)Strub, Gaudel, and Mary] Florian Strub, Romaric Gaudel, and Jérémie Mary. Hybrid recommender system based on autoencoders. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 11–16. ACM, 2016.

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

[Wang et al.(2016)Wang, Xingjian, and Yeung] Hao Wang, SHI Xingjian, and Dit-Yan Yeung. Collaborative recurrent autoencoder: recommend while learning to fill in the blanks. In *Advances in Neural Information Processing Systems*, pages 415–423, 2016.