Collaborative Deep Learning with Knowledge Base Embedding for Recommender Systems

Tianyu Zhong[*], Zhewei Song[†]
College of Engineering
Northeastern University
Boston, MA 02115
{zhong.t, song.zhew}@husky.neu.edu

Abstract

In recent years, deep neural networks have yielded immense success on speech recognition, computer vision and natural language processing. However, the exploration of deep neural networks on recommender systems has received relatively less scrutiny. In this work, we strive to develop techniques based on neural networks to tackle the key problem in recommendation on the basis of implicit feedback. Although recommendation algorithms such as matrix factorization has been implemented widely and successfully, cold start problem is still a hard problem. Several methods such as popular ranking, similar items are used. In this article, we present a algorithm based on deep neural network in order to associate information from users, items and rating history. Cold start problem can be solved if user information is available. The experiments shows that recommendation via deep neural network have 13% better performance than tradition CF and MF.

1 Introduction

A recommender system can be viewed as a search ranking system, where the input query is a set of user and contextual information, and the output is a ranked list of items. Given a query, the recommendation task is to find the relevant items in a database and then rank the items based on certain objectives, such as clicks or purchases.

One challenge in recommender systems, similar to the general search ranking problem, is to achieve both memorization and generalization[1]. Memorization can be loosely defined as learning the frequent co-occurrence of items or features and exploiting the correlation available in the historical data. Generalization, on the other hand, is based on transitivity of correlation and explores new feature combinations that have never or rarely occurred in the past. In this project, we focus on the movies recommendation problem using MovieLens dataset, but the approach should apply to generic recommender systems.

For massive-scale online recommendation systems in an industrial setting, user/item-based models such as collaborative filtering(CF) are widely used because they are simple, scalable and interpretable. The models are often trained on binarized sparse features with one-hot encoding. Memorization can be achieved effectively using cross-product transformations over sparse features. Generalization can be added by using features that are less granular, But manual feature engineering is often required. One limitation of cross-product transformations is that they do not generalize to query-item feature pairs that have not appeared in the training data.

[*]NUID: 001259986
[†]NUID: 001811455

Preprint. Work in progress.

Embedding-based models, such as factorization machines or deep neural networks, can generalize to previously unseen query-item feature pairs by learning a low-dimensional dense embedding vector for each query and item feature, with less burden of feature engineering. Dense embeddings will lead to nonzero predictions for all query-item pairs, and thus can over-generalize and make less relevant recommendations.

Since a knowledge base provides rich information including both structured and unstructured data with different semantics, the usage of the knowledge base within the context of hybrid recommender systems are attraction increasing attention. However, previous studies have not fully exploited the potential of the knowledge base since they suffer from the following limitations[2]: 1) only utilize the single network structure information of the knowledge base while ignore other important signals such as items' textual and visual information. 2) rely on heavy and tedious feature engineering process to extract features from the knowledge base.

To address the above issues, in this project, we present the Wide & Deep learning framework to achieve both memorization and generalization in one model, by jointly training a linear model component and a neural network component. We propose a novel recommendation framework to integrate collaborative filtering with items' different semantic representations from the knowledge base. For a knowledge base, except for the network structure information, we also consider items' textual content (e.g., movie's titles). To be specific, we first apply a network embedding approach to extract items' structural representations by considering the heterogeneity of both nodes and relationships.

2 Related Work

The idea of combining wide linear models with cross-product feature transformations and deep neural networks with dense embeddings is inspired by previous work, such as factorization machines[3] which add generalization to linear models by factorizing the interactions between two variables as a dot product between two low-dimensional embedding vectors. In this project, we expanded the model capacity by learning highly nonlinear interactions between embeddings via neural networks instead of dot products.

In the recommender systems literature, collaborative deep learning has been explored by coupling deep learning for content information and CF for the ratings matrix[4]. There has also been previous work on mobile app recommender systems, such as AppJoy which used CF on users' app usage records[5]. Different from the CF-based or content-based approaches in the previous work, we jointly train Wide & Deep models on user and item data for movie recommender systems.

For textual knowledge embedding model, we all know that deep learning models have achieved remarkable results in computer vision[6] and speech recognition[7] in recent years. Within natural language processing, much of the work with deep learning methods has involved learning word vector representations through neural language models[8] and performing composition over the learned word vectors for classification. Word vectors, wherein words are projected from a sparse, 1-of-V encoding(here V is the vocabulary size) onto a lower dimensional vector space via a hidden layer, are essentially feature extractors that encode semantic features of word in their dimensions. In such dense representations, semantically close words are likewise close—— in Euclidean or cosine distance——in the lower dimensional vector space.

Convolutional neural networks(CNN) utilize layers with convolving filters that are applied to local features[9]. Originally invented for computer vision, CNN models have subsequently been shown to be effective for NLP and have achieved excellent results in semantic parsing[10], search query retrieval[11], sentence modeling[12], and other traditional NLP tasks[13].

In this project, we train a simple CNN with one layer of convolution on the top of word vectors obtained from embedding movie title words We initially keep the word vectors static and learn only the other parameters of the model. Despite little tuning of hyperparameters, this simple model achieves excellent results on multiple benchmarks.

Some applications used method based on reinforcement learning such as bandit. This algorithm detect preference of new users and reward the algorithm when getting positive feedback. As training continuing, the algorithm will find what users prefer, and give users good recommendation.[14]

3 Recommendation via Deep Neural Networks

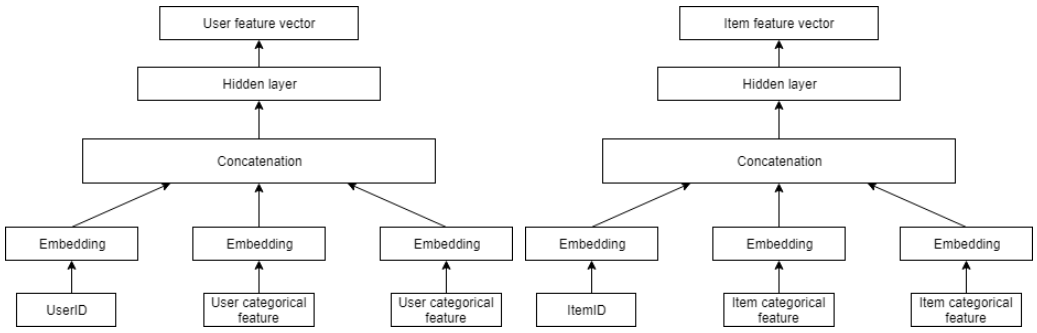
To permit a full neural treatment if collaborative filtering, we adopt a multi-layer representation to model a user-item interaction $y_u i$ as shown in Figure1, where the output of one layer serves as the input of the next one. The bottom input layer consists of two feature vectors v_u^U and v_i^I that describe user u and item i, respectively; Since this work focus on the pure collaborative filtering setting, we use only the identity of a user and item as the input feature, transforming it to a binarized sparse vector with one-hot encoding. Note that with such a generic feature representation for inputs, our method can be easily adjusted to address the cold-start problem by using content features to represent users and items.

3.1 User and Item Feature Extraction

Above the input layer is the embedding layer; It is a fully connected layer that projects the sparse representation to a dense vector. The obtained user(item embedding can be seen as the latent vector for user(item) in the context of latent factor model. The embedding vectors are initialized randomly and then the values are trained to minimize the final loss function during model training. These low-dimensional dense embedding vectors are then fed into the hidden layers of a neural network in the forward pass. Specifically, each hidden layer performs the following computation:

$$a^{(l+1)} = f(W^{(l)}a^{(l)} + b^{(l)}) \tag{1}$$

where l is the layer number and f is the activation function, often rectified linear units(Relus). $a^{(l)}$,$b^{(l)}$, and $W^{(l)}$ are the activations , bias and model weights at l-th layer.



(a) User feature embedding (b) Item feature embedding

Figure 1: Embedding and extraction of User and item feature

3.2 Text Feature Extraction

For an item entity such as movie in the knowledge base, we use the textual summary to represent the textual knowledge, which usually gives the main topics of this movie. The model architecture[15], shown in figure 2, is a slight variant of Collobert[13]. Let $x_i \in R^k$be the k-dimensional word vector corresponding to the i-th word in the sentence. A sentence of length n(padded where necessary) is represented as:

$$x_{1:n} = x_1 \oplus x_2 \oplus \cdots \oplus x_n \tag{2}$$

where $\oplus$ is the concatenation operator. In general, let $x_{i:i+j}$ refer to the concatenation of words $x_i, x_{i+1}, \ldots.., x_{i+j}$ A convolution operation involves a filter $w \in R^{hk}$ which is applied to a window of h words to produce a new feature. For example, a feature c_i is generated from a window of words $x_{i:i+h-1}$ by:

$$c_i = f(w^T x_{i:i+h-1} + b) \tag{3}$$

this filter is applied to each possible window of words in the sentence$\{x_{1:h}, x_{2:h+1}, \ldots.x_{n-h+1:n}\}$ to produce a feature map: $c = [c_1, c_2, \ldots.., c_{n-h+1}]$ with $c \in \mathbb{R}^{n-h+1}$. We then apply a max-over-time pooling operation over the feature map and take the maximum value $\hat{c} = max\{c\}$ as the feature corresponding to this particular filter. The idea is to capture the most important feature —one with

the highest value—for each feature map. This pooling scheme naturally deals with variable sentence lengths.

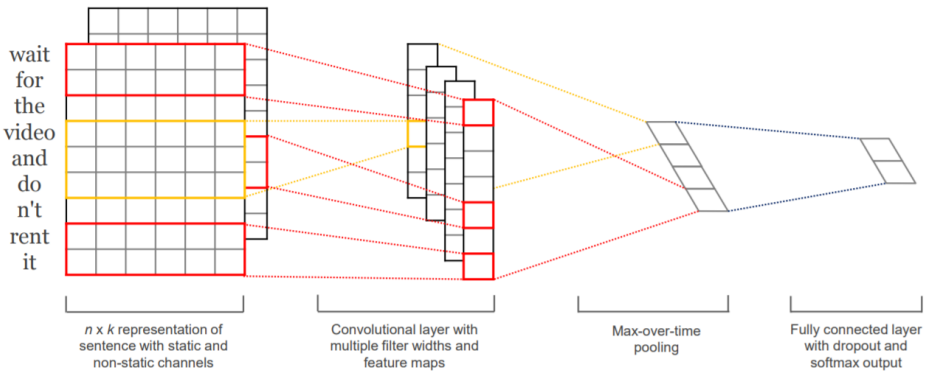


Figure 2: Embedding and extraction of User and item feature

3.3 Find the parameters of neural nets

To learn model parameters, existing methods perform a regression with squared loss:

$$L_{sqr} = \sum_{(u,i)} (y_{ui} - \hat{y})^2 \tag{4}$$

two deep components are combined using a cross inner product of their output as the prediction, which is then fed to one common loss function for ensemble. In an ensemble, individual models are trained separately without knowing each other, and their predictions are combined only at inference time but not at training time. For an ensemble, since the training is disjoint, each individual model size usually needs to be larger to achieve reasonable accuracy for an ensemble to work.

Ensemble of two deep models is done by back-propagating the gradients from the output to both deep parts of the model simultaneously using mini-batch stochastic optimization as shown in figure3.

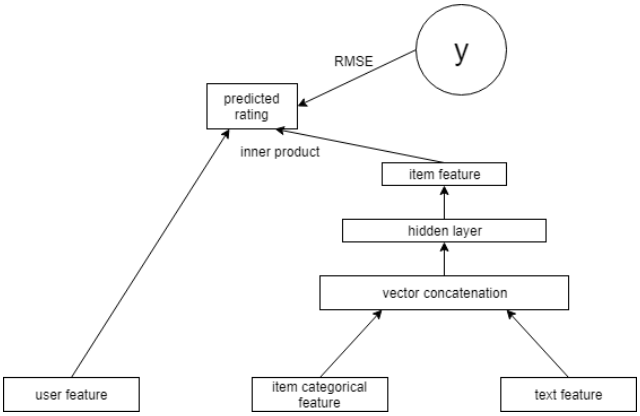


Figure 3: Regression for rating prediction

4 Experiment

4.1 Dataset

We used MovieLens [3] dataset, which included millions of rating records for 6040 users and more than 3706 movies. 1000209 records are available, where each user has at least 20 ratings. The profiles of users and movies are also included in the dataset. The user information includes their gender, age, and job. Age is divided into 7 ranges, and occupation is divided into 20 types. The movie profile contains title and genres of movies. 18 genres are available, and movies may have several genres at the same time. In the rating data, features contains userID, itemID and rating. The range of rating is 1-5. Using these profile data, we tried to extract them and build a DNN for rating prediction.

Table 1: Dataset introduction

Dataset	MovieLens
Interaction #	1000209
User #	3706
Item #	6040
Sparsity %	95.53%

4.2 Evaluation Metrics

We will use rooted mean square error for evaluation metrics(RMSE).

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{r}_i - r_i)^2} \tag{5}$$

where $\hat{r}$ is the rating prediction, and r is the ground true rating in the dataset.

4.3 Baseline Method

We have tried baseline methods including user-based collaborate filtering, item-based collaborate filtering and matrix factorization based on funkSVD[16]. We have experimented with different K and λ(regularizer), and got the best RMSE on test data. The results are listed in table 2.

These results show that funkSVD model has the smallest RMSE of 0.9, and item-based CF has the most RMSE of 1.005. Matrix factorization is more powerful than CF when dealing with rating prediction problem.

Table 2: Baseline performance

Model	**RMSE**
Item-based CF	1.005
User-based CF	0.947
FunkSVD	0.901

However, methods above just consider the interaction between users and items, but they ignored the content information from users and items. Thus, bias may be produced in the rating prediction. Also, cold start problem is not easy for these model to deal with. They cannot detect any useful information from new users and items.

4.4 Experiment, Result and Discussion

First, we preprocessed and cleaned the data so that it can be loaded by model. Second, we built the fundamental structure of neural networks including embedding and hidden layers. Finally, after getting the feature vectors of users and items, we tried to minimize the RMSE according to rating in the training set. Also, we tuned the structure and hyperparameters of the model in order to reach the best performance.

[3] https://grouplens.org/datasets/movielens

4.4.1 Configuration

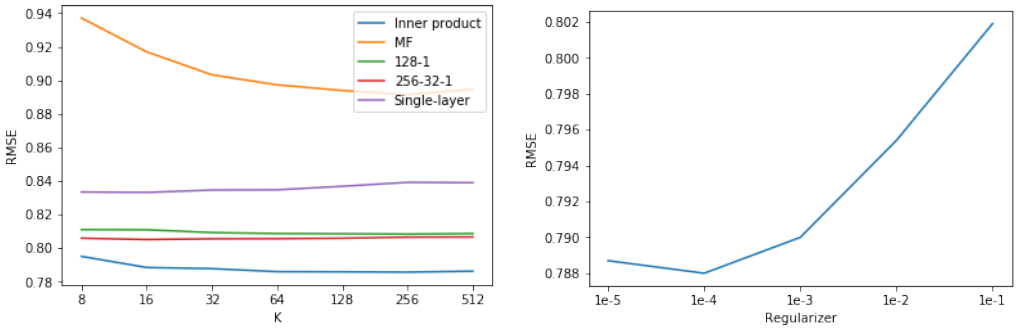
In our experiment, we implemented the model with Python 3.6 and Tensorflow 1.7.0, which is a great deep learning toolkit contributed by Google. We used 5-fold cross validation, which means test set contains 20% of all training set. We regarded the average RMSE of these 5-fold test sets as the final test error.

In the text CNN, the length of title varies. Thus, padding should be used for uniforming the length of input vector. We set the padding size 18 because all titles are not longer than 18 words. Then, a uniform size of matrix can be input into the text CNN. Moreover, we used 4 filters with size of 2, 3, 4 and 5, considering the context information of the sentences.

For the training session, ADAM optimizer is used to minimize the RMSE on training set, with learning rate 0.001[17]. This is because ADAM has great ability of finding the global minimum. The training process will not stop until validation error increases. At least 5 epochs will used for training models.

The hyperparameters that should be tuned are feature dimension K, embedding size, and the regularizer.

4.4.2 Results



(a) K v.s. RMSE

(b) Regularizer v.s. RMSE

Figure 4: Hyperparameters influences on RMSE

Figure 4(a) illustrates several structures and their RMSE respectively. The best MF only have RMSE of 0.90. However, the worst DNN model have RMSE of 0.84, which is nearly 7% performance improvement. In these DNN structures, inner product have the best performance with RMSE of 0.78. After using information of users and items, the performance of model does improve much. Pairwise algorithms, such as MF and CF, only use behaviors of users, and ignore the information from both users and items. As K increases, models of DNN are not influenced, and remain the same RMSE. However, performances of MF and inner product structure are influenced. The RMSE drops at the beginning of K increasing, and keep a relatively low error with a large K. When K is very large, the RMSE raises a little bit.

Figure 4(b) illustrates the RMSE variation as regularizer changes with structure of inner product. In our algorithm, fully-connected layers are used for feature extraction and explanation. Overfitting is an essential problem because fully-connected neural network is likely to overfit on test data. A proper regularizer will prevent overfitting of model, and make the algorithm robust. We can see the best regularizer is 0.0001 with the lowest RMSE of 0.788.

As the result shows, the structure of the network influences the performance. We have two kinds of structures for candidate, inner product and fully-connected layers. Comparison should be done for better structure.

Figure 5 illustrates our experiment results of our model. With different feature vector dimensions K and embedding sizes, we train models with several structures in order to find the best.

We can see that in all 5 charts, structure with inner product always has the best performance. This is also the same as our result in figure 4. The best embedding size is 32. As number of layers increases,

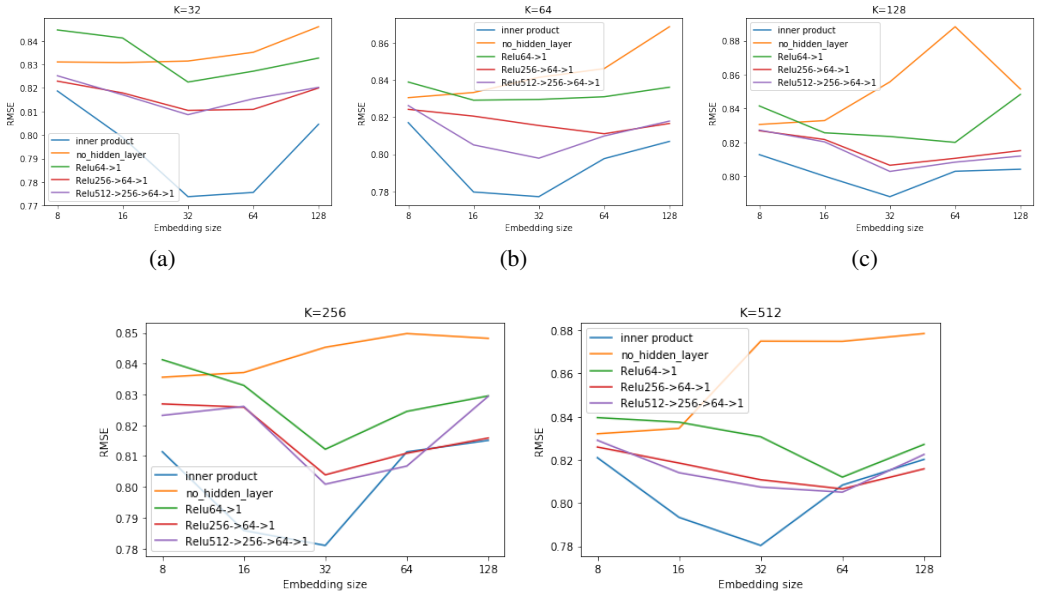


Figure 5: Structure v.s RMSE

the performance also improves. However, the efficiency of deeper structure does not increase, that is, as the number of layers grows, the enhancement of performance reduces. In our results, the performance increases clearly when the number of layer increases from 0 to 1 and 2. However, the performances are nearly the same between 3-layer and 2 layer structure. Thus, it is the truth that the deeper, the better. Here, "better" refers to the performance. But it depends if considering the efficiency of complexity growth.

4.4.3 Why inner product better than vector concatenation?

From the result of figure 4 and 5, we can conclude that inner product is far more powerful than vector concatenation. Why does it work much better? Actually, simple vector concatenation puts feature vectors of user and item together, and then inputs it into the network. The information of latent variables is not used for prediction. On the contrary, inner product will identify latent clusters inside the feature vectors, and detect the interaction among dimensions. Clustering is automatically done after inner product manipulation[18]. At the same time, the value of feature dimension influences the prediction accuracy. As the result, K is a sensitive hyperparameter when K is not so large(less than 64). Thus, when facing pairwise prediction or recommendation, inner product will have better understanding of mutual influence, and make better prediction results.

4.5 Recommendation Functions

We implemented several functions to recommend items to users, and find similar items. After training the model, feature matrices of users and items are saved. Based on these matrices, recommendation can be done without long-process calculation.[4]

4.5.1 Rating Prediction

According to our model, rating is predicted by inner product of user feature and item vector. Thus, the rating can be predicted as

$$\hat{r}_i = u_i^T v_i \tag{6}$$

where u_i^T refers to user feature, and v_i refers to item feature.

[4]https://github.com/zty8718137/recommend_via_neural_network

4.5.2 Similar Items

Similar item searching is actually to find the nearest neighbor in a vector space. We find neighbors based on cosine similarity

$$Sim = \frac{v_i^T v_j}{|v_i||v_j|} \tag{7}$$

20 of similar items will be found in our recommendation result.

4.5.3 Recommendation for Users

We will get users' history of their rating. Then, the predicted rating for each items will be calculated.

$$Rate = u_i^T V \tag{8}$$

In the items this user has not rated, system will give 20 items as recommendation.

4.5.4 Also-like Items

This function will recommend some items based on users' preference. Starting from a target item, based on users' history and behavior, preferred items of users who like the target item will be recommended.

The strategy is that, based on the target, users who give high rating will be found. Then, according to their rating history, find items they prefer and put them into a candidate list. Finally, give recommendation to the target item. Random selection are utilized to prevent duplication.

5 Conclusion

In this work, we introduced collaborative deep learning with knowledge base embedding for recommender systems. We used embedding layers for encoding categorical features to prevent curse of dimensionality. It is proved by our experiment that using content information have 13% better performance than normal matrix factorization. Neural networks give great ability of explanation to the recommender system. Inspired by matrix factorization, inner product is used to associate both user and item features. This strategy can predict users' preference in higher accuracy. Moreover, cold start problem can be solved by our algorithm, because we have found association among users, items and history.

Also, a lot more work should be done. Our work focused on rating prediction. Now in industrial application, top K recommendation is a popular task on personal information distribution. In the future, we will focus on top K recommendation and combine it with neural network application. Moreover, click-through rate(CTR) has been used for recommendation quality evaluation[19, 20, 1]. This task focus on personal recommendation for users. We plan to combine the advantages of these algorithms to make the quality better.

References

[1] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10. ACM, 2016.

[2] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 353–362. ACM, 2016.

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

[4] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1244. ACM, 2015.

[5] Bo Yan and Guanling Chen. Appjoy: personalized mobile application discovery. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 113–126. ACM, 2011.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[7] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[8] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Wen-tau Yih, Xiaodong He, and Christopher Meek. Semantic parsing for single-relation question answering. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 643–648, 2014.

[11] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[12] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 373–374. ACM, 2014.

[13] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[14] Shuai Li, Alexandros Karatzoglou, and Claudio Gentile. Collaborative filtering bandits. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 539–548. ACM, 2016.

[15] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[16] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. 2007.

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

[18] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[19] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[20] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 43–50. ACM, 2016.