Improved Recurrent Neural Networks for Session-based Recommendations

Article · June 2016

CITATIONS
7

READS
225

3 authors, including:

Yong Kiam Tan
Carnegie Mellon University
9 PUBLICATIONS **27** CITATIONS
SEE PROFILE

Xinxing Xu
Agency for Science, Technology and Research (…
15 PUBLICATIONS **210** CITATIONS
SEE PROFILE

Improved Recurrent Neural Networks for Session-based Recommendations

Yong Kiam Tan
tanyongkiam@gmail.com

Xinxing Xu
xuxinx@ihpc.a-star.edu.sg

Yong Liu
liuyong@ihpc.a-star.edu.sg

Institute of High Performance Computing, A*STAR
1 Fusionopolis Way, Singapore 138632

ABSTRACT

Recurrent neural networks (RNNs) were recently proposed for the session-based recommendation task. The models showed promising improvements over traditional recommendation approaches. In this work, we further study RNN-based models for session-based recommendations. We propose the application of two techniques to improve model performance, namely, data augmentation, and a method to account for shifts in the input data distribution. We also empirically study the use of generalised distillation, and a novel alternative model that directly predicts item embeddings. Experiments on the RecSys Challenge 2015 dataset demonstrate relative improvements of 12.8% and 14.8% over previously reported results on the Recall@20 and Mean Reciprocal Rank@20 metrics respectively.

CCS Concepts

•Computing methodologies → Supervised learning; Neural networks; •Information systems → Recommender systems;

Keywords

Recurrent neural networks; Recommender systems; Session-based recommendations

1. INTRODUCTION

Users of e-commerce websites are often inundated by the huge number of items available for sale. Recommender systems can be used to enhance user experience by making personalized and useful recommendations for each user. For example, the system could automatically display items of interest, or suggest new discounts relevant to each user. In order to personalize recommendations, traditional recommender systems often need to build up a user profile. Collaborative filtering approaches [15, 14, 21] can define user-user similarity based on their history of purchases, or they could rely on matrix factorization to build latent factor vectors for each user. Crucially, these approaches require the user to be identified when making recommendations. This may not always be possible: new users to the site will not have any profile, or users may not be logged in, or they may have deleted their tracking information. This leads to the problem of cold-start for recommendation methods that require user history.

An alternative to relying on historical data is to make session-based recommendations [23]. In this setting, the recommender system makes recommendations based only on the behaviour of users in the current browsing session. This avoids the aforementioned cold-start issue but we must ensure that the system remains accurate and responsive (i.e. the predictions do not take too long to make). Recurrent Neural Networks (RNNs) were recently proposed in [10] for the session-based recommendation task. The authors showed significant improvements over traditional session-based recommendation models using an RNN. The proposed model utilizes session-parallel mini-batch training, and also employs ranking-based loss functions for learning the model.

In this work, we further study the application of RNNs for session-based recommendations. In particular, we examine and adapt various techniques from the literature for this task. These include:

- Data augmentation via sequence preprocessing and embedding dropout to enhance training and reduce overfitting.
- Model pre-training to account for temporal shifts in the data distribution.
- Distillation using privileged information to learn from small datasets.

Additionally, we propose a novel alternative model that reduces the time and space requirements for predictions by predicting item embeddings directly. This makes RNNs more readily deployable in real-time settings.

Our proposed techniques were evaluated on the RecSys Challenge 2015 data set. The effectiveness of our data augmentation strategy is evidenced by relative model performance improvements of 12.8% and 14.8% over previously reported results on the Recall@20 and Mean Reciprocal Rank@20 (MRR@20) metrics respectively. We also showed that distillation could be successfully applied for performance gains on small datasets. Finally, our novel item embedding output approach significantly reduces the time and space requirements of the RNN model.

ACM ISBN XXX-XXXX-XX-XXX/XX/XX.
DOI: XX.XXX/XXX_X

We start with a discussion of related work in Section 2. Then, we present the details of our improved RNN models in Section 3, and our experiments on the models in Section 4.

2. RELATED WORK

Matrix factorization and neighbourhood-based methods are widely utilized for recommender systems in the literature. Matrix factorization methods [15, 30] are based on the sparse user-item interaction matrix, where the recommendation problem is formulated as a matrix completion task. After decomposing the matrix, each user and item is represented by a latent factor vector. The missing value of the user-item matrix can then be filled by multiplying the appropriate user and item vectors. Since this requires us to identify both the user and item vectors, matrix factorization methods are not directly suitable for session-based recommendations where the users are unknown. One way to solve this cold-start problem is to use pairwise preference regression [20]. Neighbourhood based methods [22, 14] utilize the similarities between item and user purchase history; they can be applied to session-based recommendations by comparing session similarity.

Deep learning has recently been applied very successfully in areas such as image recognition [16, 9], speech recognition [8, 1] and natural language processing [24]. Deep models can be trained to learn discriminative representmations from unstructured data such as images and speech signals. They have also been used for collaborative filtering [29, 21]. In [10], RNNs were proposed for session-based recommendations. The authors compared RNNs (with several customized ranking losses) to existing methods for session-based predictions and found that RNN-based models performed 20% to 30% better than the baselines. Our work is closely related, and we study extensions to their RNN models. In [31], the authors also use RNNs for click sequence prediction; they consider historical user behaviours as well as hand engineered features for each user and item. In this work, we rely entirely on automatically learned feature representations.

Many approaches have been proposed to improve the predictive performance of trained deep neural networks. Popular approaches include data augmentation [16], dropout [25, 6], batch normalization [12] and residual connections [9]. We seek to apply some of these methods to enhance the training of our recommendation RNNs.

The learning using privileged information (LUPI) framework [28, 27] was proposed to utilize the additional feature representations that are only available during training but not during testing. When there is a limited amount of training data, the use of such information has been found to be helpful [28]. In the generalized distillation approach [7], a student model learns from soft labels provided by a teacher model. If we train the teacher model on the privileged dataset, then this approach can be applied to LUPI. In this work, we propose the use of this framework for the click sequence prediction by using the future portion of each click sequence as a form of privileged information.

3. PROPOSED APPROACHES

In this section, we explain the use of RNNs for the session-based recommendation problem (3.1). This is followed by our proposed data augmentation methods (3.2), our approach to handling temporal shifts (3.3), an explanation of the application of LUPI (3.4), and finally, an alternative model based on embeddings to trade model accuracy for speed and memory requirements (3.5).

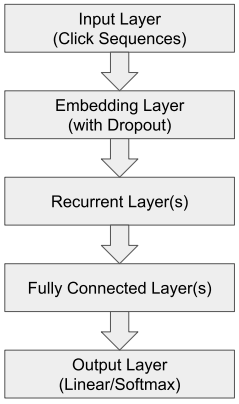

Figure 1: Generic structure of the network used in our models. The output layer can either use a softmax or linear activation function.

3.1 RNNs for session-based recommendations

The session-based recommendation problem can be formulated as a sequence-based prediction problem as follows. Let $[x_1, x_2, \ldots, x_{n-1}, x_n]$ be a click session, where $x_i \in \mathbb{R}$ $(1 \leq i \leq n)$ is the index of one clicked item out of a total number of m items.

We seek a model M such that for any given prefix click-sequence of the session, $\mathbf{x} = [x_1, x_2, \ldots, x_{r-1}, x_r]$, $1 \leq r < n$, we get the output $\mathbf{y} = M(\mathbf{x})$, where $\mathbf{y} = [y_1, \ldots, y_m]' \in \mathbb{R}^m$. We view $\mathbf{y}$ as a ranking over all the next items that can occur in that session, where y_i corresponds to the score of item i. Since we typically need to make more than one recommendation for the user to choose from, the top-k items (as ranked by $\mathbf{y}$) are recommended.

In most of our models, we use a classification-based output, where $\mathbf{y}$ corresponds to a probability distribution over the items. Let x_{r+1} be the next click of the click sequence $\mathbf{x}$; we can represent it with an m-dimensional 1-HOT encoded vector $V(x) \in \mathbb{R}^m$. The model can be tuned by minimizing a chosen loss function e.g. the cross entropy loss, $L(M(\mathbf{x}), V(x_{r+1}))$. Other outputs are possible: the models in [10] output ranking scores for each item, and they are trained with ranking losses.

We follow the generic structure of the RNN model shown in Figure 1. For the recurrent layers, we use the Gated Recurrent Unit (GRU) [4] as it was found in [10] that they outperformed the Long-term Short Memory (LSTM) [11] units. However, we do not utilize the stateful RNN training procedure, where the models are trained in a session-parallel, sequence-to-sequence manner. Instead, our networks process each sequence $[x_1, x_2, \ldots, x_r]$ separately, and are trained to predict the next item, x_{r+1}, in that sequence. We also represent all our input using trainable embeddings. Our networks can be trained using standard mini-batch gradient descent on the cross-entropy loss via Backpropagation-Through-Time (BPTT) for a fixed number of time steps. This training procedure is visualized in Figure 2.

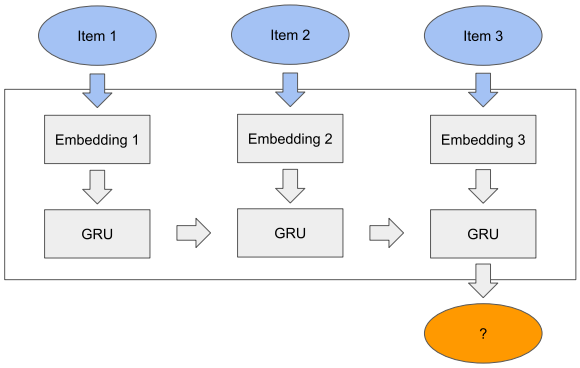

Figure 2: Training procedure for a single sequence in our RNN. Gradients are backpropagated along the grey arrows.

3.2 Data augmentation

Click sessions often vary in length: some users may take a long time before finding their desired item, while others find it with just a few clicks. One aim of the recommender system should be to provide accurate predictions regardless of the current session length. Data augmentation techniques have been widely used to enhance image-based models [16]. Here, we propose two methods to augment click sequences.

The first is an application of the sequence preprocessing method proposed in [5]. All prefixes of the original input sessions are treated as new training sequences. Given an input training session $[x_1, x_2, \ldots, x_n]$, we generate the sequences and corresponding labels $([x_1], V(x_2))$, $([x_1, x_2], V(x_3))$, $\ldots$, $([x_1, x_2, \ldots, x_{n-1}], V(x_n))$ for training.

Embedding dropout is a form of regularization applied to input sequences [6]. Applying it to a click sequence is equivalent to a preprocessing step that randomly deletes clicks at random. Intuitively this makes our model less sensitive to noisy clicks, e.g. where users may have accidentally clicked on items that are not of interest. Hence, it makes the model less likely to over-fit to specific noisy sequences. It can also be viewed as a form of data augmentation, where shorter, pruned sequences are being generated for model training.

We apply both methods to all our models, and a graphical example is shown in Figure 3. Note that different clicks are dropped in each sequence for every training epoch.

3.3 Adapting to temporal changes

A key assumption of many machine learning models is that the input is independent and identically distributed. This is not strictly true in the item recommendation setting since new products will only appear in sessions collected after that product is released, and user behaviour/preferences may also shift over time. Moreover, the purpose of the recommender system is to make prediction on new sequences, i.e. those arising from recent user behaviours. Learning a recommendation model on the entire dataset may, therefore, lead to worse performance because the model ends up focusing on some out-of-date properties that are irrelevant to the latest sequences. One way to handle this is to define a temporal threshold, and discard click sequences that are older than the threshold when building the model. However, this reduces the amount of training data available for our models to learn from.

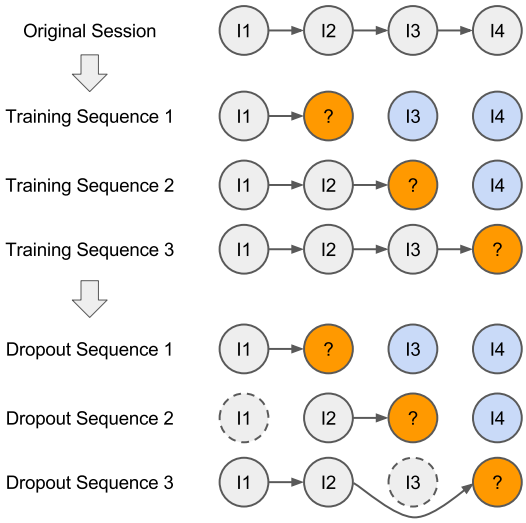

Figure 3: An example application of our preprocessing step on a session with four clicks. The items in orange are output labels corresponding to their respective training sequences (in grey), and the items with a dotted outline are randomly dropped during training. The privileged information for each preprocessed sequence is coloured in blue, they are not used in the standard training procedure.

We propose a simple solution to get the best of both worlds via pre-training. We first train a model on the entire dataset. The trained model is then used to initialize a new model, which is only trained using only a more recent subset of the data, e.g. the last month worth of data out of a year of click sequences. This allows the model to have the benefit of a good initialization using large amounts of data, and yet is focused on more recent click-sequences. In this way, it resembles the fine-tuning process used in training of image-based networks [2], where the models are typically initialized by pre-training on ImageNet (a large image classification dataset) before the weights are fine-tuned on a smaller image dataset in the desired domain.

3.4 Use of privileged information

The item sequence clicked by users *after* an item may also contain information about that item (highlighted in Figure 3). This information cannot be used for making predictions since we cannot view the future sequences when making recommendations. We can, however, utilize these future sequences as *privileged information* [28] in order to provide soft labels for regularizing and training our models. We use the generalized distillation framework [17] for this purpose.

Formally, given a sequence $[x_1, x_2, \ldots, x_r]$ with label x_{r+1} from a session, we define the privileged sequence as $\mathbf{x}^* = [x_n, x_{n-1}, \ldots, x_{r+2}]$ where n is the length of the original session before our preprocessing. The privileged sequence is simply the reversed, future sequence that occurs after the r^{th} item. We can now train a teacher model on the privileged sequences $\mathbf{x}^*$, with the same label, x_{r+1}.

Next, we tune our student model $M(\mathbf{x})$ by minimizing a

loss function[1] of the form: $(1-\lambda) * L(M(\mathbf{x}), V(x_n)) + \lambda * L(M(\mathbf{x}), M^*(\mathbf{x}^*))$, where $\lambda \in [0,1]$ is a tradeoff parameter between the two sets of labels. This allows M to learn from both the real labels, as well as the labels predicted by its teacher, M^*. This learning procedure is useful when the amount of training data available is small, which may be the case for a new, small scale website.

3.5 Output embeddings for faster predictions

An issue with the models we have described thus far is the size of the output layer. The output layer[2] is typically fully connected to the previous hidden layer – this means that the number of parameters to be tuned in these two layers alone is $H * N$ where H is the number of nodes in the hidden layer and N is the number of candidate items for prediction. Besides the memory requirements, this also makes prediction slower since the model has to perform an additional large matrix multiplication.

A similar problem has also been studied in natural language processing, where the output vocabulary can be huge. Typical approaches include the use of a hierarchical softmax layer [19], and sampling only the most frequent items. The hierarchical softmax approach does not apply directly in our case, since we are required to make a top-k prediction, rather than just a top-1 prediction.

We instead view item embeddings as a projection of the items from a 1-HOT encoded space of dimension N onto a lower dimensional space. Using this point of view, we propose to train the model to predict the *embedding* of the next item directly. The model is tuned using the cosine loss between the embedding of the true output and the predicted embedding. This approach is inspired by the distributed representations of words [18], where similar words have embeddings that are closer in cosine distance. We expect, similarly, that the items which a user is likely to click after a given sequence should be close in the item embedding space. Using this type of output reduces the number of parameters in the final layers to $H * D$, where D is the dimensionality of the embedding. A drawback of this approach is that it requires a good quality embedding for each item. One way to obtain such an embedding is to extract and re-use the trained item embedding from the models described above.

4. EXPERIMENTS

We evaluate our proposed extensions to the basic RNN model on the RecSys Challenge 2015 dataset. The dataset is split following [10], where sessions in the last day are placed in the test set, and everything else is placed in the training set. This yields 7966257 sessions in the training set, 15234 sessions in the test set, and 37483 candidate items for prediction. We have 23670981 training sequences and 55898 test sequences after preprocessing the sessions. To better evaluate some of our models, e.g. privileged information and pre-training, we sort the training sequences by time and report our results on models trained on more recent fractions ($\frac{1}{256}, \frac{1}{64}, \frac{1}{16}, \frac{1}{4}, \frac{1}{1}$) of the training sequences as well.

The evaluation metrics used were Recall@20 and Mean Reciprocal Rank (MRR)@20; each of the test sequences, i.e. prefixes of sessions, is given equal weight [10]. The metrics are designed for the recommendation setting, as we usually want to make multiple recommendations for each user. For M1-M3, we take the top 20 most probable items directly from the softmax outputs. For M4, we compute the cosine distance of the model output against the embedding of items, and take the top 20 closest items[3]. Finally, we also report the model size and batch prediction times for each model. These are important considerations if the model is going to be deployed in a real recommender system.

[1]The original presentation also includes a temperature parameter T which can be used to control the softness of the softmax labels.

[2]Although we focus on the softmax activation function, this also applies for ranking-based outputs.

Model Type (GRU Size)	Recall@20	MRR@20
S-POP (-) [10]	0.2672	0.1775
Item-KNN (-) [10]	0.5065	0.2048
TOP1 (1000) [10]	0.6206	0.2693
BPR (1000) [10]	0.6322	0.2467
M4 (1000)	0.6676	0.2847
M2 (100)	**0.7129**	**0.3091**

Table 1: Best performing models in our experiments compared against various baselines.

4.1 Experimental setup

All our models used 50-dimensional embeddings for the items, with 25% embedding dropout. Optimization was done using Adam [13], with mini-batch size fixed at 512. We truncated BPTT at 19 time-steps since 99% of the original training sessions had lengths less than or equal to 19. The number of epochs was set by early stopping using 10% of the training data as the validation set for each model. We used one recurrent (GRU) layer in all our models as we found that additional layers did not improve performance. The GRU was set at 100 and 1000 hidden units for each model. The models are defined and trained in Keras [3] and Theano [26] on a GeForce GTX Titan Black GPU. The specifics of each model (along with their labels) are as follows:

M1 The RNN model with softmax outputs, sequence preprocessing and embedding dropout. The recurrent layer is fully connected to the output layer.

M2 M1, but additionally re-tuned on more recent fractions of the training dataset.

M3 An M1 model trained on the privileged information (future sequences) available in each data fraction. This is used to provide soft labels for another M1 model with parameters $T = 1$ and $\lambda = 0.2$. We did not extensively tune these parameters.

M4 The output of this model predicts an item embedding directly. We added a fully connected hidden layer between the recurrent and output layers as we found that this improved the model's stability. We used the embeddings trained on the full training dataset in M1 for these models.

B This refers to the best results reported in [10].

4.2 Experimental results

[3]This can be efficiently computed on the GPU as an additional Theano expression.

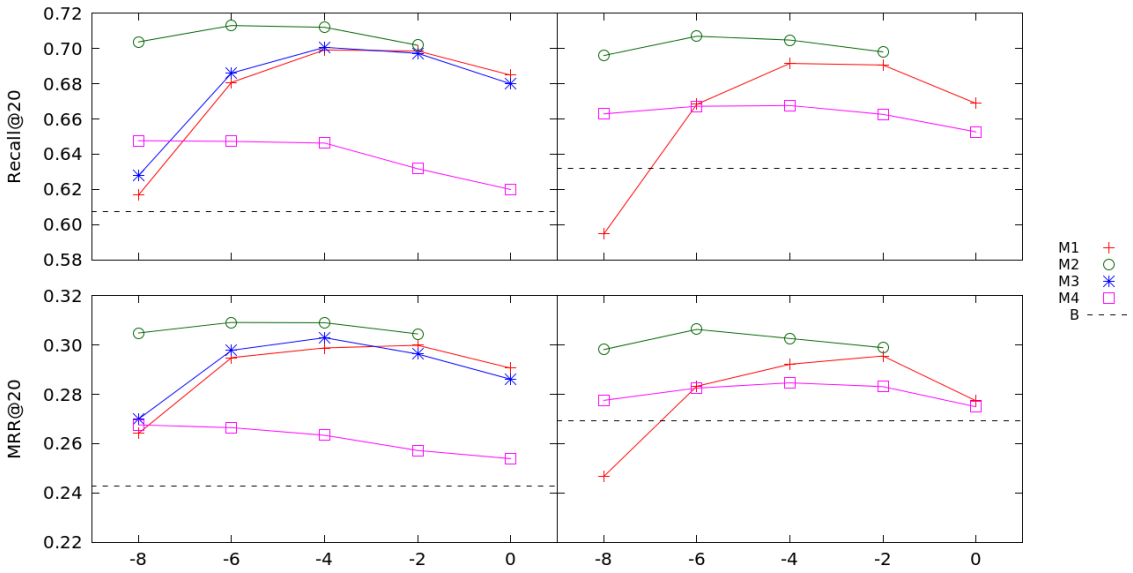

Figure 4: Plots of both evaluation metrics on models with GRU size 100 (left) and 1000 (right). The x-axis is logarithmic in dataset fraction, the rightmost point corresponds to the full dataset. M2 does not apply to the full dataset, and results for M3 on the larger GRU size were omitted.

M (GRU Size)	Prediction time (s)	Parameters
M1 - M3 (100)	0.665 (± 0.023)	5705384
M4 (100)	0.366 (± 0.022)	1950150
M1 - M3 (1000)	0.824 (± 0.025)	42548684
M4 (1000)	0.485 (± 0.022)	7133250

Table 2: Average batch prediction time in seconds and memory requirements for each proposed model (M) at the prediction phase. Prediction times for M4 includes computing the cosine distance of the predicted embedding against each item's embedding.

The performance of each model on the evaluation metrics is summarized in Figure 4. Overall, M1 and M2 yielded strong performance gains over the reported baseline RNN models. From the results of M1, we also see that training with the entire dataset yields slightly poorer results than training it on more recent fractions of the dataset. This indicates that our recommendation models do need to account for changing user behaviour over time. Our best performing models are reported in Table 1. We also list the baseline results reported in [10], including their best RNN based models (*i.e.*, TOP1 and BPR) and two traditional algorithms (*i.e.*, S-POP and Item-KNN). Surprisingly, moving from a GRU of 100 to GRU of 1000 did not significantly improve the performance of our models (M1-M3).

We found that the privileged information model (M3) takes an extremely long time to train; we omitted results for M3 with GRU size 1000 as it could not be trained in reasonable time. We believe the main reason for the drastic increase in training time was the need to (1) compute the soft labels, and (2) compute a corresponding cross-entropy loss against these labels for every mini-batch. This scales very poorly when the number of possible labels is large, as is the case here. Nevertheless, M3 yielded modest performance gains over M1 on the smallest dataset sizes. This is consistent with the use of privileged information in [17], and suggests that it might be useful in settings where little data is available.

Finally, M4 performs poorly compared to our other models in terms of predictive accuracy (although it still improves over the baseline). We may be able to further improve the accuracy in M4 if better quality embeddings were available as targets. We did not, for example, used any additional information of the items, e.g. category or brand, that will be available in an online store.

On the other hand, the batch prediction time and model sizes are shown in Table 2. Predictions can be made in M4 using only about 60% of the prediction time of classification-based models (M1-M3). M4 also has much fewer parameters, and therefore, requires less memory. Together, these are steps towards making RNN models deployable in real recommender systems.

5. CONCLUSION

We have presented, and empirically evaluated, several proposed extensions to a basic RNN model. We showed that it is possible to enhance the performance of recurrent models for session-based recommender systems by using proper data augmentation techniques, and accounting for temporal shifts in user behaviour. Directions for future work include exploring the tradeoffs of the embedding-based model, and using known features of the items in our the models.

6. REFERENCES

[1] Baidu Research. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015.

[2] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details:

Delving deep into convolutional nets. In *British Machine Vision Conference*, 2014.

[3] F. Chollet. Keras. https://github.com/fchollet/keras, 2015.

[4] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[5] A. de Brébisson, É. Simon, A. Auvolat, P. Vincent, and Y. Bengio. Artificial neural networks applied to taxi destination prediction. *CoRR*, abs/1508.00021, 2015.

[6] Y. Gal. A theoretically grounded application of dropout in recurrent neural networks. *CoRR*, abs/1512.05287, 2016.

[7] H. Geoffrey, V. Oriol, and D. Jeff. Distilling the knowledge in a neural network. *arXiv:1511.03643*, 2015.

[8] A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649, 2013.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[10] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. *CoRR*, abs/1511.06939, 2015.

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[12] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[14] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 426–434, 2008.

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

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012.

[17] D. Lopez-Paz, B. Schölkopf, L. Bottou, and V. Vapnik. Unifying distillation and privileged information. In *International Conference on Learning Representations*, 2016.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *27th Annual Conference on Neural Information Processing Systems*, pages 3111–3119, 2013.

[19] A. Mnih and G. E. Hinton. A scalable hierarchical distributed language model. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1081–1088. Curran Associates, Inc., 2009.

[20] S.-T. Park and W. Chu. Pairwise preference regression for cold-start recommendation. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 21–28, New York, NY, USA, 2009. ACM.

[21] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 791–798, New York, NY, USA, 2007. ACM.

[22] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 285–295, 2001.

[23] J. B. Schafer, J. Konstan, and J. Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 158–166, New York, NY, USA, 1999. ACM.

[24] R. Socher, C. C. Lin, A. Y. Ng, and C. D. Manning. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 2011.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[26] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[27] V. Vapnik and R. Izmailov. Learning using privileged information: Similarity control and knowledge transfer. *Journal of Machine Learning Research*, 16:2023–2049, 2015.

[28] V. Vapnik and A. Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5-6):544–557, 2009.

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

[30] M. Weimer, A. Karatzoglou, Q. V. Le, and A. J. Smola. Maximum margin matrix factorization for collaborative ranking. In *NIPS*, pages 1593–1600, 2007.

[31] Y. Zhang, H. Dai, C. Xu, J. Feng, T. Wang, J. Bian, B. Wang, and T.-Y. Liu. Sequential click prediction for sponsored search with recurrent neural networks. In C. E. Brodley and P. Stone, editors, *AAAI*, pages 1369–1375. AAAI Press, 2014.