Autoencoder-Based Collaborative Filtering

Yuanxin Ouyang[1,2], Wenqi Liu[1], Wenge Rong[1,2], and Zhang Xiong[1,2]

School of Computer Science and Engineering, Beihang University, China
Research Institute of Beihang University in Shenzhen, China
{oyyx@,wqliu@cse.,w.rong@,xiongz@}buaa.edu.cn

Abstract. Currently collaborative filtering is widely used in recommender systems. With the development of idea of deep learning, a lot of researches have been conducted to improve collaborative filtering by integrating deep learning techniques. In this research, we proposed an autoencoder based collaborative filtering method, in which pretraining and stacking mechanism is provided. The experimental study on commonly used MovieLens datasets have shown its potential and effectiveness in getting higher recall.

1 Introduction

With the explosion of information on the Internet, people relied on more and more recommender systems to solicit suggestions and/or make decisions, thereby solving the information overload problem. A lot of recommendation related techniques have been proposed and a notable one is collaborative filtering [1], which is widely lauded as a practical method for providing recommendations by utilising users' preference history to predict future preference. Generally, algorithms for collaborative filtering can be roughly divided into two general classes, i.e., memory-based and model-based approaches. Memory-based methods try to predict users' preference based on the ratings by other similar users, while model-based methods mainly rely on a prediction model by using Clusetering, Baysesian network and etc [3].

Currently, with the development of concept of deep learning, a new research area and has proven its success in speech and image recognition [4], researchers started to try to employ the inspiration of deep learning into collaborative filtering based recommender systems. For example, Salakhutdinov et al. proposed an approach employing Restricted Boltzmann Machines (RBM) [12] and Georgiev et al. further extended the original RBM-based model to a unified non-IID framework [5]. Truyen et al. explored joint modelling of users and items for collaborative filtering, but inside is an unrestricted version of Boltzmann Machines (BMs) [10]. Oord et al. used deep convolutional neural networks to provide music recommendation [9]. Gunawardana et al. described a tied Boltzmann Machine combining collaborative and content information [7].

Deep learning is also called feature learning due to its powerful ability to learn feature representations automatically. Besides, deep models can learn high-order features of input data which may be useful for recommendation as indicated in

C.K. Loo et al. (Eds.): ICONIP 2014, Part III, LNCS 8836, pp. 284–291, 2014.

[12]. Inspired by previous work, in this research we tried to employ another neural network model, autoencoder, into the collaborative filtering task. Experimental study on commonly used datasets is also conducted to present its potential and effectiveness.

The rest of the paper is organised as following. Section 2 will introduce the related work about collaborative filtering models and basic autoencoder. Then a modified autoencoder based collaborative filtering model will be illustrated in section 3. Section 4 will discuss the experimental study and Section 5 will conclude this paper and point out possible future work.

2 Related Work

Early approaches for collaborative filtering assume that similar users have similar interests, i.e., nearest neighbourhood based methods, which is normally called memory-based approaches. However, memory-based approaches do not scale well because they require access to the ratings of the entire set. Furthermore, there is another challenge that ratings are severe sparse making memory-based approaches perform unsatisfied. To overcome this challenges, model-based approaches such as singular value decomposition (SVD) have been proposed [6]. However, application of matrix factorization to sparse ratings matrices is still a non-trivial challenge. As such, Hoffman proposed a formal statistical model of user preferences using hidden variables over user-item-rating triplets [8].

Except for memory-based and model-based approaches, recently an alternative methods using idea of deep learning has been attached much importance. Among them autoencoder is a widely used deep learning model. Suppose we have only unlabelled training examples set $\{x^{(1)},x^{(2)},...\}$, where $x^{(i)} \in \Re^n$. An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs. I.e., it uses $y^{(i)} = x^{(i)}$, as shown in Fig. 1.

The autoencoder tries to learn a function $h_{W,b}(x) \approx x$. In other words, it is trying to learn an approximation to the identity function, so as to output $\hat{x}$ which is similar to x. By placing constraints on the network, such as limiting the number of hidden units, interesting structure can be discovered about the data. For instance, if some of the input features are correlated, then this algorithm will be able to discover some of those correlations.

3 Autoencoders for Collaborative Filtering

3.1 Modeling User-Item Ratings

Suppose there are N users, M movies and integer ratings from 1 to K. An important problem in applying autoencoders to movie ratings is how to cope with the missing ratings efficiently. We cannot simply substitute missing values with 0 because the model will think that user give a rating of 0 and learn the negative preference, which is not the truth. In this paper we use a different

autoencoder for each user, as shown in Fig. 2. Every autoencoder has the same number of hidden units, but an autoencoder only has input units for the movies rated by that user. As a result an autoencoder has few connections if that user rated few movies. Each hidden unit could then learn to model a significant dependency between the ratings of different movies. Each autoencoder only has a single training case, but all of the corresponding weights and biases are tied together. If two users have rated the same movie, their two autoencoders must use the same weights between the softmax input/output units for that movie and the hidden units. To simplify the notation, we will now concentrate on getting the gradients for the parameters of a single user-specific autoencoder. The full gradients with respect to the shared weight parameters can then be obtained by averaging over all N users.

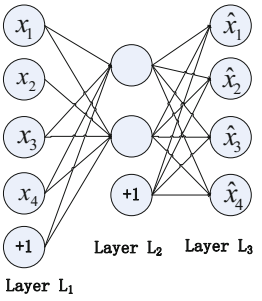

Fig. 1. Architecture of autoencoder

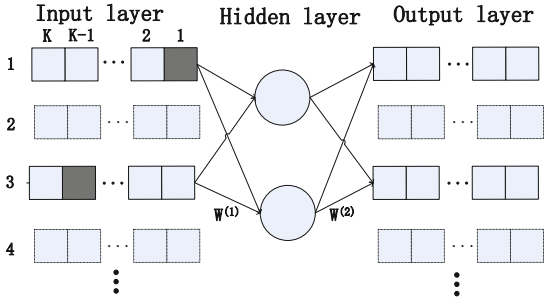

Fig. 2. An user-specific autoencoder for collaborative filtering

Learning. Suppose a user rated m movies. Let $a^{(1)}$ be a $K \times m$ observed binary indicator matrix with $a_i^{k(1)} = 1$ if the user rated movie i as k and 0 otherwise. We also let $a_j^{(2)}$, $j = 1, ..., F$, be the values of hidden variables. Here we choose activation function to be the sigmoid function. In feedforward step, the only difference is that because the output layer is of the same structure as the input layer, we compute the output unit $a_i^{k(3)}$ as:

$$a_i^{k(3)} = \frac{f(\sum_j w_{ij}^{k(2)} a_j^{(2)} + b_i^{k(2)})}{\sum_k f(\sum_j w_{ij}^{k(2)} a_j^{(2)} + b_i^{k(2)})}. \tag{1}$$

where $w_{ij}^{k(2)}$ denotes the weight associated with connection between $a_j^{(2)}$ and $a_i^{k(3)}$, $b_i^{k(2)}$ is the bias of $a_i^{k(3)}$. The denominator is the normalisation term which insures $\sum_k a_i^{k(3)} = 1$.

In backpropagation step, for a single training example $(x,\ y)$, we define the cost function $J(w, b; x, y)$ to be squared-error function. Then given a training set

of m examples, we define the overall cost function as:

$$J(w,b) = \frac{1}{m}\sum_{i} J(w,b;x^i,y^i) + \frac{\lambda}{2} \parallel w \parallel^2 . \tag{2}$$

The first item in the definition of $J(w,b)$ is an average sum-of-squares error term. the second term is a regularization term (also called a weight decay term) which tends to decrease the magnitude of the weight and helps prevent overfitting problem. Our goal is to minimise the total cost function $J(w,b)$. Here we train our autoencoder using batch gradient descent.

Making Predictions. Given the training set of one user and a new query item q, we can initialise the input layer of the autoencoder with known ratings and carry out feedforward step. Specific units $a_q^{k(3)}$ in the output layer represents the probability which item q will be rated value k. As such the expected rating for item q is computed as:

$$r_q = \sum_{k} k \cdot a_q^{k(3)}. \tag{3}$$

3.2 Initialisation of Parameters

As the optimisation problem of neural networks is nonconvex, the standard way to train autoencoders using backpropagation to reduce the reconstruction error is difficult to optimise the weights. Autoencoders with random small initial weights typically find poor local minima. A popular solution to this problem is greedy "pretraining" procedure. In this paper we use a two-layer network called Restricted Boltzmann Machine (RBM) to pretrain autoencoders. A RBM is a specific type of undirected bipartite graphical model consisting of two layers of binary variables: hidden and visible with no intra-layer connections. Training an autoencoder with RBM pretraining takes the following steps:

1) Train a RBM with analogous structure of autoencoder using input data.

2) Use the trained parameters of RBM to initialize corresponding weights and biases of autoencoder.

3) Fine-tune the weights using Backpropagationfor for optimal reconstruction of each user's ratings.

The key idea is that the greedy learning algorithm will perform a global search for a good, sensible region in the parameter space [11]. Therefore, with this pretraining, we will have a good data reconstruction model. Backpropagation is better at local fine-tuning of the model parameters than global search. So further training of the entire autoencoder using backpropagation will result in a good local optimum.

3.3 Deep Generative Models

A stacked autoencoder is a neural network consisting of multiple layers of autoencoders in which the outputs of each layer is wired to the inputs of the successive

layer. A good way to obtain good parameters for a stacked autoencoder is to use greedy layer-wise training [2]. To do this, first we train the first layer on raw input to obtain parameters and transform the raw input into a vector consisting of activation of the hidden units. The second layer is then trained on this vector. Repeat for subsequent layers, using the output of each layer as input for the subsequent layer. While training each layer, we can also use RBM pretraining method to get better local optimum as mentioned in the previous subsection.

This method trains the parameters of each layer individually while freezing parameters for the remainder of the model. To produce better results, after training phase is complete, fine-tuning using backpropagation can be used by tuning the parameters of all layers are changed at the same time.

A stacked autoencoder enjoys all the benefits of any deep network of greater expressive power. Further, it often captures a useful hierarchial grouping or part-whole decomposition of the input. The first layer of a stacked autoencoder tends to learn first-order features of the raw input. Higher layers tend to learn even high-order features corresponding to patterns of previous-order features.

4 Experiments and Discussion

4.1 Datasets and Evaluation Metrics

We evaluated the above-described autoencoders on two MovieLens datasets, which are commonly used for evaluating collaborative filtering algorithms.

The first dataset (MovieLens 100k) consists of 100,000 ratings for 1,682 movies assigned by 943 users, while the second one (MovieLens 1M) contains one million ratings for 3,952 movies by 6,040 users. Each rating is an integer between 1 (worst) to 5 (best). For both datasets, we use 80% to make training set and others to be testing set.

To evaluate the proposed method, we use both mean absolute error (MAE) and root mean squared error (RMSE). MAE measures the deviation of the predicted values p_i from their true ratings r_i, which computes the absolute difference over all N pairs. Compared with MAE, RMSE gives more weights for prediction with bigger errors. The evaluation rules are in the following form:

$$MAE = \frac{\sum_{i=1}^{N} \mid r_i - p_i \mid}{N} \qquad RMSE = \sqrt{\frac{\sum_{i=1}^{N} (r_i - p_i)^2}{N}} \tag{4}$$

4.2 Results and Discussion

Fig. 3 shows the dependency of MAE on the number of units in the hidden layer when the autoencoders are trained for 200 epochs. We can see that models get lower MAE values with the increasing number of hidden units, which is not obvious after hidden units are more than 120. It can be imagined that overfitting will become an issue with continuously increasing number of hidden units. Next, Fig. 4 shows the dependency of MAE on the number of epochs when

the autoencoders are trained for 100 hidden units. The curve is similar to the previous one. After the number of training epochs is larger than 250, the MAE value stays relatively stable. Training epochs needed to acquire stable MAE can be different if learning rate is changed or with stochastic gradient descendent.

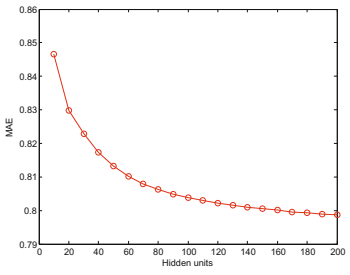

Fig. 3. Dependency of MAE on the Number of Hidden Units

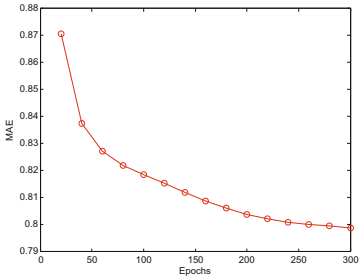

Fig. 4. Dependency of MAE on the Number of Epochs

We further compared the prediction quality achieved by different methods using the MovieLens 100k and 1M datasets, respectively. Apart from the autoencoders described before, other collaborative filtering approaches based on nearest neighbours, SVD and RBM are included.

Table 1 shows the MAE and RMSE values of some basic models and autoencoder models on both datasets. From the results on MovieLens 100k, it can be seen that autoencoders without pretraining do not perform well enough, while RBM-pretrained autoencoders have similar performance with nearest neighbors and SVD models. Besides, the results of stacked autoencoders are slightly superior to autoencoders. But the gap is not obvious.

Evaluation measures on MovieLens 1M are shown on the right side. Apart from that models perform better than those on MovieLens 100k, the trend of results do not have big differences.

Finally, experiments are made to test the overlap degree between recommended user-movie sets from autoencoders and other CF models. We define recommended movie as that whose predict and real ratings are both higher than 4. Under this condition, the prediction is precise and users have positive references over these movies. Statistical data on MovieLens 100k are shown in Table 2.

USER-BASED & AUTOENCODER represents the intersection of recommended user-movie sets between user-based CF model and autoencoder model. Comparing the first, fourth and fifth row, we can find that there is still a large number of recommended movies beyond the intersection. Same phenomenon appears between autoencoders and other CF models.

After investigating the experimental result, some interesting findings can be revealed:

Table 1. Prediction Quality on MovieLens Dataset

	MovieLens 100k		MovieLens 1M	
CF Model	RMSE	MAE	RMSE	MAE
USER-BASED CF	0.937	0.736	0.915	0.709
ITEM-BASED CF	0.932	0.732	0.901	0.698
SVD	0.940	0.737	0.893	0.684
BIASED-SVD	0.926	0.721	0.887	0.681
RBM	0.953	0.752	0.918	0.710
AUTOENCODER(NO PRETRAINING)	1.004	0.804	0.966	0.754
AUTOENCODER(PRETRAINED)	0.939	0.737	0.892	0.688
STACKED AUTOENCODER(NO PRETRAINING)	0.992	0.791	0.957	0.747
STACKED AUTOENCODER(PRETRAINED)	0.933	0.728	0.890	0.684

Table 2. Size of Recommended User-Movie Set on MovieLens 100k

CF Model	Size of Recommended User-Movie Set
USER-BASED	3244
ITEM-BASED	3299
SVD	3316
AUTOENCODER	2622
USER-BASED & AUTOENCODER	1880
ITEM-BASED & AUTOENCODER	2138
SVD & AUTOENCODER	2056

1) Autoencoders are effective models for collaborative filtering as they have no worse performance than basic methods.

2) Pretraining with RBMs do make autoencoders get better local optimum as the results improve a lot.

3) Stacked autoencoders are superior, but not enough with small rating dataset alone which do not have enough high-order information.

4) Prediction quality of autoencoders remains consistent when the amount of training data increases by an order of magnitude, which is a good indication for potential practical applicability.

5) The results of autoencoders can be merged with other methods to get a higher recall without reducing precision. It is good news for that we usually combine different models in real circumstances but not use single model.

5 Conclusion and Future Work

In this paper we proposed a revised autoencoder models for collaborative filtering. To acquire better performance, we tried some improvements such as pretraining with RBM and stacking autoencoders together. Experimental study has been conducted on two commonly used datasets to prove that those models are effective and can be integrated with other CF models to get a higher recall.

There are several extensions to be considered. First our current models focus on modelling the correlation between item ratings. We can generate a similar

model focusing on user ratings and then combine them together to get a better performance. Besides, we can introduce some content-based features into the model so that deep models may acquire more high-order information.

Acknowledgements. This work was partially supported by the National Natural Science Foundation of China (No. 61103095), the International S&T Cooperation Program of China (No. 2010DFB13350), and the Fundamental Research Funds for the Central Universities. We are grateful to Shenzhen Key Laboratory of Data Vitalization (Smart City) for supporting this research.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering 17(6), 734–749 (2005)
2. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al.: Greedy layer-wise training of deep networks. Advances in Neural Information Processing Systems 19, 153 (2007)
3. Breese, J.S., Heckerman, D., Kadie, C.M.: Empirical analysis of predictive algorithms for collaborative filtering. In: Proceedings of 14th International Conference on Uncertainty in Artificial Intelligence, pp. 43–52 (1998)
4. Deng, L.: An overview of deep-structured learning for information processing. In: Proceedings of Asian-Pacific Signal and Information Processing–Annual Summit and Conference (2011)
5. Georgiev, K., Nakov, P.: A non-iid framework for collaborative filtering with restricted boltzmann machines. In: Proceedings of the 30th International Conference on Machine Learning, pp. 1148–1156 (2013)
6. Goldberg, K., Roeder, T., Gupta, D., Perkins, C.: Eigentaste: A constant time collaborative filtering algorithm. Information Retrieval 4(2), 133–151 (2001)
7. Gunawardana, A., Meek, C.: Tied boltzmann machines for cold start recommendations. In: Proceedings of the 2008 ACM Conference on Recommender Systems, pp. 19–26. ACM (2008)
8. Hofmann, T.: Latent semantic models for collaborative filtering. ACM Transactions on Information Systems 22(1), 89–115 (2004)
9. van den Oord, A., Dieleman, S., Schrauwen, B.: Deep content-based music recommendation. In: Advances in Neural Information Processing Systems, pp. 2643–2651 (2013)
10. Phung, D.Q., Venkatesh, S., et al.: Ordinal boltzmann machines for collaborative filtering. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, pp. 548–556. AUAI Press (2009)
11. Salakhutdinov, R., Hinton, G.: An efficient learning procedure for deep boltzmann machines. Neural Computation 24(8), 1967–2006 (2012)
12. Salakhutdinov, R., Mnih, A., Hinton, G.: Restricted boltzmann machines for collaborative filtering. In: Proceedings of the 24th International Conference on Machine Learning, pp. 791–798. ACM (2007)