Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding

Jiaxi Tang
School of Computing Science
Simon Fraser University
British Columbia, Canada
jiaxit@sfu.ca

Ke Wang
School of Computing Science
Simon Fraser University
British Columbia, Canada
wangk@cs.sfu.ca

ABSTRACT

Top-N sequential recommendation models each user as a sequence of items interacted in the past and aims to predict top-N ranked items that a user will likely interact in a "near future". The order of interaction implies that sequential patterns play an important role where more recent items in a sequence have a larger impact on the next item. In this paper, we propose a Convolutional Sequence Embedding Recommendation Model (*Caser*) as a solution to address this requirement. The idea is to embed a sequence of recent items into an "image" in the time and latent spaces and learn sequential patterns as local features of the image using convolutional filters. This approach provides a unified and flexible network structure for capturing both general preferences and sequential patterns. The experiments on public data sets demonstrated that Caser consistently outperforms state-of-the-art sequential recommendation methods on a variety of common evaluation metrics.

CCS CONCEPTS

• **Information systems** → **Retrieval models and ranking**;

KEYWORDS

Recommender System; Sequential Prediction; Convolutional Neural Networks

ACM Reference format:
Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *Proceedings of Eleventh ACM International Conference on Web Search and Data Mining, February 5–9, 2018, Marina Del Rey, CA, USA, (WSDM 2018),* 9 pages.
DOI: 10.1145/3159652.3159656

1 INTRODUCTION

Recommender systems have become a core technology in many applications. Most systems, e.g., *top-N recommendation* [9][19], recommend the items based on the user's *general preferences* without paying attention to the recency of items.

For example, some user always prefer Apple's products to Samsung's products. General preferences represent user's long term and static behaviors. Another type of user behaviors is *sequential patterns* where the next item or action more likely depends on the items or actions the user engaged recently. Sequential patterns represent the user's short term and dynamic behaviors and come from a certain relationship between the items within a close proximity of time. For example, a user likely buys phone accessories soon after buying an iPhone, though in general the user does not buy phone accessories. In this case, the systems that consider only general preferences will miss the opportunity of recommending phone accessories after selling an iPhone since buying phone accessories is not a long term user behavior.

1.1 Top-N Sequential Recommendation

To model user's sequential patterns, the work in [17, 21] considers *top-N sequential recommendation* that recommends N items that a user likely interacts with in a near future. This problem assumes a set of users $\mathcal{U} = \{u_1, u_2, \cdots, u_{|\mathcal{U}|}\}$ and a universe of items $\mathcal{I} = \{i_1, i_2, \cdots, i_{|\mathcal{I}|}\}$. Each user u is associated with a sequence of some items from $\mathcal{I}$, $\mathcal{S}^u = (\mathcal{S}^u_1, \cdots, \mathcal{S}^u_{|\mathcal{S}^u|})$, where $\mathcal{S}^u_i \in \mathcal{I}$. The index t for $\mathcal{S}^u_t$ denotes the order in which an action occurs in the sequence $\mathcal{S}^u$, not the absolute timestamp as in temporal recommendation like [14, 31, 34]. Given all users' sequences $\mathcal{S}^u$, the goal is to recommend each user a list of items that maximize her/his future needs, by considering both general preferences and sequential patterns. Unlike conventional top-N recommendation, top-N sequential recommendation models the user behavior as a sequence of items, instead of a set of items.

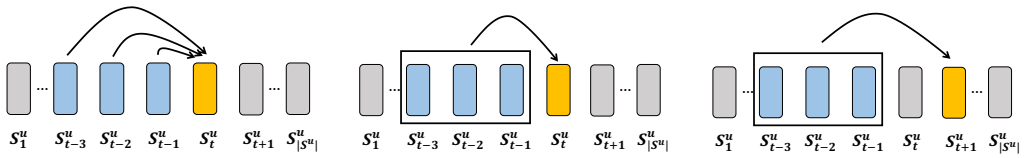
1.2 Limitations of Previous Work

The Markov chain based model [2, 6, 21, 30] is an early approach to top-N sequential recommendation, where an L-order Markov chain makes recommendations based on L previous actions. The first-order Markov chain is an item-to-item transition matrix learnt using maximum likelihood estimation. Factorized personalized Markov chains (FPMC) [21] proposed by Rendle *et al.* and its variant [2] improved this method by factorizing this transition matrix into two latent and low-rank sub-matrices. Factorized Sequential Prediction with Item Similarity ModeLs (Fossil) [6] proposed by He *et al.* generalizes this method to high-order Markov chains using a weighted sum aggregation over previous items' latent representations. However, existing approaches suffered from two major limitations:

Fail to model union-Level sequential patterns. As shown in Figure 1a, the Markov chain models only **point-level** sequential patterns where each of the previous actions (blue) influences the target action (yellow) individually, instead of collectively. FPMC

WSDM 2018, February 5–9, 2018, Marina Del Rey, CA, USA

DOI: 10.1145/3159652.3159656



(a) point-level (b) union-level, no skip (c) union-level, skip once

Figure 1: An example of point and union level dynamic pattern influences, the order of Markov chain $L = 3$

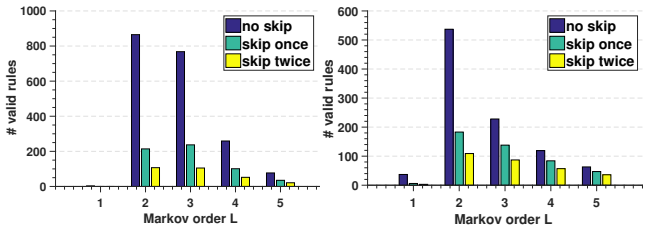
and Fossil fall into this taxonomy. Although Fossil [6] considers a high-order Markov chain, the overall influence is a weighted sum of previous items' latent representations factorized from first-order Markov transition matrices. Such aggregation of point-level influences is not sufficient to model the **union-level** influences shown in Figure 1b where several previous actions, in that order, jointly influence the target action. For example, buying both milk and butter together leads to a higher probability of buying flour than buying milk or butter individually; buying both RAM and Hard Drive is a better indication of buying Operating System next than buying only one of the components.

Fail to allow skip behaviors. Existing models don't consider **skip behaviors** of sequential patterns as shown in Figure 1c, where the impact from past behaviors may skip a few steps and still have strength. For example, a tourist has check-ins sequentially at airport, hotel, restaurant, bar, and attraction. While the check-ins at the airport and hotel do not immediately precede the check-in of the attraction, they are strongly associated with the latter. On the other hand, the check-in at the restaurant or bar has little influence on the check-in of the attraction (because they do not necessarily occur). A L-order Markov chain does not explicitly model such skip behaviors because it assumes that the L previous steps have an influence on the immediate next step.

To provide evidences of union-level influences and skip behaviors, we mine sequential association rules [1, 4] of the following form from two real life data sets, MovieLens and Gowalla (see the details of these data sets in Section 4)

$$(\mathcal{S}^u_{t-L}, \cdots, \mathcal{S}^u_{t-2}, \mathcal{S}^u_{t-1}) \rightarrow \mathcal{S}^u_t. \tag{1}$$

For a rule $X \rightarrow Y$ of the above form, the support count $sup(XY)$ is the number of sequences in which X and Y occur in order as in the rule, and the confidence, $\frac{sup(XY)}{sup(X)}$, is the percentage of the sequences in which Y follows X among those in which X occurs. This rule represents the joint influence of all the items in X on Y. By changing the right hand side to $\mathcal{S}^u_{t+1}$ or $\mathcal{S}^u_{t+2}$, the rule also captures the influences with one or two step skips. Figure 2 summarizes the number of rules found versus the Markov order L and skip steps with the minimum support count = 5 and the minimum confidence = 50% (we also tried the minimum confidence of 10%, 20%, and 30%, these trends are similar). Most rules have the orders $L = 2$ and $L = 3$ and the confidence of rules gets higher for larger L. The figure also tells that a sizable number of rules have skip steps 1 or 2. These findings support the existence of union-level influences and skip behaviors.



(a) MovieLens (b) Gowalla

Figure 2: The number of association rules vs L and skip steps. The minimum support count = 5 and the minimum confidence = 50%.

1.3 Contributions

To address these above limitations of existing works, we propose a *ConvolutionAl Sequence Embedding Recommendation Model*, or *Caser* for short, as a solution to top-N sequential recommendation. This model leverages the recent success of convolution filters of Convolutional Neural Network (CNN) to capture local features for image recognition [11, 16] and natural language processing [12]. The novelty of Caser is to represent the previous L items as an $L \times d$ matrix $\boldsymbol{E}$, where d is the number of latent dimensions and the rows preserve the order of the items. Similar to [12], we regard this embedding matrix as the "image" of the L items in the latent space and search for sequential patterns as local features of this "image" using various convolutional filters. Unlike image recognition, however, this "image" is not given in the input and must be learnt simultaneously with all filters.

Compared to existing methods, Caser offers several distinct advantages. (1) Caser uses horizontal and vertical convolutional filters to capture sequential patterns at point-level, union-level, and of skip behaviors. (2) Caser models both users' general preferences and sequential patterns, and generalizes several existing state-of-the-art methods in a single unified framework. (3) Caser outperforms state-of-the-art methods for top-N sequential recommendation on real life data sets. In the rest of the paper, we discuss further related work in Section 2, the Caser method in Section 3, and experimental studies in Section 4.

2 FURTHER RELATED WORK

Conventional recommendation methods, *e.g.*, collaborative filtering [24], matrix factorization [15, 22], and top-N recommendation [9][19], are not suitable for capturing sequential patterns because they do not model the order of actions. Early works on sequential pattern mining [1, 4] find explicit sequential association rules based on statistical co-occurrences [17]. This approach depends on

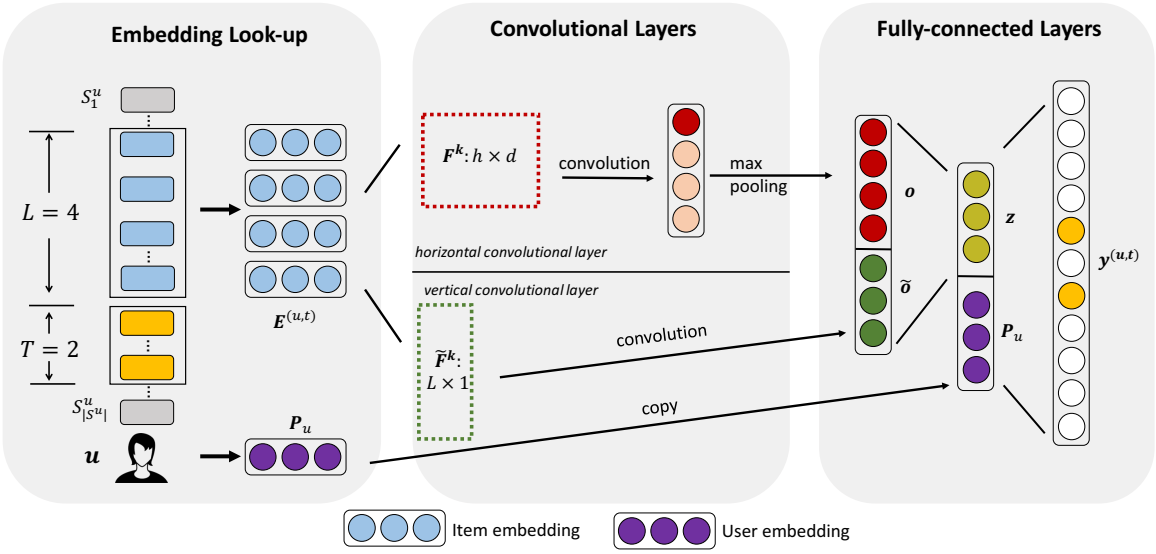


Figure 3: The network architecture of *Caser*. The rectangular boxes represent items $S_1^u, \cdots, S_{|S^u|}^u$ in user sequence, whereas a rectangular box with circles inside stands for a certain vector *e.g.,* user embedding P_u. The dash rectangular boxes are convolutional filters with different sizes. The red circles in convolutional layers stand for the max values in each of the convolution results. Here we are using previous 4 actions ($L = 4$) to predict which items this user will interact with in next 2 steps ($T = 2$).

the explicit representation of patterns, thus, could miss patterns in unobserved states. Also, it suffers from a potentially large search space, sensitivity to threshold settings, and a large number of rules, most being redundant.

Restricted Bolzmann Machine (RBM) [23] is the first successful 2-layers neural network that is applied to recommendation problems. Auto-encoder framework [25, 29] and its variant denoising auto-encoder [32] also produce a good recommendation performance. Convolutional neural network (CNN) [36] has been used to extract users' preferences from their reviews. None of these works is for sequential recommendation.

Recurrent neural networks (RNN) was used for session-based recommendation [8, 10]. While RNN has shown to have an impressive capability in modeling sequences [18], its sequentially connected network structure may not work well under sequential recommendation setting. Because in sequential recommendation problem, not all adjacent actions have dependency relationships (*e.g.* a user bought i_2 after i_1 only because she loves i_2). Our experimental results in Section 4 verify this point: RNN-based method performs better when data sets contains considerable sequential patterns. While our proposed method doesn't model sequential pattern as adjacent actions, it adopts convolutional filters from CNN and model sequential patterns as local features of the embeddings of previous items. This approach offers the flexibility of modeling sequential patterns at both point level and union level, and skip behaviors in a single unified framework. In fact, we will show that Caser generalizes several state-of-the-art methods.

A related but different problem is temporal recommendation [26, 31, 34]. For example, temporal recommendation recommends coffee in the morning, instead of evening, whereas our top-N sequential recommendation would recommend phone accessories soon after a user bought an iPhone, independently of the time. Clearly, the two problems are different and require different solutions.

3 PROPOSED METHODOLOGY

The proposed model, ConvolutionAl Sequence Embedding Recommendation (*Caser*), incorporates the Convolutional Neural Network (CNN) to learn sequential features, and Latent Factor Model (LFM) to learn user specific features. The goal of Caser's network design is multi-fold: capture both user's general preferences and sequential patterns, at both union-level and point-level, and capture skip behaviors, all in unobserved spaces. Shown in Figure 3 Caser consists of three components: Embedding Look-up, Convolutional Layers, and Fully-connected Layers. To train the CNN, for each user u, we extract every L successive items as input and their next T items as the targets from the user's sequence $\mathcal{S}^u$, shown on the left side of Figure 3. This is done by sliding a window of size $L + T$ over the user's sequence, and each window generates a training instance for u, denoted by a triplet (u, previous L items, next T items).

3.1 Embedding Look-up

Caser captures sequence features in the latent space by feeding the embeddings of previous L items into the neural network. The embedding $\boldsymbol{Q}_i \in \mathbb{R}^d$ for item i is a similar concept to its latent factors. Here d is the number of latent dimensions. The embedding look-up operation retrieves the previous L items' embeddings and stacks them together, resulting in a matrix $\boldsymbol{E}^{(u,t)} \in \mathbb{R}^{L \times d}$ for user

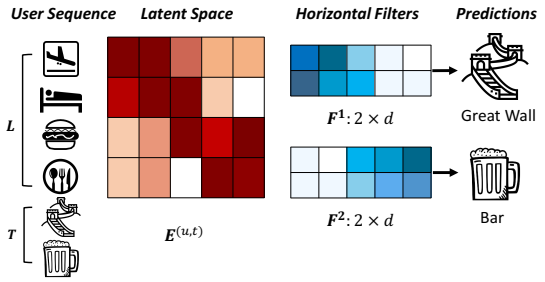


Figure 4: Darker colors mean larger values. The first filter captures "(Airport, Hotel) → Great Wall" by interacting with the embedding of airport and hotel and skipping that of fast food and restaurant. The second filter captures "(Fast Food, Restaurant) → Bar".

u at time step t:

$$E^{(u,t)} = \begin{bmatrix} Q_{S^u_{t-L}} \\ \vdots \\ Q_{S^u_{t-2}} \\ Q_{S^u_{t-1}} \end{bmatrix}. \tag{2}$$

Along with the item embeddings, we also have an embedding $P_u \in \mathbb{R}^d$ for a user u, representing user features in latent space. These embeddings are represented by blue and purple circles in the box of Embedding Look-up in Figure 3.

3.2 Convolutional Layers

Our approach leverages the recent success of convolution filters of CNN in capturing local features for image recognition [11, 16] and natural language processing [12]. Borrows the idea of using CNN in text classification [12], our approach regards the $L \times d$ matrix E as the "image" of the previous L items in the latent space and regard sequential patterns as local features of this "image". This approach enables the use of convolution filters to search for sequential patterns. Figure 4 shows two "horizontal filters" that capture two union-level sequential patterns. These filters, represented as $h \times d$ matrices, have the height $h = 2$ and the full width equal to d. They pick up signals for sequential patterns by sliding over the rows of E. For example, the first filter picks up the sequential pattern "(Airport, Hotel) → Great Wall" by having larger values in the latent dimensions where Airport and Hotel have larger values. Similarly, a "vertical filter" is a $L \times 1$ matrix and will slide over the columns of E. More details are explained below. Unlike image recognition, the "image" E is not given because the embedding Q_i for all items i must be learnt simultaneously with all filters.

Horizontal Convolutional Layer. This layer, shown in the upper part of the second component in Figure 3, has n horizontal filters $F^k \in \mathbb{R}^{h \times d}$, $1 \le k \le n$. $h \in \{1, \cdots, L\}$ is the height of a filter. For example, if $L = 4$, one may choose to have $n = 8$ filters, two for each h in $\{1, 2, 3, 4\}$. F^k will slide from top to bottom on E and interact with all horizontal dimensions of E of the items i, $1 \le i \le L - h + 1$. The result of the interaction is the i-th convolution value given by

$$c^k_i = \phi_c(E_{i:i+h-1} \odot F^k). \tag{3}$$

where the symbol $\odot$ denotes the inner product operator and $\phi_c(\cdot)$ is the activation function for convolutional layers. This value is the inner product between F^k and the sub-matrix formed by the row i to row $i - h + 1$ of E, denoted by $E_{i:i+h-1}$. The final convolution result of F^k is the vector

$$c^k = \left[c^k_1\ c^k_2 \cdots c^k_{L-h+1}\right]. \tag{4}$$

We then apply a max pooling operation to c^k to extract the maximum value from all values produced by this particular filter. The maximum value captures the most significant feature extracted by the filter. Therefore, for the n filters in this layer, the output value $o \in \mathbb{R}^n$ is

$$o = \{max(c^1), max(c^2), \cdots, max(c^n)\}. \tag{5}$$

Horizontal filters interact with every successive h items through their embeddings E. Both the embeddings and the filters are learnt to minimize an objective function that encodes the prediction error of target items (more in Section 3.4). By sliding filters of various heights, a significant signal will be picked up regardless of location. Therefore, horizontal filters can be trained to capture **union-level patterns** with multiple union sizes.

Vertical Convolutional Layer. This layer is shown in the lower part of the second component in Figure 3. We use tilde (~) for the symbols of this layer. Suppose that there are $\tilde{n}$ vertical filters $\tilde{F}^k \in \mathbb{R}^{L \times 1}$, $1 \le k \le \tilde{n}$. Each filter $\tilde{F}^k$ interacts with the columns of E by sliding d times from left to right on E, yielding the vertical convolution result $\tilde{c}^k$:

$$\tilde{c}^k = \left[\tilde{c}^k_1\ \tilde{c}^k_2 \cdots \tilde{c}^k_d\right]. \tag{6}$$

For the inner product interaction, it is easy to verify that this result is equal to the weighted sum over the L rows of E with $\tilde{F}^k$ as the weights:

$$\tilde{c}^k = \sum_{l=1}^{L} \tilde{F}^k_l \cdot E_l, \tag{7}$$

where E_l is the l-th row of E. Therefore, with vertical filters we can learn to *aggregate* the embeddings of the L previous items, similar to Fossil's [6] weighted sum to aggregate the L previous items' latent representations. The difference is that each filter $\tilde{F}^k$ is acting like a different aggregator. Thus, similar to Fossil, these vertical filters are capturing **point-level sequential patterns** through weighted sums over previous items' latent representations. While Fossil uses a single weighted sum for each user, we can use $\tilde{n}$ global vertical filters to produce $\tilde{n}$ weighted sums $\tilde{o} \in \mathbb{R}^{d\tilde{n}}$ for all users:

$$\tilde{o} = \left[\tilde{c}^1\ \tilde{c}^2 \cdots \tilde{c}^{\tilde{n}}\right]. \tag{8}$$

Since their usage is aggregation, vertical filters have some differences from horizontal ones: (1) The size of each vertical filter is fixed to be $L \times 1$. This is because each column of E is latent for us, it is meaningless to interact with multiple successive columns at one time. (2) There is no need to apply max pooling operation over the vertical convolution results, as we want to keep the aggregation for every latent dimension. Thus, the output of this layer is $\tilde{o}$.

3.3 Fully-connected Layers

We concatenate the outputs of the two convolutional layers and feed them into a fully-connected neural network layer to get more

high-level and abstract features:

$$\boldsymbol{z} = \phi_a(\boldsymbol{W} \begin{bmatrix} \boldsymbol{o} \\ \tilde{\boldsymbol{o}} \end{bmatrix} + \boldsymbol{b}), \tag{9}$$

where $\boldsymbol{W} \in \mathbb{R}^{d \times (n+d\tilde{n})}$ is the weight matrix that projects the concatenation layer to a d-dimensional hidden layer, $\boldsymbol{b} \in \mathbb{R}^d$ is the corresponding bias term and $\phi_a(\cdot)$ is the activation function for fully-connected layer. $\boldsymbol{z} \in \mathbb{R}^d$ is what we called *convolutional sequence embedding*, which encodes all kinds of sequential features of the L previous items.

To capture user's general preferences, we also look-up the user embedding $\boldsymbol{P}_u$ and concatenate the two d-dimensional vectors, $\boldsymbol{z}$ and $\boldsymbol{P}_u$, together and project them to an output layer with $|I|$ nodes, written as

$$\boldsymbol{y}^{(u,t)} = \boldsymbol{W}' \begin{bmatrix} \boldsymbol{z} \\ \boldsymbol{P}_u \end{bmatrix} + \boldsymbol{b}', \tag{10}$$

where $\boldsymbol{b}' \in \mathbb{R}^{|I|}$ and $\boldsymbol{W}' \in \mathbb{R}^{|I| \times 2d}$ are the bias term and weight matrix for output layer, respectively. As explained in Section 3.4, the value $\boldsymbol{y}_i^{(u,t)}$ in the output layer is associated with the probability of how likely user u will interact with item i at time step t. $\boldsymbol{z}$ intends to capture short term sequential patterns, whereas the user embedding $\boldsymbol{P}_u$ captures user's long-term general preferences. Here we put the user embedding Pu in the last hidden layer for several reasons: (1) As we shall see in Section 3.6, it can have the ability to generalize other models. (2) we can pre-train our model's parameters with other generalized models' parameters. As stated in [7], such pre-training is critical to model performance

3.4 Network Training

To train the network, we transform the values of the output layer, $\boldsymbol{y}^{(u,t)}$, to probabilities by:

$$p(\mathcal{S}_t^u \mid \mathcal{S}_{t-1}^u, \mathcal{S}_{t-2}^u, \cdots, \mathcal{S}_{t-L}^u) = \sigma(\boldsymbol{y}_{\mathcal{S}_t^u}^{(u,t)}), \tag{11}$$

where $\sigma(x) = 1/(1+e^{-x})$ is the *sigmoid* function. Let $C^u = \{L+1, L+2, ..., |\mathcal{S}^u|\}$ be the collection of time steps for which we would like to make predictions for user u. The likelihood of all sequences in the dataset is:

$$p(\mathcal{S}|\Theta) = \prod_u \prod_{t \in C^u} \sigma(\boldsymbol{y}_{\mathcal{S}_t^u}^{(u,t)}) \prod_{j \neq \mathcal{S}_t^u} (1 - \sigma(\boldsymbol{y}_j^{(u,t)})). \tag{12}$$

To further capture **skip behaviors**, we could consider the next T target items, $\mathcal{D}_t^u = \{\mathcal{S}_t^u, \mathcal{S}_{t+1}^u, ..., \mathcal{S}_{t+T}^u\}$, at once by replacing the immediate next item $\mathcal{S}_t^u$ in the above equation with $\mathcal{D}_t^u$. Taking the negative logarithm of likelihood, we get the objective function, also known as *binary cross-entropy* loss:

$$\ell = \sum_u \sum_{t \in C^u} \sum_{i \in \mathcal{D}_t^u} -\log(\sigma(\boldsymbol{y}_i^{(u,t)})) + \sum_{j \neq i} -\log(1 - \sigma(\boldsymbol{y}_j^{(u,t)})). \tag{13}$$

Following previous works [6, 21, 32], for each target item i, we randomly sample several (3 in our experiments) negative instances j in the second term.

The *model parameters* $\Theta = \{\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{F}, \tilde{\boldsymbol{F}}, \boldsymbol{W}, \boldsymbol{W}', \boldsymbol{b}, \boldsymbol{b}'\}$ are learned by minimizing the objective function in Eqn (13) on the training set, whereas the *hyperparameters* (*e.g.,* $d, n, \tilde{n}, L, T$) are tuned on the validation set via grid search. We adopt an variant of Stochastic Gradient Descent (SGD) called Adaptive Moment Estimation (Adam) [13] for faster convergence, with a batch size of 100. To control model complexity and avoid over-fitting, we use two kinds of regularization methods: the $L2$ Norm is applied for all model parameters and *Dropout* [27] technique with 50% drop ratio is used on fully-connected layers. We implemented Caser with MatConvNet [28]. The whole training time is proportional to the number of training instances. For example, it took around 1 hour for MovieLens data and 2 hours for Gowalla data, 2 hours for Foursquare and 1 hour for Tmall on a 4-cores i7 CPU and 32GB RAM machine. These times are comparable to Fossil's [6] running time and can be further reduced by using GPU.

3.5 Recommendation

After obtaining the trained neural network, to make recommendations for a user u at time step t, we take u's latent embedding $\boldsymbol{P}_u$ and extract his last L items' embeddings given by Eqn (2) as the neural network input. We recommend the N items that have the highest values in the output layer $\boldsymbol{y}$. The complexity for making recommendations to all users is $O(|\mathcal{U}||I|d)$, where the complexity of convolution operations is ignored. Note that the number of target items T is a hyperparameter used during the model training, whereas N is the number of items recommended after the model is trained.

3.6 Connection to Existing Models

We show that Caser is a generalization of several previous models.

Caser vs. MF. By discarding all convolutional layers and all bias terms, our model becomes a vanilla LFM with user embeddings as user latent factors and its associated weights as item latent factors. MF usually contains bias terms[1], which is $\boldsymbol{b}'$ in our model. After discarding all convolutional layers, the resulting model is the same as MF:

$$\boldsymbol{y}_i^u = \boldsymbol{W}_i' \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{P}_u \end{bmatrix} + \boldsymbol{b}_i'. \tag{14}$$

Caser vs. FPMC. FPMC fuses factorized first-order Markov chain with LFM and is optimized by Bayesian personalized ranking (BPR). Although Caser uses a different optimization criterion, *i.e.,* the cross-entropy, it is able to generalize FPMC by copying the previous item's embedding to the hidden layer $\boldsymbol{z}$ and not using any bias terms:

$$\boldsymbol{y}_i^{(u,t)} = \boldsymbol{W}_i' \begin{bmatrix} \boldsymbol{Q}_{\mathcal{S}_{t-1}^u} \\ \boldsymbol{P}_u \end{bmatrix}. \tag{15}$$

As FPMC uses BPR as the criterion, our model is not exactly the same as FPMC. However, BPR is limited to have only 1 target and negative sample at each time step. Our cross-entropy loss does not have these limitations.

Caser vs. Fossil. By omitting the horizontal convolutional layer and using one vertical filter and copying the vertical convolution result $\tilde{\boldsymbol{c}}$ to the hidden layer $\boldsymbol{z}$, we get

$$\boldsymbol{y}_i^{(u,t)} = \boldsymbol{W}_i' \begin{bmatrix} \tilde{\boldsymbol{c}} \\ \boldsymbol{P}_u \end{bmatrix} + \boldsymbol{b}_i'. \tag{16}$$

As discussed for Eqn (7), this vertical filter serves as the weighted sum of the embeddings of the L previous items, like in Fossil, though Fossil uses *Similarity Model* instead of LFM and factorizes it in the

[1] Top-N recommendation ranks the items for each user individually, which is invariant to user bias and global bias.

same latent space as Markov model. Another difference is that Fossil uses one local weighting for each user while we use a number of global weighting through vertical filters.

4 EXPERIMENTS

We compare *Caser* with state-of-the-art methods. The source code of Caser and processed data sets are available online[2].

4.1 Experimental Setup

Datasets. Sequential recommendation makes sense only when the data set contains sequential patterns. To identify such data sets, we applied sequential association rule mining to several public data sets and computed their sequential intensity defined by:

$$\text{Sequential Intensity } (SI) = \frac{\#\text{rules}}{\#\text{users}}. \tag{17}$$

The numerator is the total number of rules in the form of Eqn (1) found using a minimum threshold on support (*i.e.,* 5) and confidence(*i.e.,* 50%) with Markov order L range from 1 to 5. The denominator is the total number of users. We use *SI* to estimate the intensity of sequential signals in a data set.

The four data sets with their *SI* are described in Table 1. MovieLens[3] is the widely used movie rating data. Gowalla[4] constructed by [3] and Foursquare obtained from [33] contain implicit feedback through user-venue check-ins. Tmall, the largest B2C platform in China, is a user-purchase data obtained from IJCAI 2015 competition[5], which aims to forecast repeated buyers. Following previous works [6, 20, 32], we converted all numeric ratings to implicit feedback of 1. We also removed cold-start users and items of having less than n feedbacks, as dealing with cold-start recommendation is usually treated as a separate issue in the literature [6, 7, 21, 32]. n is 5,15,10,10 for MovieLens, Gowalla, Foursquare, and Tmall. The Amazon data previously used in [5, 6] was not used due to its *SI* (0.0026 for 'Office Products' category, 0.0019 for 'Clothing, Shoes, Jewelry' and 'Video Games' category), in other words, its sequential signals are much weaker than the above data sets.

Following [17, 33, 35], we hold the first 70% of actions in each user's sequence as the *training set* and use the next 10% of actions as the *validation set* to search the optimal hyperparameter settings for all models. The remaining 20% actions in each user's sequence are used as the *test set* for evaluating a model's performance.

Evaluation Metrics. As in [19, 21, 29, 32], we evaluate a model by Precision@N, Recall@N, and Mean Average Precision (MAP). Given a list of top N predicted items for a user, denoted $\hat{R}_{1:N}$, and the last 20% of actions in her/his sequence (*i.e.*, denoted R (*i.e.*, the test set), Precision@N and Recall@N are computed by

$$\begin{aligned} \text{Prec@}N &= \frac{|R \bigcap \hat{R}_{1:N}|}{N}, \\ \text{Recall@}N &= \frac{|R \bigcap \hat{R}_{1:N}|}{|R|}. \end{aligned} \tag{18}$$

We report the average of these values of all users. $N \in \{1, 5, 10\}$. The Average Precision (AP) is defined by

$$\text{AP} = \frac{\sum_{N=1}^{|\hat{R}|} \text{Prec@}N \times \text{rel}(N)}{|\hat{R}|}, \tag{19}$$

where $rel(N) = 1$ if the N-th item in $\hat{R}$ is in R. The Mean Average Precision (MAP) is the average of AP for all users.

4.2 Performance Comparison

We compare our method, Caser, proposed in Section 3 with the following baselines.

- **POP**. All items are ranked by their popularity in all users' sequences, and the popularity is determined by the number of interactions.
- **BPR**. Combined with Matrix Factorization model, Bayesian personalized ranking [20] is the state-of-the-art method for non-sequential item recommendation on implicit feedback data.
- **FMC and FPMC**. As introduced in [21], FMC factorizes the first-order Markov transition matrix into two low-dimensional sub-matrices, and FPMC is a fusion of FMC and LFM. These are the state-of-the-art sequential recommendation methods. FPMC allows a basket of several items at each step. For our sequential recommendation problem, each basket has a single item.
- **Fossil**. Fossil [6] models high-order Markov chains and uses Similarity Model instead of LFM for modeling general user preferences.
- **GRU4Rec**. This is the session-based recommendation proposed by [8]. This model uses RNN to capture sequential dependencies and make predictions.

For each method, the grid search is applied to find the optimal settings of hyperparameters using the validation set. These include latent dimensions d from $\{5, 10, 20, 30, 50, 100\}$, regularization hyperparameters, and the learning rate from $\{1, 10^{-1}, ..., 10^{-4}\}$. For Fossil, Caser and GRU4Rec, the Markov order L is from $\{1, \cdots, 9\}$. For Caser itself, the height h of horizontal filters is from $\{1, \cdots, L\}$, the target number T is from $\{1, 2, 3\}$, the activation functions ϕ_a and ϕ_c are from $\{identity, sigmoid, tanh, relu\}$. For each height h, the number of horizontal filters is from $\{4, 8, 16, 32, 64\}$. The number of vertical filters is from $\{1, 2, 4, 8, 16\}$. We report the result of each method under its optimal hyperparameter settings.

The best results of the six baselines and Caser are summarized in Table 2. The best performer on each row is highlighted in bold face. The last column is the improvement of Caser relative to the best baseline, defined as $\frac{Caser-baseline}{baseline}$. Except for MovieLens, Caser improved the best baseline on all N tested by a large margin *w.r.t.* the three metrics. Among the baseline methods, the sequential recommenders (*e.g.,* FPMC and Fossil) usually outperform non-sequential recommenders (*i.e.,* BPR) on all data sets, suggesting the importance of considering sequential information. FPMC and Fossil outperform FMC on all data sets, suggesting the effectiveness of personalization. On MovieLens, GRU4Rec achieved a performance close to Caser's, but got a much worse performance on the other three data sets. In fact, MovieLens has more sequential signals than

[2] https://github.com/graytowne/caser
[3] https://grouplens.org/datasets/movielens/1m/
[4] https://snap.stanford.edu/data/loc-gowalla.html
[5] https://ijcai-15.org/index.php/repeat-buyers-prediction-competition

Table 1: Statistics of the datasets

Datasets	Sequential Intensity	#users	#items	avg. actions per user	Sparsity
MovieLens	0.3265	6.0k	3.4k	165.50	95.16%
Gowalla	0.0748	13.1k	14.0k	40.74	99.71%
Foursquare	0.0378	10.1k	23.4k	30.16	99.87%
Tmall	0.0104	23.8k	12.2k	13.93	99.89%

Table 2: Performance comparison on the four data sets.

Dataset	Metric	POP	BPR	FMC	FPMC	Fossil	GRU4Rec	Caser	Improv.
MovieLens	Prec@1	0.1280	0.1478	0.1748	0.2022	0.2306	**0.2515**	0.2502	-0.5%
	Prec@5	0.1113	0.1288	0.1505	0.1659	0.2000	0.2146	**0.2175**	1.4%
	Prec@10	0.1011	0.1193	0.1317	0.1460	0.1806	0.1916	**0.1991**	4.0%
	Recall@1	0.0050	0.0070	0.0104	0.0118	0.0144	**0.0153**	0.0148	-3.3%
	Recall@5	0.0213	0.0312	0.0432	0.0468	0.0602	0.0629	**0.0632**	0.5%
	Recall@10	0.0375	0.0560	0.0722	0.0777	0.1061	0.1093	**0.1121**	2.6%
	MAP	0.0687	0.0913	0.0949	0.1053	0.1354	0.1440	**0.1507**	4.7%
Gowalla	Prec@1	0.0517	0.1640	0.1532	0.1555	0.1736	0.1050	**0.1961**	13.0%
	Prec@5	0.0362	0.0983	0.0876	0.0936	0.1045	0.0721	**0.1129**	8.0%
	Prec@10	0.0281	0.0726	0.0657	0.0698	0.0782	0.0571	**0.0833**	6.5%
	Recall@1	0.0064	0.0250	0.0234	0.0256	0.0277	0.0155	**0.0310**	11.9%
	Recall@5	0.0257	0.0743	0.0648	0.0722	0.0793	0.0529	**0.0845**	6.6%
	Recall@10	0.0402	0.1077	0.0950	0.1059	0.1166	0.0826	**0.1223**	4.9%
	MAP	0.0229	0.0767	0.0711	0.0764	0.0848	0.0580	**0.0928**	9.4%
Foursquare	Prec@1	0.1090	0.1233	0.0875	0.1081	0.1191	0.1018	**0.1351**	13.4%
	Prec@5	0.0477	0.0543	0.0445	0.0555	0.0580	0.0475	**0.0619**	6.7%
	Prec@10	0.0304	0.0348	0.0309	0.0385	0.0399	0.0331	**0.0425**	6.5%
	Recall@1	0.0376	0.0445	0.0305	0.0440	0.0497	0.0369	**0.0565**	13.7%
	Recall@5	0.0800	0.0888	0.0689	0.0959	0.0948	0.0770	**0.1035**	7.9%
	Recall@10	0.0954	0.1061	0.0911	0.1200	0.1187	0.1011	**0.1291**	7.6%
	MAP	0.0636	0.0719	0.0571	0.0782	0.0823	0.0643	**0.0909**	10.4%
Tmall	Prec@1	0.0010	0.0111	0.0197	0.0210	0.0280	0.0139	**0.0312**	11.4%
	Prec@5	0.0009	0.0081	0.0114	0.0120	0.0149	0.0090	**0.0179**	20.1%
	Prec@10	0.0007	0.0063	0.0084	0.0090	0.0104	0.0070	**0.0132**	26.9%
	Recall@1	0.0004	0.0046	0.0079	0.0082	0.0117	0.0056	**0.0130**	11.1%
	Recall@5	0.0019	0.0169	0.0226	0.0245	0.0306	0.0180	**0.0366**	19.6%
	Recall@10	0.0026	0.0260	0.0333	0.0364	0.0425	0.0278	**0.0534**	25.6%
	MAP	0.0030	0.0145	0.0197	0.0212	0.0256	0.0164	**0.0310**	21.1%

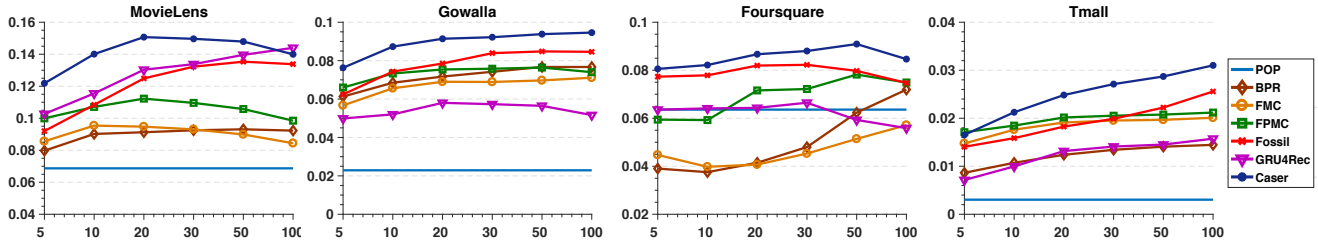


Figure 5: MAP (y-axis) vs. the number of latent dimensions d (x-axis).

the other three data sets, thus, the RNN-based GRU4Rec could perform well on MovieLens but can easily get biased on training sets of the other three data sets despite the use of regularization and dropout as described in [8]. In addition, GRU4Rec's recommendation is session-based, instead of personalized, which enlarge the generalization error to some extent.

In the following studies, we examine the impact of the hyperparameters d, L, T one at a time by holding the remaining hyperparameters at their optimal settings. We focus on MAP as it is an overall performance indicator and consistent with other metrics.

4.2.1 Influence of Latent Dimensionality d. Figure 5 shows MAP for various d while keeping the other optimal hyperparameters unchanged. On the denser MovieLens, a larger d does not always lead to a better model performance. A model achieves its best performance when d is chosen properly and gets worse for a larger d because of over-fitting. But for the other three sparser data sets, each model requires more latent dimensions to achieve their best results. For all data sets, Caser beats the strongest baseline performance by using a relatively small number of latent dimensions.

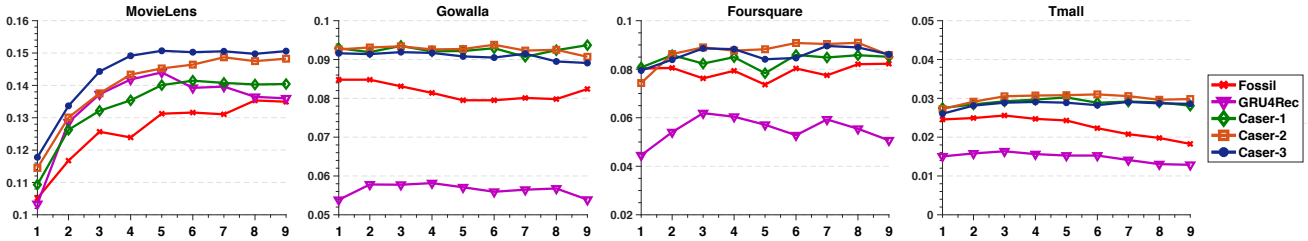


Figure 6: MAP (y-axis) vs. the Markov order L (x-axis). Caser-1, Caser-2, and Caser-3 denote Caser with the number of targets T set to $1, 2, 3$.

4.2.2 Influence of Markov Order L and Target Number T. We vary L to explore how much of Fossil, GRU4Rec and Caser can gain from high-order information while keeping other optimal hyperparameters unchanged. Caser-1, Caser-2, and Caser-3 denote Caser with the target number T at 1, 2, 3 to study the effect of skip behaviors. The results are shown in Figure 6. On the dense MovieLens, Caser best utilizes the extra information provided by a larger L and Caser-3 performs the best, suggesting the benefits of skip steps. However, for the sparser data sets, all models do not consistently benefit from a larger L. This is reasonable, because for a sparse data set, a higher order Markov chain tends to introduce both extra information and more noises. In most cases, Caser-2 slightly outperforms the other models on these three data sets.

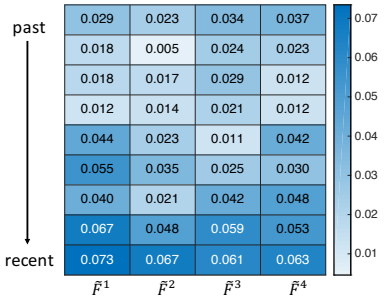


Figure 7: Visualization for four vertical convolutional filters of a trained model on MovieLens data when $L = 9$.

Table 3: MAP vs. Caser Components

	MovieLens	Gowalla
Caser-p	0.0935	0.0777
Caser-h	0.1304	0.0805
Caser-v	0.1403	0.0841
Caser-vh	0.1448	0.0856
Caser-ph	0.1372	0.0911
Caser-pv	0.1494	0.0921
Caser-pvh	0.1507	0.0928

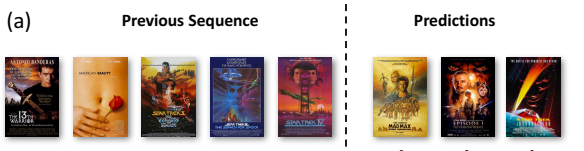
4.2.3 Analysis of Caser Components. Finally, we evaluate the contribution of each of Caser's components, the horizontal convolutional layer (*i.e.*, o), the vertical convolutional layer (*i.e.*, $\tilde{o}$), and personalization (*i.e.*, P_u), to the overall performance while keeping all hyperparameters at their optimal settings. The result is shown in Table 3 for MovieLens and Gowalla; the results of the other two data sets are similar. For $x \in \{p, h, v, vh, ph, pv, pvh\}$, Caser-$x$ denotes Caser with the components x enabled. h denotes horizontal convolutional layer; v denotes vertical convolutional layer; p denotes personalization, which is similar to BPR and uses LFM only. Any missing component is represented by setting its corresponding o, $\tilde{o}$, and P_u to zero. For example, vh denotes both vertical convolutional layer and horizontal convolutional layer by setting P_u to all zeros, and pv denotes vertical convolutional layer and personalization by setting o to all zeros. Caser-p performs the worst whereas Caser-h, Caser-v, and Caser-vh improve the performance significantly, suggesting that treating top-N sequential recommendation as the conventional top-N recommendation will lose useful information, and that modeling both sequential patterns at the union-level and point-level is useful for improving the prediction. For both data sets, the best performance is achieved by jointly using all parts of Caser, i.e., Caser-pvh.

4.3 Network Visualization

We have a closer look at some trained networks and prediction. Figure 7 shows the values of four vertical convolutional filters after training Caser on MovieLens with $L = 9$. In the micro perspective, the four filters are trained to be diverse, but in the macro perspective, they follow an ascending trend from past positions to recent positions. With each vertical filter serving as a way of weighting the embeddings of previous actions (see the related discussion in Section 3), this trend indicates that Caser puts more emphasis on recent actions, demonstrating a major difference from the conventional top-N recommendation.

To see the effectiveness of horizontal filters, Figure 8(a) shows top $N = 3$ ranked movies recommended by Caser, *i.e.*, $\hat{R}_1$ (Mad Max), $\hat{R}_2$ (Star War), $\hat{R}_3$ (Star Trek) in that order, for a user with $L = 5$ previous movies, *i.e.*, S_1 (13th Warrior), S_2 (American Beauty), S_3 (Star Trek), S_4 (Star Trek III), and S_5 (Star Trek IV). $\hat{R}_3$ is the ground truth (*i.e.*, the next movie in the user sequence). Note that $\hat{R}_1$ and $\hat{R}_2$ are quite similar to $\hat{R}_3$, *i.e.*, all being action and science fiction movies, so are also recommended to the user. Figure 8(b) shows the new rank of $\hat{R}_3$ after masking some of the L previous movies by setting their item embeddings to zeros in the trained network. Masking S_1 and S_2 actually increases the rank of $\hat{R}_3$ to 2 (from 3); in fact, S_1 and S_2 are history or romance movies and act like noises for recommending $\hat{R}_3$. Masking each of S_3, S_4 and S_5 decreases the rank of $\hat{R}_3$ because these movies are in the same category as $\hat{R}_3$. The most decrease occurs after masking S_3, S_4 and S_5 all together. This study clearly indicates that our model correctly captures the dependence

(a)



(b)

Masking Items	New Rank of $\hat{R}_3$ after masking
S_1, S_2	2
S_3	32
S_4	117
S_5	77
S_3, S_4, S_5	513

Figure 8: Horizontal convolutional filters's effectiveness of capturing union-level sequential patterns on MovieLens data.

of $\hat{R}_3$ on the related $\{S_3, S_4, S_5\}$ as a union-level sequential feature for recommending $\hat{R}_3$.

5 CONCLUSION

Caser is a novel solution to top-N sequential recommendation by modeling recent actions as an "image" among time and latent dimensions and learning sequential patterns using convolutional filters. This approach provides a unified and flexible network structure for capturing many important features of sequential recommendation, *i.e.*, point-level and union-level sequential patterns, skip behaviors, and long term user preferences. Our experiments and case studies on public real life data sets suggested that Caser outperforms the state-of-the-art methods for top-N sequential recommendation.

ACKNOWLEDGEMENT

The work of the second author is partially supported by a Discovery Grant from Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. 1995. Mining sequential patterns. In *International Conference on Data Engineering*. IEEE, 3–14.
[2] Chen Cheng, Haiqin Yang, Michael R Lyu, and Irwin King. 2013. Where You Like to Go Next: Successive Point-of-Interest Recommendation.. In *International Joint Conference on Artificial Intelligence*. 2605–2611.
[3] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *International Conference on Knowledge Discovery and Data Mining*. ACM, 1082–1090.
[4] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques*. Elsevier.
[5] R. He, W.-C. Kang, and J. McAuley. 2017. Translation-based recommendation. In *ACM Conference on Recommender systems*.
[6] R. He and J. McAuley. 2016. Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation. In *International Conference on Data Mining*. IEEE.
[7] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *International Conference on World Wide Web*. ACM, 173–182.
[8] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).
[9] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *International Conference on Data Mining*. IEEE, 263–272.
[10] Dietmar Jannach and Malte Ludewig. 2017. When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation. In *ACM Conference on Recommender systems*. ACM, 306–310.
[11] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale video classification with convolutional neural networks. In *IEEE conference on Computer Vision and Pattern Recognition*. 1725–1732.
[12] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Conference on Empirical Methods on Natural Language Processing*. ACL, 1756–1751.
[13] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[14] Yehuda Koren. 2010. Collaborative filtering with temporal dynamics. *Commun. ACM* 53, 4 (2010), 89–97.
[15] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).
[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.
[17] Duen-Ren Liu, Chin-Hui Lai, and Wang-Jung Lee. 2009. A hybrid of sequential rules and collaborative filtering for product recommendation. *Information Sciences* 179, 20 (2009), 3505–3519.
[18] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model.. In *Interspeech*, Vol. 2. 3.
[19] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. 2008. One-class collaborative filtering. In *International Conference on Data Mining*. IEEE, 502–511.
[20] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 452–461.
[21] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *International Conference on World Wide Web*. ACM, 811–820.
[22] Ruslan Salakhutdinov and Andriy Mnih. 2007. Probabilistic Matrix Factorization.. In *Advances in Neural Information Processing Systems*, Vol. 1. 2–1.
[23] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann machines for collaborative filtering. In *International Conference on Machine learning*. ACM, 791–798.
[24] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *International Conference on World Wide Web*. ACM, 285–295.
[25] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *International Conference on World Wide Web*. ACM, 111–112.
[26] Yang Song, Ali Mamdouh Elkahky, and Xiaodong He. 2016. Multi-rate deep learning for temporal recommendation. In *International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 909–912.
[27] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
[28] Andrea Vedaldi and Karel Lenc. 2015. Matconvnet: Convolutional neural networks for matlab. In *International conference on Multimedia*. ACM, 689–692.
[29] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *International Conference on Knowledge Discovery and Data Mining*. ACM, 1235–1244.
[30] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2015. Learning hierarchical representation model for nextbasket recommendation. In *International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 403–412.
[31] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. 2017. Recurrent Recommender Networks. In *International Conference on Web Search and Data Mining*. ACM, 495–503.
[32] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *International Conference on Web Search and Data Mining*. ACM, 153–162.
[33] Quan Yuan, Gao Cong, and Aixin Sun. 2014. Graph-based point-of-interest recommendation with geographical and temporal influences. In *International Conference on Information and Knowledge Management*. ACM, 659–668.
[34] Chenyi Zhang, Ke Wang, Hongkun Yu, Jianling Sun, and Ee-Peng Lim. 2014. Latent factor transition for dynamic collaborative filtering. In *SIAM International Conference on Data Mining*. SIAM, 452–460.
[35] Shenglin Zhao, Tong Zhao, Haiqin Yang, Michael R Lyu, and Irwin King. 2016. Stellar: spatial-temporal latent ranking for successive point-of-interest recommendation. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 315–321.
[36] Lei Zheng, Vahid Noroozi, and Philip S. Yu. 2017. Joint Deep Modeling of Users and Items Using Reviews for Recommendation. In *International Conference on Web Search and Data Mining*. ACM, 425–434.