ConvRec: A simple lightweight convolutional network for next item recommendation

Md Mehrab Tanjim
Computer Science & Engineering
University of California, San Diego, CA 92037
mtanjim@ucsd.edu

Abstract

Historical sequential data holds important information of a user's preference. By successfully modeling a user's past action, we can give effective next item recommendation. In past, various sequential neural networks (e.g. RNN, LSTM) have been deployed to model such sequential behaviors. Recently, self-attention models have become very popular and it has been shown it can outperform many state-of-the-art models in recommender system. However, the number of operations required by this approach is quadratic and parameter size also grows linearly with input, both of which may not be practical for long sequences (e.g. clicks). In this paper, we show that these amounts of computation and parameter required by self-attention model are not necessary and a very lightweight convolution can match its performance while requiring only linear number of operations in term of in the input length. In addition, number of parameters does not depend on input length which is crucial for long-range sequences. Extensive experiments on four benchmark datasets show that our model performs competitively with respect to state-of-the-art results of self-attention while requiring less memory and maintaining scalability.

1 Introduction

Everyday users interact with various items in different platforms. As users take various actions with items they leave a trail of their actions which forms a sequential behaviors. The goal of sequential recommender system is, therefore, to capture these sequential dynamics and predict the next item that he/she will most likely interact with. Capturing useful patterns from sequential dynamics is challenging, primarily because the dimension of the input space grows exponentially with the number of past actions used as context. Markov chains are a classical example which assumes next action is conditioned on only the previous one or more actions. It is very effective when it comes to capture short-range item transitions [1]. In advance of deep learning, Recurrent Neural Networks (RNNs) have become very popular for sequence modeling. It can capture the sequence dynamics by summarizing previous actions via a hidden state. Both models are strong in one case and weak in another. MC models while can model short-term dependencies, cannot capture long range relationships, while RNNs can perform well in denser datasets, may underperform in sparser (i.e. short sequences) datasets.

Recently, a new 'self-attention' model, namely Transformer, has been proposed which is purely based on attention mechanism and achieved new state-of-the-art performance for machine translation tasks [3]. Inspired by this method, self-attention model has been applied in task of next item recommendation, and it has been shown that it can draw context from all actions in the past (like RNNs) but on the other hand is able to frame predictions in

terms of just a small number of actions (like MCs) [49]. The proposed model significantly outperforms state-of-theart MC/CNN/RNN-based sequential recommendation methods on several benchmark datasets. It has also been shown that it is suitable for parallel acceleration, resulting in a model that is an order of magnitude faster than CNN/RNN based alternatives.

While self-attention models show the significant promise in the field of recommender system, it however has some drawbacks as well. Although the operations are highly parallelizable, each operations are quadratically dependent on the input length which can pose a problem for a long sequences (e.g. clicks). At the same time, the memory requirement for each blocks is also high. In this work, we show that a simple lightweight convolution model can match or exceed self-attentions performance. Unlike self-attention, it requires linear operations and less memory. We perform experiments on four different datasets showing effectiveness of our model.

2 Related Work

Here, we briefly go through different models (general, temporal, sequential, convolutional) models for recommender system.

2.1 General Recommendation

In past, recommender system focused on the users and items interaction matrices. These interaction can be either explicit (e.g. ratings) or implicit (e.g. clicks, purchase, comments, etc.) [4, 5]. Popular approaches include Matrix Factorization (MF) methods which aim to discover to uncover latent dimensions from the interaction matrix. These dimension can represent users' preferences and items' properties. Then we can prediction based on the inner product between the user and item embeddings [6], [7]. Modeling implicit behavior is challenging that explicit due to the ambiguous interpretation of of 'non-observed' data (i.e. the item which user did not interact with). To address the problem, point-wise [4] and pairwise [5] methods are proposed to solve such challenges.

2.2 Temporal Recommendation

A stronger recommendation can be made if we can model the timestamp of each action of a user. It has shown to perform better than any other content or collaborative based recommendation system in Netflix Challenge. The winner of that challenge, i.e. TimeSVD++ [16], showed that it achieved strong results by splitting time into several segments and modeling users and items separately in each. Such models are essential to understand datasets that exhibit significant temporal 'drift' [16, 18].

2.3 Sequential Recommendation

Sequential recommendation (or next-item recommendation) is slightly different from temporal recommendation. In sequential recommendation, it only considers the order actions ordered by the timestamp, but does not explicitly model the temporal patterns. In other words, in sequential recommendation, the 'context' of users' actions are more important than 'exact' time of the action. We can model such action, by item-item transition matrices. For instance, FPMC uses the same matrix factorization mentioned above, but augment it with an item-item transition term to capture long-term preferences and short-term transitions respectively [1]. Essentially, this is a first-order Markov Chain (MC) model which captures the transition. There can be also higher-order MCs which assume the next action is related to several previous actions. Since the last visited item is often the key factor affecting the user's next action (essentially providing 'context'), first-order MC based methods show strong performance, especially on sparse datasets [19]. There are also methods adopting high-order MCs that consider more previous items [20], [21]. For example, GRU4Rec uses Gated Recurrent Units (GRU) to model click sequences for session-based recommendation [2], and an improved version further boosts its Top-N recommendation performance [26]. In each time step, RNNs take the state from the last step and current action as its input.

Table 1: Notation.

Notation	Explanation
$(a_1, \ldots, a_n)$	user interactions containing n actions
s_u	sequence belonging to user u
$e_t \in \mathbb{R}^d$	d-dimensional embedding for action a_t
$p_t \in \mathbb{R}^d$	d-dimensional positional embedding
o_t	model prediction
r_{o_t}	score for model prediction o_t
$\|I\|$	total number of items
$\|U\|$	total number of users

These dependencies make RNNs less efficient, though techniques like 'session parallelism' have been proposed to improve efficiency [2].

2.4 Attention Mechanisms

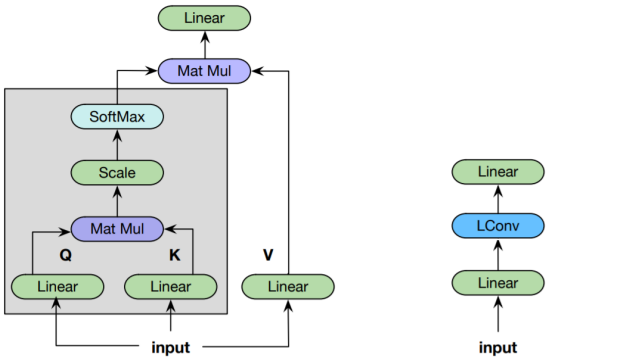
Attention mechanism have been very active topic of research in recent years. It has shown a significant amount of improvement in performance in various difficult task, especially, in image captioning [27] and machine translation [28]. The idea behind attention mechanisms is fairly simple too. Essentially, the mechanism puts some weight on some part of the input that are relevant to sequential outputs. As it focuses on some part of the input while generating output, it makes the results more interpretable too. Such application of attention mechanism has been done in recommender system as well [29]-[31]. For example, Attentional Factorization Machines (AFM) [30] learn the importance of each feature interaction for content-aware recommendation. But in these cases, attention is actually an added component in the original underlying model (e.g. RNNs+attention, FMs+attention, etc.). Recently, a purely attention-based sequence-to-sequence method, Transfomer [3], achieved state-of-the-art performance and efficiency on machine translation tasks which had previously been dominated by RNN/CNN-based approaches [32, 33]. Its model relies heavily on the proposed 'self-attention' modules to capture complex structures in sentences. Inspired by its recent success, self-attention has been adapted into recommender system [49]

2.5 CNN based Models

Convolution networks have proved to be very successful in image recognition task. Given that they require much less parameters, there have been some interests to make CNN models work in sequential task as well. Particularly, in recommender system, Convolutional Sequence Embedding (Caser), a CNN-based method, has been proposed. It views the embedding matrix of L previous items as an 'image' and applies convolutional operations to extract transitions [22]. But sequences are different than image, and such consideration may hinder capturing the item to item transitions. Recently, [50] showed that a purely convolutional based network may perform competitively to the best reported self-attention networks. Our work is based on their findings. In next section, we further discuss the details of self-attention and our convolutional networks.

3 Methods

In this section, we formally introduce both self-attention and our proposed model. Both models share some common structure, i.e., embedding layer, sequential layer, and prediction layer, except in the sequential layer, self-attention is replaced by lightweight convolutions in our model.



(a) Self-attention (b) Lightweight convolution

Figure 1: Comparison of two models. Credit [50]

3.1 Problem Description

We have a sequence of user actions (e.g. rate, review) $s = (a_1, a_2, \ldots, a_n)$, where each action a_i contains an item ID. Our task is to predict the last the action a_n, i.e. the next item based on $a_1, \ldots, a_{n-1}$ actions. More specifically, in each step, our goal is to predict the next action, i.e. given $a_1, \ldots, a_{n-1}$, predict $a_2, \ldots, a_n$. Table 1 shows the necessary notations.

3.2 Embedding Layer

In this layer, we seek to embed item features into a d-dimensional space, i.e. a_t into $e_t \in \mathbb{R}^d$. We transform the training sequence $s = (a_1, a_2, \ldots, a_n)$ into a fixed-length sequence where n represents the maximum length that our model can handle. If the sequence length is greater than n, we consider the most recent n actions. If the sequence length is less than n, we repeatedly add a 'padding' item to the left until the length is n. A constant zero vector 0 is used as the embedding for the padding item.

Positional Embedding: As self-attention is not aware of the positions of previous items, we have to inject a learnable position embedding. $p_t \in \mathbb{R}^d$. We also do the same for our convolutional model, and observe a better performance.

3.3 Sequential Models

With the embeddings $(e_1, e_2, \ldots, e_{n-1})$, we need to build a model to make predictions of next items, i.e. $(e_2, e_3, \ldots, e_n)$

Self-Attention The definition of self-attention is

$$\mathit{Attention}(Q, K, V) = \mathit{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

, where the query Q, key K and value V consider the same object. That is to say $Q = EW^Q$, $K = EW^K$, $V = EW^V$, where $E = [e_1^T; \ldots; e_n^T]$, and $W^Q, W^K, W^V \in \mathbb{R}^{d \times d}$ are projection matrices. If there are multiple H attention-heads, embedding dimension is divided into H ways, and the final results are concatenated. To maintain causality and to prevent information leaking from back to front, the attention is modified by forbidding all links between Q_i and K_j $(j > i)$. As the self-attention model is essentially a linear model, a two-layer feed-forward network is applied on its output to achieve nonlinearity. Note that as the self-attention operation is not aware of order, similar to [3], each position is assigned a learned embedding which is added to the input embeddings E.

Lightweight Convolution In contrast to self-attention, convolution has a fixed context window. We can perform convolution in different number of ways. In our task, we consider

to perform convolution in 1d space (i.e. sequence) and is based on fixed size of kernel which slides over the input sequence and it determines the importance of context elements with a set of weights that do not change over time steps. The number of parameters can be reduced from d^2k to dk where k is the kernel width if we perform a depthwise convolution with weight $W \in \mathbb{R}^{d \times k}$. The output $O \in \mathbb{R}^n \times d$ of for each element i in input sequence S is defined as:

$$O_{i,c} = \text{DepthwiseConv}(S, W_{c,:}, i, c) = \sum_{j=1}^{k} W_{c,j}.S_{(i+j-[k+1]/2),c}$$

Here c is the channel. Each dimension of the latent space can be considered as number of channel, so usually $c = d$. Similar to self-attention, if we consider multiple heads H, this can be further reduced to $c = d/H$. This can allow parallel computation of $d \times k$ to $d/H \times k$ operation. But it will require us same number of number of parameters ($H \times d/H \times k$). To reduce the number of parameters, we can tie each of the d/H weights in each H channels, so we can get $H \times 1 \times k$ or $H \times k$ number of weights. This is a huge reduction of number of weights. A side by side comparison of these two architectures is shown in Figure 1.

FeedForward Network Both self-attention and convolutional models are linear models. To introduce non linearly, the next step is to feed the output from these models, x, to two layer feed-forward network:

$$o = FFN(x) = W_2(ReLU(W_1x + b_1)) + b_2$$

where W_1, and W_2 are $d \times d$ matrices and b_1 & b_2 are d-dimensional vectors.

3.4 Prediction Layer

For each sequence, there is one ground truth item (i.e. e_t). We can sample some negative items for each sequence as well. After getting the output from FFN layer, we can make prediction in the following manner: we calculate the dot product between the output o and positive/negative items, and we will obtain a score. Whichever product gives us the highest score is our prediction. For example if our model is effective, the ground truth item should have more scores than other generated negative items, and thus the accuracy will go up. The loss function is defined as follows:

$$-\sum_{s^u \in S} \sum \left[\log(\sigma(r_{o_t})) + \sum_{j \notin s^u} \log(\sigma(1 - r_{o_j})) \right]$$

where r_{o_t} is for ground truth score, and r_{o_j} for negative item score.

4 Complexity Analysis

4.1 Space Complexity

The learned parameters in self-attention model are from the embeddings and parameters in the self-attention layers, feed-forward networks and layer normalization. The total number of parameters are $O(|I|d + nd + d^2)$. On the other hand, complexity of parameters of the sequential layer in our model does not depend on the length of input or dimensions. As mentioned previously, dimensions of each parameter in each channel is $1 \times k$, so for H channels total number of parameters are $O(|I|d + Hk + d^2)$ which are a major reduction of parameters.

4.2 Time Complexity

The computational complexity of self-attention model is mainly due to the self-attention layer and the feedforward network, which is $O(n^2d + nd^2)$. The dominant term is typically $O(n^2d)$ from the self-attention layer. As each self-attention layer is fully parallelizable and ideal for GPU acceleration, operations are still fast. In contrast, our model needs only $O(ndk + nd^2)$ number operations (convolutional layer + feed forward network). Thus operations of our model scales linearly. This is important especially for long-range sequences.

	Amazon Beauty	Amazon Games	Steam	MovieLens-1M
#Users	52,024	31,013	334,730	6,040
#Items	57,289	23,715	13,047	3,416
#Total actions	0.4M	0.3M	3.7M	1M
#Avg. actions/users	7.6	9.3	11	163.5
#Avg. actions/items	6.9	12.1	282.5	289.1

Table 2: Data Statistics (after preprocessing).

5 Experiments

In this section, we present our experimental setup and empirical results. Our experiments are designed to answer the following research questions:

RQ1: How does ConvRec perform compared to state-of-the-art models including other CNN/RNN/Self-attention based methods?

RQ2: What is the influence of various components in the ConvRec architecture?

RQ3: What is the training efficiency and scalability (regarding n) of ConvRec?

5.1 Datasets

We evaluate our methods on four datasets from three real world applications. The datasets vary significantly in domains, platforms, and sparsity:

Amazon: A series of datasets introduced in [46], comprising large corpora of product reviews crawled from Amazon.com. Top-level product categories on Amazon are treated as separate datasets. We consider two categories, 'Beauty,' and 'Games.' This dataset is notable for its high sparsity and variability.

Steam: Steam is a large online video game distribution platform. [49] provides the dataset contains 2,567,538 users, 15,474 games and 7,793,069 English reviews spanning October 2010 to January 2018. The dataset also includes rich information that might be useful in future work, like users' play hours, pricing information, media score, category, developer (etc.).

MovieLens: A widely used benchmark dataset for evaluating collaborative filtering algorithms. We use the version (MovieLens-1M) that includes 1 million user ratings. We followed the same preprocessing procedure from [1],[19], [21].

For all datasets, we treat the presence of a review or rating as implicit feedback (i.e., the user interacted with the item) and use timestamps to determine the sequence order of actions. We discard users and items with fewer than 5 related actions. For partitioning, we split the historical sequence s_u, for each user u into three parts: (1) the most recent action a_t for testing, (2) the second most recent action $a_t - 1$ for validation, and (3) all remaining actions for training. Note that during testing, the input sequences contain training actions and the validation action.

Data statistics are shown in Table 2. We see that the two Amazon datasets have the fewest actions per user and per item (on average), Steam has a high average number of actions per item, and MovieLens-1m is the most dense dataset.

5.2 Implementation Details

The default version of SASRec as proposed by the author[49] uses two self-attention blocks (b = 2), the learned positional embedding, shared Item embeddings, single head, maximum length of 50 (except 200 for ML-1m), dropout of 0.5 (0.2 in case of ML-1m). SASRec is only available in TensorFlow. The optimizer is the Adam optimizer [41], the learning rate is set to 0.001, and the batch size is 128.

On the other hand, ConvRec utilizes various modules of FairSeq and therefore, currently we provide a Pytorch implementation. The optimizer we used is the latest corrected version of

dataset	metric	**SASRec**	**ConvRec**
Amazon Beauty	NDGC@10	0.3082	**0.3257**
	HR@10	0.4652	**0.4833**
Amazon Games	NDGC@10	0.5301	**0.5379**
	HR@10	0.7324	**0.7361**
Steam	NDGC@10	**0.6246**	0.6091
	HR@10	**0.8678**	0.8576
ML-1m	NDGC@10	0.5841	**0.5936**
	HR@10	**0.8178**	0.8149

Table 3: Performance of both models (maximum number of epochs 200)

Adam, namely AdamW [51]. Both learing rate and batch size are kept unchanged. However, dropout of 0.2 in all cases. Our number of block is one, with learned positional embeddings. Item embeddings are shared with embedding dimension of 50. Maximum length is 50, except for ML-1m (200). We use 5 attention heads in order to decrease number of parameters and increase number of channels. We fixed kernel size of 31.

5.3 Evaluation Metrics

The two of the most popular Top-N metrics in recommender systems are Hit Rate@10 and NDCG@10 [14], [19]. Hit@10 is the ratio of the ground-truth next item is among the top 10 predicted items, while NDCG@10 is a position-aware metric which assigns larger weights on higher positions. In our case, we only have one test item for each user, and so Hit@10 is equivalent to Recall@10, and is proportional to Precision@10. For each user u, we randomly sample 100 negative items (to avoid heavy computation [14], [48]), and rank these items with the ground-truth item. Based on the rankings of these 101 items, Hit@10 and NDCG@10 can be evaluated.

5.4 Performance

Table 3 presents the recommendation performance of both of these methods on the four datasets (RQ1). We can see that our method ConvRec performs competitively on both sparse and dense datasets. It either matches or outperforms in all dataset except for Steam. One likely reason can be there are more users in Steam dataset, possibly we are facing some training issues. We plan to investigate performance for this dataset in future.

5.5 Ablation Study

Since there are many components in ConvRec, we analyze their significance via an ablation study (Table 4).

Positional Embedding Without the positional embedding P, the model makes recommendations based on users' past actions, but their order does not matter. In our experiment, removing PE did not improve performance, which shows it is important to capture the sequential behavior by ordering the input. We also experimented with fixed positional embedding and sinusoidal embedding but did not observe any improvements.

Residual Connection Residual connection is useful when the network is very deep, and we want the gradient to flow to the beginning without vanishing. But as we have seen good performance with only one decoder, our model is not that deep, and we think residual connection is a redundant operation. In denser dataset, this is actually responsible for worse performance as we are putting more importance to most recent actions as can be seen from the table.

	NDGC@10	HitRate@10
1. Default	**0.5936**	0.8149
2. Remove PE	0.5875	0.8109
3. Fixed PE	0.5933	0.8165
4. Sinusoidal PE	0.5841	0.8117
5. RC	0.5865	0.8167
6. Remove Layer Norm	0.5850	0.8158
6. Remove LightConv	0.4890	0.7306
7. Remove Dropout	0.5586	0.7882
8. 2 Blocks	0.5927	**0.8172**
9. Single Head	0.5870	0.8100

Table 4: Ablation Study for Ml-1m dataset

	50	200	400	600	900
1. SASRec	1.4	1.8	3.2	5.2	Failed
2. ConvRec	0.9	**1.3**	**2.8**	**4**	**7**

Table 5: Memory Consumption in Gigabyte

Remove Layer Normalization Layer normalization is another trick and is applied to optimize the training of the network. Removing them may hamper the training process, but not significantly, because we are using the recent corrected version of Adam optimizer which has been shown very strong in most cases. This is shown when we remove layer normalization but do not see any significant decrease in improvement in performance.

Remove LightConv Lightweight convolution is the most important component of our model. Naturally without it, the performance drops drastically, as FFN itself are not capable to capture the sequential behaviors.

Remove Dropout Dropout is an important part especially because it helps the model to generalize well. When we remove the dropout, the performance drops significantly, which shows the necessity of this component.

Number of blocks To keep the model simple, we usually use only one block. We believe making the network wider (i.e. increased parameter in feed forward network) is more practical than deeper network. That is why when we increase the number of blocks it did not increase the performance significantly.

Number of Heads As stated previously, number of heads correspond to number of channels for convolution network. If we increase the number of heads, we believe different kernel can capture different important dynamics of the sequence. So, using only head can reduce the performance, which is the case in our experiment.

5.6 Scalability

One of the drawbacks of self-attention model is the quadratic relationship of the maximum length n. However, it has been shown that this can be effectively parallelized with GPUs. The number of parameters are dependent on n too, which can become a bottleneck for long-range sequences. On other hand, our convolutional model's computation scales linearly with length of sequence and thus much more scalable. The number of parameters does not depend on input length, so memory footprint is also much less. As these models have been deployed in different platforms (i.e. tensorflow and pytorch), runtime comparision will not be fair. But we can show the different memory requirement of these two models. Here we measure the required memory for different length of input n and performance of both models. Table 5 shows the performance and efficiency of ConvRec with various sequence lengths. As we can memory requirement for SASRec is much higher than ConvRec.

6 Conclusion

In this work, we proposed a lightweight convolutional method for next item recommendation. We performed extensive experiments on four real-world e-commerce datasets, and investigated the performance of both self-attention and convolutional methods. Our analysis and experiment show that our lightweight convolutional counterpart can match or exceed the performance of the state-of-the-art results of self-attention in much less number of parameters and operations, which are crucial, especially for long range sequences. In future, we plan to apply our method in session-based recommendations. We also plan to provide a Tensorflow implementation of ConvRec for a fair runtime comparison.

References

[1] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in WWW, 2010.

[2] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," in ICLR, 2016.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in NIPS, 2017.

[4] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in ICDM, 2008.

[5] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: bayesian personalized ranking from implicit feedback," in UAI, 2009.

[6] F. Ricci, L. Rokach, B. Shapira, and P. Kantor, Recommender systems handbook. Springer US, 2011.

[7] Y. Koren and R. Bell, "Advances in collaborative filtering," in Recommender Systems Handbook. Springer, 2011.

[8] S. Kabbur, X. Ning, and G. Karypis, "Fism: factored item similarity models for top-n recommender systems," in SIGKDD, 2013.

[9] S. Zhang, L. Yao, and A. Sun, "Deep learning based recommender system: A survey and new perspectives," arXiv, vol. abs/1707.07435, 2017.

[10] S. Wang, Y. Wang, J. Tang, K. Shu, S. Ranganath, and H. Liu, "What your images reveal: Exploiting visual contents for point-of-interest recommendation," in WWW, 2017.

[11] W. Kang, C. Fang, Z. Wang, and J. McAuley, "Visually-aware fashion recommendation and design with generative image models," in ICDM, 2017.

[12] H. Wang, N. Wang, and D. Yeung, "Collaborative deep learning for recommender systems," in SIGKDD, 2015.

[13] D. H. Kim, C. Park, J. Oh, S. Lee, and H. Yu, "Convolutional matrix factorization for document context-aware recommendation," in RecSys, 2016.

[14] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural collaborative filtering," in WWW, 2017.

[15] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in WWW, 2015.

[16] Y. Koren, "Collaborative filtering with temporal dynamics," Communications of the ACM, 2010.

[17] C. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing, "Recurrent recommender networks," in WSDM, 2017.

[18] L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. G. Carbonell, "Temporal collaborative filtering with bayesian probabilistic tensor factorization," in SDM, 2010.

[19] R. He, W. Kang, and J. McAuley, "Translation-based recommendation," in RecSys, 2017.

[20] R. He, C. Fang, Z. Wang, and J. McAuley, "Vista: A visually, socially, and temporally-aware model for artistic recommendation," in RecSys, 2016.

[21] R. He and J. McAuley, "Fusing similarity models with markov chains for sparse sequential recommendation," in ICDM, 2016.

[22] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in WSDM, 2018.

[23] H. Jing and A. J. Smola, "Neural survival recommender," in WSDM, 2017.

[24] Q. Liu, S. Wu, D. Wang, Z. Li, and L. Wang, "Context-aware sequential recommendation," in ICDM, 2016.

[25] A. Beutel, P. Covington, S. Jain, C. Xu, J. Li, V. Gatto, and E. H. Chi, "Latent cross: Making use of context in recurrent recommender systems," in WSDM, 2018.

[26] B. Hidasi and A. Karatzoglou, "Recurrent neural networks with top-k gains for session-based recommendations," CoRR, vol. abs/1706.03847, 2017.

[27] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in ICML, 2015.

[28] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in ICLR, 2015.

[29] J. Chen, H. Zhang, X. He, L. Nie, W. Liu, and T. Chua, "Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention," in SIGIR, 2017.

[30] J. Xiao, H. Ye, X. He, H. Zhang, F. Wu, and T. Chua, "Attentional factorization machines: Learning the weight of feature interactions via attention networks," in IJCAI, 2017.

[31] S. Wang, L. Hu, L. Cao, X. Huang, D. Lian, and W. Liu, "Attentionbased transactional context embedding for next-item recommendation," in AAAI, 2018.

[32] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey et al., "Googleâs neural machine translation system: Bridging the gap between human and machine translation," arXiv preprint arXiv:1609.08144, 2016.

[33] J. Zhou, Y. Cao, X. Wang, P. Li, and W. Xu, "Deep recurrent models with fast-forward connections for neural machine translation," TACL, 2016.

[34] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in ECCV, 2014.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in CVPR, 2016.

[36] L. J. Ba, R. Kiros, and G. E. Hinton, "Layer normalization," CoRR, vol. abs/1607.06450, 2016.

[37] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in ICML, 2015.

[38] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," JMLR, 2014.

[39] D. Warde-Farley, I. J. Goodfellow, A. C. Courville, and Y. Bengio, "An empirical analysis of dropout in piecewise linear networks," CoRR, vol. abs/1312.6197, 2013.

[40] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," Computer, 2009.

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in ICLR, 2015.

[42] D. Povey, H. Hadian, P. Ghahremani, K. Li, and S. Khudanpur, "A time-restricted self-attention layer for asr," in ICASSP, 2018.

[43] P. Wang, J. Guo, Y. Lan, J. Xu, S. Wan, and X. Cheng, "Learning hierarchical representation model for next basket recommendation," in SIGIR, 2015.

[44] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” CoRR, vol. abs/1803.01271, 2018.

[45] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber et al., “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies,” 2001.

[46] J. J. McAuley, C. Targett, Q. Shi, and A. van den Hengel, “Image-based recommendations on styles and substitutes,” in SIGIR, 2015.

[47] S. Feng, X. Li, Y. Zeng, G. Cong, Y. M. Chee, and Q. Yuan, “Personalized ranking metric embedding for next new poi recommendation,” in IJCAI, 2015.

[48] Y. Koren, “Factorization meets the neighborhood: a multifaceted collaborative filtering model,” in SIGKDD, 2008.

[49] Kang, Wang-Cheng, and Julian McAuley. "Self-attentive sequential recommendation." 2018 IEEE International Conference on Data Mining (ICDM). IEEE, 2018.

[50] Wu, Felix, et al. "Pay Less Attention with Lightweight and Dynamic Convolutions." arXiv preprint arXiv:1901.10430 (2019).

[51] Loshchilov, Ilya, and Frank Hutter. "Fixing weight decay regularization in adam." arXiv preprint arXiv:1711.05101 (2017).