A Simple but Hard-to-Beat Baseline for Session-based Recommendations

Fajie Yuan[*]
University of Glagow
Glasgow, UK
f.yuan.1@research.gla.ac.uk

Alexandros Karatzoglou
Telefonica Research
Barcelona, Spain
alexandros.karatzoglou@gmail.com

Ioannis Arapakis
Telefonica Research
Barcelona, Spain
arapakis.ioannis@gmail.com

Joemon M Jose
University of Glagow
Glasgow, UK
joemon.jose@glasgow.ac.uk

Xiangnan He
National University of Singapore
Singapore
xiangnanhe@gmail.com

ABSTRACT

Convolutional Neural Networks (CNNs) models have been recently introduced in the domain of top-N session-based recommendations. An ordered collection of past items the user has interacted with in a session (or sequence) are embedded into a 2-dimensional latent matrix, and treated as an image. The convolution and pooling operations are then applied to the mapped item embeddings. In this paper, we first examine the typical session-based CNN recommender and show that both the generative model and network architecture are suboptimal when modeling long-range dependencies in the item sequence. To address the issues, we introduce a simple, but very effective generative model that is capable of learning high-level representation from both short- and long-range item dependencies. The network architecture of the proposed model is formed of a stack of *holed* convolutional layers, which can efficiently increase the receptive fields without relying on the pooling operation. Another contribution is the effective use of residual block structure in recommender systems, which not only reduces the number of parameters but also eases the optimization for much deeper networks. The proposed generative model attains state-of-the-art accuracy with less training time in the session-based recommendation task. It accordingly can be used as a powerful recommendation baseline to beat in future, especially when there are long sequences of user feedback.

1 INTRODUCTION

Leveraging sequences of item interactions (e.g., clicks or purchases) to improve real-world recommender systems has become increasingly popular in recent years. These sequences are automatically generated when users interact with online systems in sessions (e.g., shopping session, or music listening session). For example, users on Last.fm[1] or Soundcloud[2] typically enjoy a series of songs during a certain time period without any interruptions, i.e., a listening session. The set of songs played in one session usually have strong correlations [6], e.g., sharing the same album, artist, or genre. Accordingly, a good recommender system is supposed to generate recommendations by taking advantage of these sequential patterns in the session.
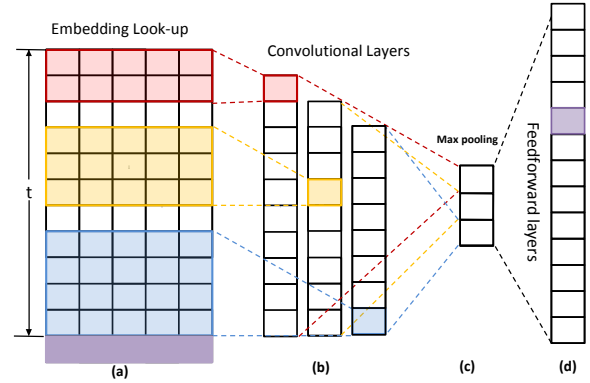
---
[*]The work was done when the first author was an intern in Telefonica Research, Spain.
[1]https://www.last.fm
[2]https://www.soundcloud.com

Figure 1: The basic structure of *Caser* [27]. The red, yellow and blue regions denotes a $2 \times k$, $3 \times k$ and $4 \times k$ convolution filter respectively, where $k = 5$. The purple row stands for the true next item.

A class of models often employed for these sequences of item interactions are the Recurrent Neural Networks (RNNs). RNNs typically generate a softmax output where high probabilities represent the most relevant recommendations. While effective, these RNN-based models, such as [2, 14, 22], depend on a hidden state of the entire past that cannot fully utilize parallel computation within a sequence [8]. Thus their speed is limited in both training and evaluation.

By contrast, training CNNs does not depend on the computations of the previous time step and therefore allow parallelization over every element in a sequence. Inspired by the successful use of CNNs in image tasks, a newly proposed sequential recommender, referred to as *Caser* [27], abandoned RNN structures, proposing instead a convolutional sequence embedding model, and demonstrated that this CNN-based recommender is able to achieve comparable or superior performance to the popular RNN model in the top-N sequential recommendation task. The basic idea of the convolution processing is to treat the $t \times k$ embedding matrix as the "image" of the previous t items in k dimensional latent space and regard the sequential pattens as local features of the "image". A max pooling operation that only preserves the maximum value of the convolutional layer is performed to increase the receptive field, as well as dealing with the varying length of input sequences. Fig. 1 depicts the key architecture of *Caser*.

Considering the training speed of networks, in this paper we follow the path of sequential convolution techniques for the next

item recommendation task. We show that the typical network architecture used in *Caser* has several obvious drawbacks — e.g.,: (1) the max pooling scheme that is safely used in computer vision may discard important position and recurrent signals when modeling long-range sequence data; (2) generating the softmax distribution only for the desired item fails to effectively use the compete set of dependencies. Both drawbacks become more severe as the length of the sessions and sequences increases. To address these issues, we introduce a simple but fundamentally different CNN-based sequential generative model that allows us to model the complex conditional distributions even in very long-range item sequences. To be more specific, first our generative model is designed to explicitly encode item inter-dependencies, which allows to directly estimates the distribution of the output sequence (rather than the desired item) over the raw item sequence. Second, instead of using inefficient huge filters, we stack the 1D dilated convolutional layers [30] on top of each other to increase the receptive fields when modeling long-range dependencies. The pooling layer can be safely removed in the proposed network structure. It is worth noting that although the dilated convolution was invented for dense prediction in image generation tasks [3, 24, 30], and has been applied in other fields (e.g., acoustic [20, 24] and text [16] tasks), it is yet unexplored in recommender systems with huge sparse data. Furthermore, to ease the optimization of the deep generative architecture, we design a residual learning structure to wrap each convolutional layer by residual block. To the best of our knowledge, this is the first work in terms of adopting residual learning to model the recommendation task. The combination of these choices enables us to tackle large-scale problems and attain state-of-the-art results in both short- and long-range sequential recommendation data sets. In summary, our main contributions in this paper include a novel probabilistic generative model (Section 3.1) and a fundamentally different convolutional network architecture (Sections 3.2 ∼ 3.4).

2 PRELIMINARIES

First, the problem of recommending items from sequences is described. Next, a recent convolutional sequence embedding recommendation model (*Caser*) is shortly recapitulated along with its limitations. Lastly, we review previous work on sequence-based recommender systems.

2.1 Top-N Session-based Recommendation

Let $\{x_0, x_1, ..., x_{t-1}, x_t\}$ (interchangeably denoted by $x_{0:t}$) be a clicked item sequence (or a session), where $x_i \in \mathbb{R}$ $(0 \leq i \leq t)$ is the index of the clicked item out of a total number of $t + 1$ items in the sequence. The term "item" here can represent more general instances, such as a music track, a query, an action and even a person. The goal of sequential recommendation is to seek a model such that for a given prefix item sequence, $\mathbf{x} = \{x_0, ..., x_i\}$ $(0 \leq i < t)$, it generates a ranking or classification distribution $\mathbf{y}$ for all candidate items, where $\mathbf{y} = [y_1, ..., y_n] \in \mathbb{R}^n$. y_j can be a score, probability or a rank of item $i+1$ that will occur in this sequence. In practice, we typically make more than one recommendation by choosing the top-N items from $\mathbf{y}$, referred to as the top-N sequential recommendations.

2.2 Limitations of *Caser*

The basic idea of *Caser* is to embed the previous t items as a $t \times k$ matrix $\mathbf{E}$ by the embedding look-up operation, as shown in Fig. 1 (a). Each row vector of the matrix corresponds to the latent features of one item. The embedding matrix can be regarded as the "image" of the t items in the k-dimensional latent space. Intuitively, models of various CNNs that are successfully applied in computer vision can be adapted to model the "image" of an item sequence. However, there are two aspects that differentiate sequence modeling from image processing, which makes the use of CNN based models non-straightforward. First, the variable-length item sequences in real-world scenarios produce a large number of "images" of different sizes, where traditional convolutional structures with fix-sized filters may fail. Second, the most effective filters for images, such as 3×3 and 5×5, are not suitable for sequence "images" since these small filters (in terms of row-wise orientation) are not suitable to capture the representations of full-width embedding vectors.

To address the above limitations, filters in *Caser* slide over full columns of the sequence "image" by large filter. That is, the width of filters is usually the same as the width of the input "images". The height typically varies by sliding windows over $2 - 5$ items at a time (Fig. 1 (a)). Filters of different sizes will generate variable-length feature maps after convolution (Fig. 1 (b)). To ensure that all maps have the same size, the max pooling is performed over each map, which selects only the largest number of each feature map, resulting in a 1×1 map (Fig. 1 (c)). Finally, these 1×1 maps from all filters are concatenated to form a feature vector, followed by a softmax layer that yields the probabilities of next item (Fig. 1 (d)). Note that we have omitted the vertical convolution in Fig. 1, since it does not solve the major problems discussed below. However, in our experiments, we report final results of *Caser* with both horizontal and vertical convolutions, although the vertical one results in a very slight accuracy gain.

Based on the above analysis of the convolutions in *Caser*, one may find that there exist several drawbacks with the current design. First, the max pooling operator has obvious disadvantages. It cannot distinguish whether an important feature in the map occurs just one or multiple times and it ignores the position in which it occurs. The max pooling operator while safely used in image processing (with small pooling filters, e.g., 3×3) may be harmful for modeling long-range sequences (with large filters, e.g., 1×20). Second, the shallow network structure in *Caser* that suits for only one hidden convolutional layer is likely to fail when modeling complex relations or long-range dependences. The last important disadvantage comes from the generative process of next item, which we will describe in detail in Section 3.1.

2.3 Related Work

Early work in sequential recommendations mostly rely on the markov chain ([5, 23, 29]) and feature-based matrix factorization ([4, 31, 32]) approaches. Compared with neural network models, the markov chain based approaches fail to model complicated relations in the sequence data. For example, in *Caser*, the authors showed that markov chain approaches failed to model union-level sequential patterns and did not allow skip behaviors in the item sequences. Factorization based approaches such as factorization
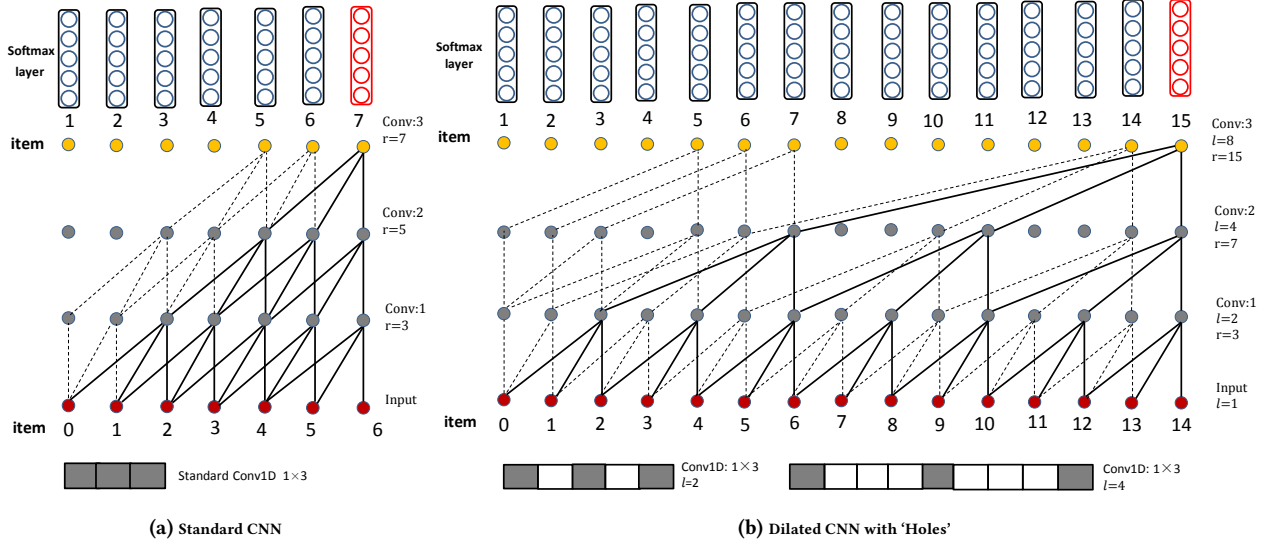
Figure 2: The proposed generative architecture with 1D standard CNNs (a) and efficient dilated CNNs (b). An example of a standard 1D convolution filter and dilated filters are shown at the bottom of (a) and (b) respectively. We will refer to a dilated convolution with a dilation factor l as l-dilated convolution. Apparently, compared with the standard CNN that linearly increases the receptive field by the depth of the network, the dilated CNN has a much larger receptive field by the same stacks without introducing more parameters. It can be seen that the standard convolution is a special case of 1-dilated convolution.

machines model a sequence by the sum of its item vectors. However, these methods do not consider the order of items and are not specifically invented for sequential recommendations.

Recently, deep learning models have shown state-of-the-art recommendation accuracy in contrast to conventional models. Moreover, RNNs, a class of deep neural networks, have almost dominated the area of sequential recommendations. For example, a Gated Recurrent Unit (*GRURec*) architecture with a ranking loss was proposed by [14] for session-based recommendation. In the follow-up papers, various RNN variants have been designed to extend the typical one for different application scenarios, such as by adding personalization [22], content [10] and contextual features [25], attention mechanism [7, 18, 21] and different ranking loss functions [13].

By contrast, CNN based sequential recommendation models are more challenging and much less explored because convolutions are not a natural way to capture sequential patterns. To our best knowledge, only two types of sequential recommendation architectures have been proposed to date: the first one by *Caser* is a standard 2D CNN, while the second is a 3D CNN [28] designed to model high-dimensional features. Unlike the aforementioned examples, we plan to investigate the effects of 1D CNNs with efficient dilated convolution filters and residual blocks for building the architecture of our generative model.

3 MODEL DESIGN

To address the above limitations, we introduce a new probabilistic generative model that is formed of a stack of 1D convolution layers. We first focus on the form of the distribution, and then the architectural innovations. Generally, our proposed model is fundamentally

different from *Caser* in several key ways: (1) our probability estimator explicitly models the distribution transition of all individual items at once, rather than the final one, in the sequence; (2) our network has a deep, rather than shallow, structure; (3) our convolutional layers are based on the efficient 1D dilated convolution rather than standard 2D convolution; and (4) pooling layers are not used.

3.1 A Simple Generative Model

In this section, we introduce a simple yet very effective generative model directly operating on the sequence of previous items. Our aim is to estimate a distribution over the original item sequences that can be used to *tractably* compute the likelihood of items and to generate the future items. Let $p(\mathbf{x})$ be the joint distribution of item sequence $\mathbf{x} = \{x_0, ..., x_t\}$. To model $p(\mathbf{x})$, we can factorize it as a product of conditional distributions by the chain rule.

$$p(\mathbf{x}) = \prod_{i=1}^{t} p(x_i|x_{0:i-1}, \boldsymbol{\theta})p(x_0) \tag{1}$$

where the value $p(x_i|x_{0:i-1}, \boldsymbol{\theta})$ is the probability of i-th item x_i conditioned on all the previous items $x_{0:i-1}$. To model the highly complicated function in Eq. (1), an expressive sequence model is necessary.

Owing to the ability of neural networks in modeling complex nonlinear relations, in this paper we model the conditional distributions by a stack of 1D convolutional networks. To be more specific, the network receives $x_{0:t-1}$ as the input and outputs distributions over possible $x_{1:t}$, where the distribution of x_t is our final expectation. For example, as illustrated in Fig. 2, the output distribution of x_{15} is determined by $x_{0:14}$, while x_{14} is determined by $x_{0:13}$. It is

worth noting that in previous sequential recommendation literatures, such as *Caser*, *GRURec* and [18, 22, 26, 28], they only model a single conditional distribution $p(x_i|x_{0:i-1}, \boldsymbol{\theta})$ rather than all conditional probabilities $\prod_{i=1}^{t} p(x_i|x_{0:i-1}, \boldsymbol{\theta})p(x_0)$. Within the context of the above example, assuming $\{x_0, ..., x_{14}\}$ is given, models like *Caser* only estimate the probability distribution (i.e., softmax) of the next item x_{15} (also see Fig. 1 (d)), while our generative method estimates the distributions of all individual items in $\{x_1, ..., x_{15}\}$. The comparison of the generating process is shown below.

$$Caser/GRURec : \underbrace{\{x_0, x_1, ..., x_{14}\}}_{input} \Rightarrow \underbrace{x_{15}}_{output}$$
$$Ours : \underbrace{\{x_0, x_1, ..., x_{14}\}}_{input} \Rightarrow \underbrace{\{x_1, x_2, ..., x_{15}\}}_{output} \quad (2)$$

where $\Rightarrow$ denotes 'predict'. Clearly, our proposed model is more effective in capturing the set of all sequence relations, whereas *Caser* and *GRURec* fail to *explicitly* model the internal sequence features between $\{x_0, ..., x_{14}\}$. In practice, to address the drawback, such models will typically generate a number of sub-sequences (or sub-sessions) for training by means of data augmentation techniques [26] (e.g., padding, splitting or shifting the input sequence), such as shown in Eq. (3) (see [18, 22, 27, 28]).

$$Caser/GRURec \; sub-session-1 : \{x_{-1}, x_0, ..., x_{13}\} \Rightarrow x_{14}$$
$$Caser/GRURec \; sub-session-2 : \{x_{-1}, x_{-1}, ..., x_{12}\} \Rightarrow x_{13}$$
$$...... \quad (3)$$
$$Caser/GRURec \; sub-session-12 : \{x_{-1}, x_{-1}, ..., x_2\} \Rightarrow x_3$$

While effective, the above approach to generate sub-session cannot guarantee the optimal results due to the separate optimization for each sub-session. In addition, optimizing these sub-sessions separately will result in corresponding computational costs. Detailed comparison with empirical results has also been reported in our experimental sections.

In summary, our generative model has three properties: (1) similar as *Caser* and *GRURec*, it is also autoregressive in the target item (as opposed to GANs [9]); (2) different from *Caser* and *GRURec*, it is capable of explicitly modeling the complete set of sequential dependencies in a natural way without resorting to more additional sub-sequences; and (3) different from *Caser*, it is sensitive to the ordering of the items in both input and output. An additional advantage is that pooling layers are not used in our architecture, and thus the output of the convolutional layer is guaranteed to have the same spatial dimensions as the input. The unchanged dimensionality makes layer-wise stacking and residual learning much easier, which is detailed later.

3.2 Network Architecture

3.2.1 Embedding Look-up Layer: Given an item sequence $\{x_0, ..., x_t\}$, the model retrieves each of the first t items $\{x_0, ..., x_{t-1}\}$ via a look-up table, and stacks these item embeddings together. Assuming the embedding dimension is $2k$, where k will be set as the number of inner channels in the convolutional network. This results in a matrix of size $t \times 2k$. Note that unlike *Caser* that treats the input

matrix as a 2D "image" during convolution, our proposed architecture learns the embedding layer by 1D convolutional filters, which we will describe later.

3.2.2 Dilated layer: As shown in Fig. 2 (a), the standard filter is only able to perform convolution with the receptive field linearly by the depth of the network. This makes it difficult to handle long-range sequences. Hence, instead of using standard convolutions, we employ the dilated convolution [30] to construct the proposed generative model. The basic idea of dilation is to apply the convolutional filter over a field larger than its original length by dilating it with zeros. As such, it is more efficient since it utilizes fewer parameters. For this reason, a dilated filter is also referred to as a *holed* or *sparse* filter. Another benefit is that dilated convolution can preserve the spatial dimensions of the input, which makes the stacking operation much easier for both convolutional layers and residual structures.

Fig. 2 shows the network comparison between the standard convolution and dilated convolutions with the proposed sequential generative model. The dilation factor in (b) are 1, 2, 4 and 8. To describe the network architecture, we denote receptive field, j-th convolutional layer, channel and dilation as r, F_j, C and l respectively. By setting the width of convolutional filter f as 3, we can see that the dilated convolutions (Fig. 2 (b)) allow for exponential increase in the size of receptive fields ($r = 2^{j+1} - 1$), while the same stacking structure for the standard convolution (Fig. 2 (a)) has only linear receptive fields ($r = 2j + 1$). Formally, with dilation l, the filter window from location i is given as

$$\begin{bmatrix} x_i & x_{i+l} & x_{i+2l} & ... & x_{i+(f-1)\cdot l} \end{bmatrix}$$

Clearly, the dilated convolutional structure is more effective to model long-range item sequences, and thus more efficient without using larger filters or becoming deeper. In practice, to further increase the model capacity and receptive fields, one just need to repeat the architecture in Fig. 2 multiple times by stacking, e.g., 1, 2, 4, ..., 16, 1, 2, 4, ..., 16, 1, 2, 4, ..., 16.

3.2.3 One-dimensional Transformation: Although our dilated convolution operator depends on the 2D input matrix E, the proposed network architecture is actually composed of all 1D convolutional layers. To model the 2D embedding input, we perform a simple reshape operation, which serves as a prerequisite for performing 1D convolution. Specifically, the 2D matrix E is reshaped from $t \times 2k$ to a 3D tensor T of size $1 \times t \times 2k$, where $2k$ is treated as the "image" channel rather than the width of the standard convolution filter in *Caser*. Fig. 3 (b) illustrates the reshaping process.

3.3 Masked Convolutional Residual Network

Although increasing the depth of network layers can help obtain higher-level feature representations, it also easily results in the vanishing gradient issue, which makes the learning process much harder. To address the degradation problem, residual learning [11] has been introduced for deep networks. While residual learning has achieved huge success in the domain of computer vision, to our knowledge it has not appeared in the recommender system literature.

The basic idea of residual learning is to stack multiple convolutional layers together as a block and then employ a skip connection

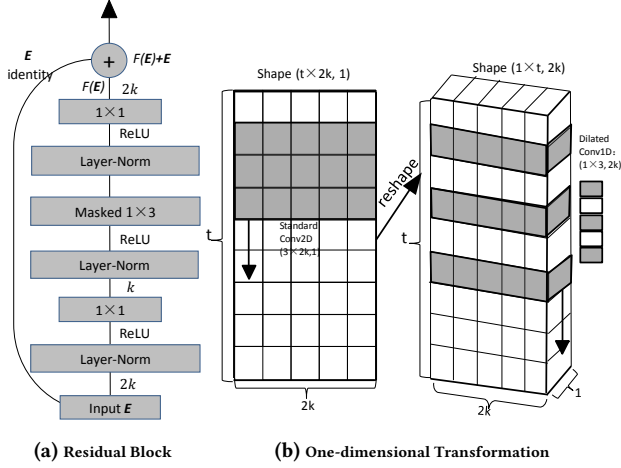**(a)** Residual Block **(b)** One-dimensional Transformation

Figure 3: Dilated residual block (a) and one-dimensional transformation (b) shows the 2D filter ($C = 1$) in *Caser* (left) and the 1D 2-dilated filter ($C = 2k$) in our model (right); the vertical black arrows represent the direction of the sliding convolution. In this work, the default stride for the dilated convolution is 1. Note the reshape operation in (b) is performed before each convolution in (a) (i.e., 1×1 and masked 1×3), which is then followed by a reshape back step after convolution.

scheme that passes the previous layers's feature information to its posterior layer. The skip connection scheme allows to explicitly fit the residual mapping rather than the original identity mapping, which can maintain the input information and thus enlarge the propagated gradients. Formally, denoting the desired mapping as $H(E)$, we let the residual block fit another mapping of $F(E) = H(E) - E$. The desired mapping now is recast into $F(E) + E$ by element-wise addition (assuming that $F(E)$ and E are of the same dimension). As has been evidenced in [11], optimizing the residual mapping $F(E)$ is much easier than the original, unreferenced mapping $H(E)$. In the following, we describe in detail the proposed residual building blocks.

We design a residual block structure and then wrap each convolutional layer by it (see Fig. 3 (a)). Each block is made up of the normalization, activation (e.g., ReLU [19]), convolutional layers and a skip connection in a specific order. First, we adopt the state-of-the-art layer normalization [1] before each activation layer, as it is well suited to sequence processing in contrast with batch normalization [15]. Second, we focus on introducing the property of the convolutional layers. The residual block consists of 3 convolution filters: one dilated filter of size 1×3 and two regular filters of size 1×1. The 1×1 filters are introduced to change the size of C so as to reduce the parameters to be learned by the 1×3 kernel. The first 1×1 filter (close to input E in Fig. 3 (a)) is to change C from $2k$ to k, while the second 1×1 filter does the opposite transformation in order to maintain the spatial dimensions for the next stacking operation. To clearly show the effectiveness of the 1×1 filters in dimensionality reduction, we compute the number of parameters in both cases. For simplicity, we omit the activation and normalization layers. As we can see, the number of parameters for the 1×3 filter is $1 \times 3 \times 2k \times 2k = 12k^2$ without the 1×1 filters. While with our design, the number of parameters to be learned is

$1 \times 1 \times 2k \times k + 1 \times 3 \times k \times k + 1 \times 1 \times k \times 2k = 7k^2$. The residual mapping $F(E, \{W_i\})$ in Fig. 3 is formulated as:

$$F(E, \{W_i\}) = W_3(\sigma(\psi(W_2(\sigma(\psi(W_1(\sigma(\psi(E)))))))))\qquad(4)$$

where σ and ψ denote ReLU and layer-normalization, W_1 and W_3 denote the convolution weights of standard 1×1 convolutions, while W_2 denotes the weight of l-dilated convolution filter with size of 1×3. Note that bias terms are omitted for simplifying notations.

3.3.1 Dropout-mask: To avoid the future information leakage problem, we propose a clever masking-based dropout trick for the 1D dilated convolution to prevent the network from seeing the future items. Specifically, when predicting $p(x_i|x_{0:i-1})$, the convolution filters are not allowed to make use of the information from $x_{i:t}$. Fig. 4 shows several different ways to perform convolution. As shown, our dropout-masking operation can be implemented either by padding the input sequence in (d) or shifting the output sequence by a few timesteps in (e). The padding method in (e) is very likely to result in information loss in a sequence, particularly for short sequences. Hence in this work, we apply the padding strategy in (d) with the padding size of $(f - 1) * l$.

3.4 Final Layer, Network Training and Generating

As mentioned before, the matrix in the last layer of the convolution architecture (see Fig. 2), denoted by E^o, preserves the same dimensional size of the input E, i.e., $E^o \in \mathbb{R}^{t \times 2k}$. However, the output should be a matrix or tensor that contains probability distributions of all items in the output sequence $x_{1:t}$, where the probability distribution of x_t is the desired one that generates top-N predictions. To do this, we propose using one more convolutional layer on top of the last convolutional layer in Fig. 2 with filter of size $1 \times 1 \times 2k \times n$, where n is the number of items. Following the procedure of one-dimensional transformation in Fig. 3 (b), we obtain the expected ouput matrix $E^p \in \mathbb{R}^{t \times n}$, where each row vector after the softmax operation represents the categorical distribution over x_i ($0 < i \leq t$).

The aim of optimization is to maximize the log-likelihood of the training data w.r.t. θ. Clearly, maximizing $\log p(\mathbf{x})$ is mathematically equivalent to minimizing the *sum* of the binary cross-entropy loss for each item in $x_{1:t}$. To save the space, we simply omit the specific formulation.

For comparison purpose, we only predict the next one item in our evaluation, and then stop the generating process. Nevertheless, the model is able to generate a sequence of items simply by feeding the predicted one item (or sequence) into the network to predict the next one, and thus the prediction at the generating phrase is *sequential*. This matches most real-world recommendation scenarios, where the next action is followed when the current one has been observed. But at both training and evaluation phases, the conditional predictions for all timesteps can be made *in parallel*, because the complete sequence of input items $\mathbf{x}$ is already available.

4 EXPERIMENTS

In this section we detail our experiments, report results for several data sets, and compare our model (called *NextItNet*) with the well-known RNN-based model *GRURec*[3] [14, 26] and the state-of-the-art
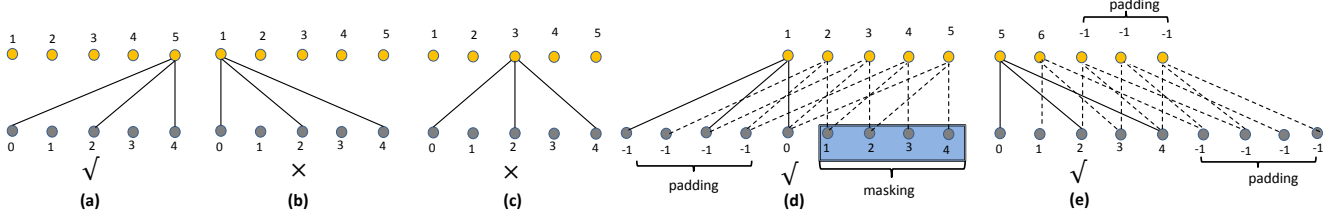

Figure 4: The future item can only be determined by the past ones according to Eq. (1). (a) (d) and (e) show the correct convolution process, while (b) and (c) are wrong. E.g., in (d), items of $\{1, 2, 3, 4\}$ are masked when predicting 1, which can be technically implemented by padding.

Table 1: Session statistics of all data sets.

DATA	YOO	MUSIC_M5	MUSIC_L5	MUSIC_L10	MUSIC_L20	MUSIC_L50	MUSIC_L100
RAW-SESSIONS	0.14M	0.61M	2.14M	1.07M	0.53M	0.21M	0.11M
SUB-SESSIONS-T	0.07M	0.31M	1.07M	3.21M	4.28M	4.91M	5.10M

MUSIC_M5 denotes *MUSIC_M* with maximum session size of 5. The same applies to *MUSIC_L*. 'M' denotes 1 million.

CNN-based model *Caser*[4] [27]. Note that (1) since the main contributions in this paper do not focus on combining various features, we omit the comparison with content- or context-based sequential recommendation models, such as the 3D CNN recommender [28] and other RNN variants [10, 18, 22, 25]; (2) the *GRURec* baseline could be regarded as the state-of-the-art *Improved GRURec* [26] when dealing with the long-range session data sets because our main data augmentation technique for the two baseline models follows the same way in *Improved GRURec*.

4.1 Datasets and Experiment Setup

4.1.1 Datasets and Preprocessing. The first data set 'Yoochoose-buys' (YOO for short) is chosen from the RecSys Challenge 2015[5], which contains buying and clicking datasets. We choose the buying dataset for our evaluation. To avoid noise data, we filter out sessions of length shorter than 3. To fairly compare the capacity of *NextItNet* in modeling sequences, we do not consider additional contexts in this paper, although Yoo contains item price and quality information. We find that in the processed Yoo data 96% sessions have a length shorter than 10, and we simply remove the 4% longer sessions and refer it as a short-range sequential data.

The remaining data sets are extracted from Last.fm[6]: one medium-size (*MUSIC_M*) and one large-scale (*MUSIC_L*) collection by randomly drawing 20,000 and 200,000 songs respectively. In the Last.fm data set, we observe that most users listen to music several hundred times a week, and some even listen to more than one hundred songs within a day. Hence, we are able to test our model in both short- and long-range sequences by cutting up these long-range listening sessions. In *MUSIC_L*, we define the maximum session length t as 5, 10, 20, 50 and 100, and then extract every t successive items as our input sequences. This is done by sliding a window of both size and stride of t over the whole data. We ignore sessions in which the time span between the last two items is longer than 2 hours. In this way, we create 5 data sets, referred to as *RAW-SESSIONS*.

We randomly split these *RAW-SESSIONS* data into training (50%), validation (5%), and testing (45%) sets.

As mentioned before, the performance of *Caser* and *GRURec* is supposed to degrade significantly for long sequence inputs, such as when $t = 20$, 50 and 100. For example, when setting $t = 50$, *Caser* and *GRURec* will predict x_{49} by using $x_{0:48}$, but without explicitly modeling the item inter-dependencies between x_0 and x_{48}. To remedy this defect, when $t > 5$, we follow the common approach [18, 26] by manually creating additional sessions from the *training* sets of *RAW-SESSIONS* so that *Caser* and *GRURec* can leverage the full dependency to a large extent Similar observation has also been made in *YOO*. Still setting $t = 50$, one training session will then produce 45 more sub-sessions by padding the beginning and removing the end indices, referred to as *SUB-SESSIONS*. The example of the 45 sub-sessions are given as follows: $\{x_{-1}, x_0, x_1, ..., x_{48}\}$, $\{x_{-1}, x_{-1}, x_0, ..., x_{47}\}$,..., $\{x_{-1}, x_{-1}, x_{-1}, ..., x_4\}$. We show the statistics of *RAW-SESSIONS* & training data of *SUB-SESSIONS* (i.e., *SUB-SESSIONS-T*) in Table 1.

All models were trained on GPUs (TITAN V) using Tensorflow. From the different gradient descent optimizers tried, Adam [17] gave the best convergence performance and was used for all evaluations. The learning rates and batch sizes of baseline methods were manually set according to performance in validation sets. For all data sets, *NextItNet* used the learning rate of 0.001 and batch size of 32. Embedding size $2k$ is set to 64 for all models without special mention.

4.1.2 Evaluation Protocols. We reported the evaluated results by three popular top-N metrics, namely MRR@N (Mean Reciprocal Rank) [14] and HR@N (Hit Ratio) [12] and NDCG@N [12] (Normalized Discounted Cumulative Gain). N is set to 5 and 20 for comparison. We evaluate the prediction accuracy of the *last* (i.e., next) item of each sequence in the testing set, similarly to [13, 27].

4.2 Accuracy Summary

Overall performance results of all methods are illustrated in Table 2 and 3, which clearly show that the neural network models (i.e., *Caser*, *GRURec* and our model) obtain very promising accuracy in

---

[4]https://github.com/graytowne/caser
[5]http://2015.recsyschallenge.com/challenge.html
[6]http://www.dtic.upf.edu/ ocelma/MusicRecommendationDataset/lastfm-1K.html

Table 2: Accuracy comparison. The upper, middle and below tables are MRR@5, HR@5 and NDCG@5 respectively.

DATA	YOO	MUSIC_M5	MUSIC_L5	MUSIC_L10	MUSIC_L20	MUSIC_L50	MUSIC_L100
MostPop	0.0050	0.0024	0.0006	0.0007	0.0008	0.0007	0.0007
GRURec	0.1645	0.3019	0.2184	0.2124	0.2327	0.2067	0.2086
Caser	0.1523	0.2920	0.2207	0.2214	0.1947	0.2060	0.2080
NextItNet	**0.1715**	**0.3133**	**0.2327**	**0.2596**	**0.2748**	**0.2735**	**0.2583**
MostPop	0.0151	0.0054	0.0014	0.0016	0.0016	0.0016	0.0016
GRURec	0.2773	0.3610	0.2626	0.2660	0.2694	0.2589	0.2593
Caser	0.2389	0.3368	0.2443	0.2631	0.2433	0.2572	0.2588
NextItNet	**0.2871**	**0.3754**	**0.2695**	**0.3014**	**0.3166**	**0.3218**	**0.3067**
MostPop	0.0075	0.0031	0.0008	0.0009	0.0010	0.0009	0.0009
GRURec	0.1923	0.3166	0.2294	0.2258	0.2419	0.2197	0.2212
Caser	0.1738	0.3032	0.2267	0.2318	0.2068	0.2188	0.2207
NextItNet	**0.2001**	**0.3288**	**0.2419**	**0.2700**	**0.2853**	**0.2855**	**0.2704**

MostPop returns the most popular item respectively. Regarding the setup of our model, we use two-hidden-layer convolution structure with dilation factor 1, 2, 4 for the first four data sets (i.e., *YOO*, *MUSIC_M5* , *MUSIC_L5* and *MUSIC_L10*), while for the last three long-range sequence data sets, we use 1, 2, 4, 8, 1, 2, 4, 8, to obtain above results.

Table 3: Accuracy comparison. The upper, middle and below tables are MRR@20, HR@20 and NDCG@20 respectively.

DATA	YOO	MUSIC_M5	MUSIC_L5	MUSIC_L10	MUSIC_L20	MUSIC_L50	MUSIC_L100
MostPop	0.0090	0.0036	0.0009	0.0010	0.0011	0.0011	0.0011
GRURec	0.1839	0.3103	0.2242	0.2203	0.2374	0.2151	0.2162
Caser	0.1660	0.2979	0.2234	0.2268	0.2017	0.2133	0.2153
NextItNet	**0.1901**	**0.3223**	**0.2375**	**0.2669**	**0.2815**	**0.2794**	**0.2650**
MostPop	0.0590	0.0180	0.0052	0.0053	0.0056	0.0056	0.0056
GRURec	0.4603	0.4435	0.3197	0.3434	0.3158	0.3406	0.3336
Caser	0.3714	0.3937	0.2703	0.3150	0.3110	0.3273	0.3298
NextItNet	**0.4645**	**0.4626**	**0.3159**	**0.3709**	**0.3814**	**0.3789**	**0.3731**
MostPop	0.0195	0.0066	0.0018	0.0019	0.0021	0.0020	0.0020
GRURec	0.2460	0.3405	0.2460	0.2481	0.2553	0.2433	0.2427
Caser	0.2122	0.3197	0.2342	0.2469	0.2265	0.2392	0.2412
NextItNet	**0.2519**	**0.3542**	**0.2554**	**0.2904**	**0.3041**	**0.3021**	**0.2895**

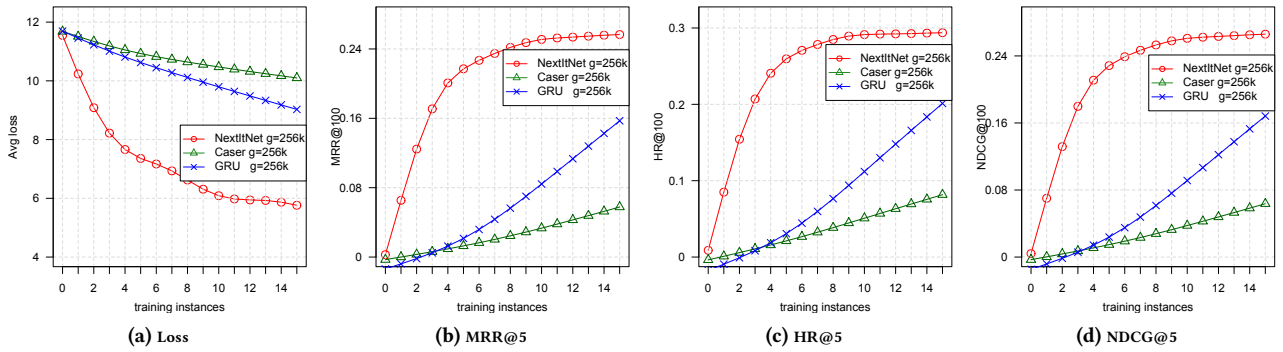(a) Loss   (b) MRR@5   (c) HR@5   (d) NDCG@5

Figure 5: Convergence behaviors of *MUSIC_L20*.

the top-N sequential recommendation task. For example, in *MU-SIC_M5*, the three neural models perform more than 120 times better on MRR@5 than *MostPop*, which is a widely used recommendation benchmark. The best MRR@20 result we have achieved by *NextIt-Net* is 0.3223 in this data set, which roughly means that the desired

item is ranked on position 3 in average among the 20,000 candidate items. We then find that among these neural network based models, *NextItNet* largely outperforms *Caser & GRURec*. We believe there are several reasons contributing to the state-of-the-art results. First, as highlighted in Section 3.1, *NextItNet* takes full advantage
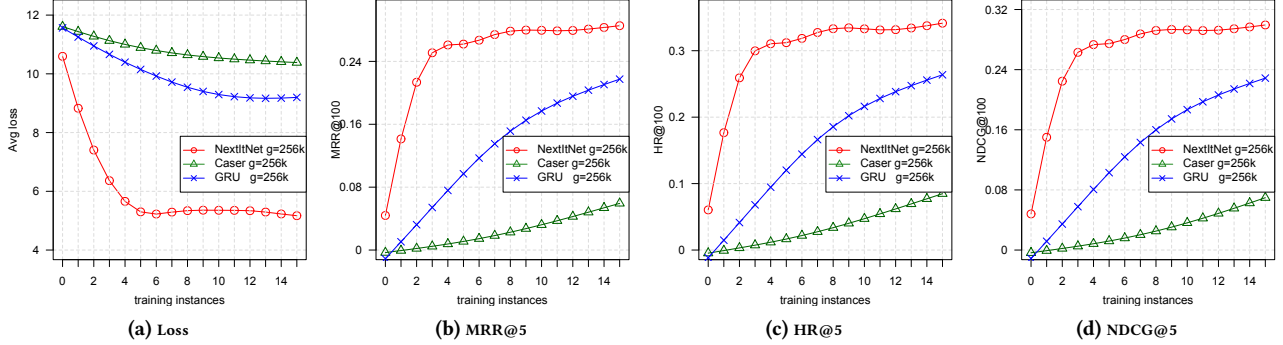
(a) Loss   (b) MRR@5   (c) HR@5   (d) NDCG@5

Figure 6: Convergence behaviors of *MUSIC_L100*. GRU is short for *GRURec*. g = 256k means the number of training sequences (or sessions) of one unit in x-axis is 256k. Note that (1) to speed up the experiments, all of the convergence tests are evaluated on the first 1024 sessions in the testing set; (2) only *NextItNet* has converged in above figures, while *GRU* and *Caser* require more training instances for convergence.

Table 4: Effects of sub-session in terms of MRR@5. The upper, middle and below tables represent GRU, Caser and NextItNet respectively. "10","20","50" and "100" the session length. All high parameters are fixed.

Sub-session	10	20	50	100
Without	0.1985	0.1645	0.1185	0.0746
With	0.2124	0.2327	0.2067	0.2086
Without	0.1571	0.1012	0.0216	0.0084
With	0.2214	0.1947	0.2060	0.2080
Without	0.2596	0.2748	0.2735	0.2583

Table 5: Overall training time (mins).

Model	GRURec	Caser	NextItNet
MUSIC_L5	66	59	54
MUSIC_L20	282	191	76
MUSIC_L100	586	288	150

of the complete sequential information instead of just the sequence information between the last item and the previous sequence. This can be easily verified in Table 4, where we observe that *Caser & GRURec* without subsession perform extremely worse with long sessions. Particularly, *Caser* performs much worse than *GRURec*. The reason is because compared with a regular CNN model[7], RNN has an inherent advantage in modeling long-range sequences. More importantly, the poor performance of *Caser* can be attributed to the max pooling operation, which seriously hurts its performance when modeling long-range sequences. In addition, even with sub-session *Caser & GRURec* still show significantly worse results than *NextItNet* because the separate optimization of each sub-session is clearly suboptimal compared with leveraging full sessions by *NextItNet*. Second, unlike *Caser*, *NextItNet* has no pooling layers, although it is also a CNN based model. As a result, *NextItNet* preserves the whole spatial resolution of the original embedding matrix E without any information lost. The third advantage may be that *NextItNet* can support deeper layers by using residual learning, which better suits for modeling complicated relations and long-range dependencies.

4.2.1 Efficiency Comparison: In addition to the advantage of recommendation accuracy, we have also evaluated the efficiency of *NextItNet* in Table 5. First, it can be seen that *NextItNet* and *Caser* requires less training time than *GRURec* in all three data sets. The reason that CNN-based models can be trained much faster is due to

the full parallel mechanism of convolutions. Clearly, the training speed advantage of CNN models are more preferred by modern parallel computing systems. Second, it shows that *NextItNet* achieves further improvements in training time compared with *Caser*. The faster training speed is mainly because *NextItNet* leverages the complete sequential information during training and then converges much faster by less training epochs. To better understand of the convergence behaviours, we have shown them in Fig. 5 and 6. As can be seen, our model with the same number of training sessions is converging faster (a) and better (b, c, d) than *Caser* and *GRURec*. This confirms our claim in Section 3.1 since *Caser* and *GRURec* cannot make full use of the internal sequential information in the session. The faster and better convergence behaviors indicate the effectiveness of the proposed generative architecture.

5 CONCLUSION

In this paper, we presented a simple, efficient and highly effective convolutional generative model for session-based top-N item recommendations. The proposed model combines masked filters with 1D dilated convolutions to increase the receptive fields, which is very important to model the long-range dependencies. In addition, we have applied residual learning to enable training of much deeper networks. We have shown that our model can greatly outperform state-of-the-arts in real-world session-based recommendation tasks. The proposed model can serve as a generic method for modeling both short- and long-range session-based recommendation data.

For comparison purposes, we have not considered additional contexts in either our model or baselines. However, our model is flexible to incorporate various context information. For example, if we know the user identity u and location p, the distribution in

---

[7]In fact, *Caser* is not a real sequential model because the max pooling operation does not retain sequential relations of items.

Eq. (1) can be modified as follows to incorporate these information.

$$p(\mathbf{x}) = \prod_{i=1}^{t} p(x_i|x_{0:i-1}, \mathbf{u}, P, \boldsymbol{\theta})p(x_0) \qquad (5)$$

where we can combine E (before convolution) or E^o (after convolution) with the user embedding vector $\mathbf{u}$ and location matrix P by element-wise operations, such as multiplication, addition or concatenation. We leave the evaluation for future work.

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[2] Sotirios P. Chatzis, Panayiotis Christodoulou, and Andreas S. Andreou. 2017. Recurrent Latent Variable Networks for Session-Based Recommendation. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems (DLRS 2017)*. ACM, 38–45.

[3] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. 2016. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915* (2016).

[4] Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. 2012. SVDFeature: a toolkit for feature-based collaborative filtering. *Journal of Machine Learning Research* 13, Dec (2012), 3619–3622.

[5] Chen Cheng, Haiqin Yang, Michael R Lyu, and Irwin King. 2013. Where You Like to Go Next: Successive Point-of-Interest Recommendation.. In *IJCAI*.

[6] Zhiyong Cheng, Jialie Shen, Lei Zhu, Mohan Kankanhalli, and Liqiang Nie. 2017. Exploiting Music Play Sequence for Music Recommendation. In *IJCAI*.

[7] Qiang Cui, Shu Wu, Yan Huang, and Liang Wang. 2017. A Hierarchical Contextual Attention-based GRU Network for Sequential Recommendation. *arXiv preprint arXiv:1711.05114* (2017).

[8] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional Sequence to Sequence Learning. *arXiv preprint arXiv:1705.03122* (2017).

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[10] Youyang Gu, Tao Lei, Regina Barzilay, and Tommi S Jaakkola. 2016. Learning to refine text based recommendations.. In *EMNLP*. 2103–2108.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[12] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 549–558.

[13] Balázs Hidasi and Alexandros Karatzoglou. 2017. Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. *arXiv preprint arXiv:1706.03847* (2017).

[14] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[15] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*. 448–456.

[16] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099* (2016).

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

[18] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural Attentive Session-based Recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 1419–1428.

[19] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 807–814.

[20] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

[21] Wenjie Pei, Jie Yang, Zhu Sun, Jie Zhang, Alessandro Bozzon, and David M.J. Tax. 2017. Interacting Attention-gated Recurrent Networks for Recommendation. *arXiv preprint arXiv:1711.05114* (2017).

[22] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks. *arXiv preprint arXiv:1706.04148* (2017).

[23] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM, 811–820.

[24] Tom Sercu and Vaibhava Goel. 2016. Dense Prediction on Sequences with Time-Dilated Convolutions for Speech Recognition. *arXiv preprint arXiv:1611.09288* (2016).

[25] Elena Smirnova and Flavian Vasile. 2017. Contextual Sequence Modeling for Recommendation with Recurrent Neural Networks. *arXiv preprint arXiv:1706.07684* (2017).

[26] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 17–22.

[27] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *ACM International Conference on Web Search and Data Mining*.

[28] Trinh Xuan Tuan and Tu Minh Phuong. 2017. 3D Convolutional Networks for Session-based Recommendation with Content Features. In *Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys '17)*. ACM.

[29] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2015. Learning hierarchical representation model for nextbasket recommendation. In *Proceedings of the 38th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 403–412.

[30] Fisher Yu and Vladlen Koltun. 2015. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122* (2015).

[31] Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. 2016. Lambdafm: learning optimal ranking with factorization machines using lambda surrogates. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 227–236.

[32] Fajie Yuan, Xin Xin, Xiangnan He, Guibing Guo, Weinan Zhang, Tat-Seng Chua, and Joemon Jose. 2018. fBGD: Learning Embeddings From Positive Unlabeled Data with BGD. *Association for Uncertainty in Artificial Intelligence*.