UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JÚLIO BARRETO GUEDES DA COSTA

NATURAL LANGUAGE PROCESSING TECHNIQUES FOR SESSION-BASED RECOMMENDATION

CAMPINA GRANDE - PB

2020

JÚLIO BARRETO GUEDES DA COSTA

NATURAL LANGUAGE PROCESSING TECHNIQUES FOR SESSION-BASED RECOMMENDATION

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Leandro Balby Marinho.

CAMPINA GRANDE - PB

2020

C837n Costa, Júlio Barreto Guedes da.
Natural language processing techniques for session-based recommendation. / Júlio Barreto Guedes da Costa. – 2020.

13 f.

Orientador: Prof. Dr. Leandro Balby Marinho.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Processamento de linguagem natural. 2. Sistemas de recomendação. 3. Recomendação baseada em sessões. 4. Performance de sistema de recomendação. 5. Matrizes. 6. Rede neural recorrente. 7. Gated recurrent unit. I. Marinho, Leandro Balby. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

JÚLIO BARRETO GUEDES DA COSTA

NATURAL LANGUAGE PROCESSING TECHNIQUES FOR SESSION-BASED RECOMMENDATION

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Trabalho aprovado em: 2020.

CAMPINA GRANDE - PB

Natural Language Processing Techniques for Session-Based Recommendation

Júlio Barreto Guedes da Costa
julio.costa@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brazil

Leandro Balby Marinho
lbmarinho@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brazil

ABSTRACT

Recommender Systems is a field of research and application focused on identifying and retrieving relevant items given user preferences. There are many scenarios where a Recommender System can be applied, but its performance usually depends on the availability of user consumption historic data. In this work, we evaluate the performance of Recommender Systems in the Session-based scenario, in which the user cannot be identified, comparing the performance of naïve, matrix-based, sequential, and session-based models, also introducing an alternative implementation of one of these models, based on a specific type of Recurrent Neural Network called Gated Recurrent Units. We use Natural Language Processing techniques to create three different input strategies and create session embeddings, analyzing their performance in our model implementation, extracting insights, and applying fine tuning to achieve better results. This work was evaluated using a real-world database extracted from the Last.fm online radio platform.

RESUMO

Sistemas de Recomendação é um campo de pesquisa e aplicação que objetiva identificar e recuperar itens relevantes, dadas as preferências do usuário. Existem muitos cenários em que Sistemas de Recomendação podem ser aplicados, mas sua performance usualmente depende da disponibilidade de dados relacionados ao histórico de consumo do usuário. Neste trabalho, nós avaliamos a performance de Sistemas de Recomendação no cenário baseado em sessões, em que o usuário não pode ser identificado, comparando a performance de métodos ingênuos, baseados em matrizes, sequenciais, e baseados em sessões, além de introduzir uma implementação alternativa de um destes, cuja implementação faz uso de um tipo específico de Rede Neural Recorrente chamado *Gated Recurrent Unit*. Nós usamos técnicas de Processamento de Linguagem Natural para criar três diferentes estratégias de entrada para os dados e gerar *Embeddings* das sessões, analisando a performance da nossa implementação, percebendo possíveis melhorias, e aplicando ajustes finos para obter melhores resultados. Este trabalho foi avaliado usando uma base de dados real extraída da plataforma Last.fm.

KEYWORDS

Recommender Systems, Natural Language Processing, Session-based Recommendation

PALAVRAS-CHAVE

Sistemas de Recomendação, Processamento de Linguagem Natural, Recomendação baseada em sessões

1 INTRODUCTION

With the evolution of internet, Recommendation Systems (RSs) have become an essential part of what is accessed everyday. Emerging in the mid-1990s, RSs are software tools and techniques that provide customized suggestions for each user, indicating items (e.g. a movie, a product), to support the process of decision-making. Social networks, blogs, e-commerce, streaming, and many other websites and services are applying RSs to improve functionalities, better retain users, or sell more products. The creation of new RSs methods involves expertise in different fields, such as Artificial Intelligence, Machine Learning, Data Mining, Statistics, Marketing, and others [20].

Traditional RSs rely on historic user data to generate personalized recommendations about products or items that such user is likely interested in. Collaborative Filtering (CF), a widely used approach, considers interactions of other users to make recommendations. These interactions can either be explicit, such as ratings given by users to items, or implicit, such as how many times an user listened to a song, to make recommendations. There are two different approaches for CF:

(1) **Memory-Based Methods**. Also referred to as neighborhood-based methods, the value of an unseen user-item interaction is defined according to its neighborhood under certain criteria. These methods can be user-based or item-based, according to the neighborhood considered;
(2) **Model-Based Methods**. In these methods, Machine Learning (ML) and Data Mining are used for prediction. Examples include the use of models such as decision trees, regression models, support vector machines and neural networks (NNs) [1].

The performance of ML algorithms often depends on the given data representation, such as predictive features. Deep Learning (DL), a sub-field of ML, came up with techniques that address such data representation problem by introducing complex representations that are expressed in terms of other, simpler ones [9]. DL has been applied over multiple tasks (e.g. Natural Language Processing, Image Classification) and achieved remarkable results [7]. According to a recent survey, the increase of available data, among other factors, encouraged the use of DL in recommendation tasks, and recently proposed models achieved or overcame state-of-the-art performance [24].

When addressing DL to Natural Language Processing (NLP), Recurrent Neural Networks (RNNs) have been used due to the sequential nature of the data (i.e. text or audio) in tasks such as speech recognition, text generation and word spotting [7]. Word Embeddings, a more recent data representation strategy that captures the contextual meaning of the word, was used in recently proposed algorithms that surpassed the previous state-of-the-art methods [14, 15, 17].

Recalling the recommendation scenario, an important condition for traditional CF algorithms to work is the existence of informative user profiles. However, there are many application scenarios, such as e-commerce websites or streaming services, where this condition is not satisfied, either because users are not required to authenticate and/or they are newcomers. This scenario is often called Sequence-Aware Recommendation, and its goal is to recommend objects that match a given sequence of user actions [18].

It is important to emphasize that the intent of a user can change after certain period of inactivity: an e-commerce website user can search for a book and, some time later, use the same website searching for a cellphone. In such scenario, a RS that analyzes the most recent interactions apart from the previous ones is preferred, creating an sub-field called Session-Based Recommendation [12].

Due to the sequential nature of the data, and the notion co-occurrence of items, RNNs have also been applied in the Sequential Recommendation and Session-based scenarios, and robust NLP models that using RNNs have been adapted and proven to be suited for these tasks [8, 21]. However, they often use different frameworks, architectures, and datasets, making it difficult to create a direct comparison. The goal of our work is to evaluate the performance of Sequence-Aware and Session-based models in a common ground, and how the data representation might change the performance of these models. Additionally, we implemented an alternative version of a Session-based model using the recently released Tensorflow Recommenders.

2 THEORETICAL BACKGROUND

This section introduces particular concepts of NLP and NNs, essential to understanding the principal aspects of this study.

2.1 Data Representation

The RSs are applied over three different entities: (1) the users that interact with the system, denoted by $U = \{u_1, u_2, ..., u_n\}$; (2) the items available in the system, denoted by $I = \{i_1, i_2, ..., i_m\}$; and (3) the rating given by an user to an item, i.e., $r_{p,q}, \forall p \in U, q \in I$.

In most RSs scenarios, the goal is the matrix-completion problem: the data is represented using a $N \times M$ matrix such that each cell $r_{p,q}$ represents the interaction between the user c_p and the item i_q. An example of such representation is given in Eq. (1).

$$\begin{array}{c|cccc} & i_1 & i_2 & \dots & i_m \\ \hline u_1 & 1 & 3 & \dots & 4 \\ u_2 & - & 4 & \dots & 5 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_n & 2 & 3 & \dots & 3 \end{array} \tag{1}$$

In contrast to matrix-completion problems, Sequential RSs receive a timestamp ordered list $H = [h_1, h_2, \dots, h_{|H|}]$ where each element represents an action over an item, but may have additional contextual information, such as user-related information. The goal is not predicting the value of each missing $r_{p,q}$, but in computing an ordered list of objects L of length k for each user, where each element $l \in L$ corresponds to an element of $i \in I$ [19], as shown in Eq. (2).

$$[h_1, h_2, \dots, h_{|H|}] \rightarrow RS \rightarrow [l_1, l_2, \dots, l_k] \tag{2}$$

The goal is to compare the predicted next items L with the true list of next items T. The length k of these lists is often denoted along the name of a metric, such as *Metric@k*.

2.2 Natural Language Processing (NLP)

NLP is an area of research and application that aims to gather knowledge on how human beings understand and use language so that appropriate tools and techniques can be developed to make computer systems understand and manipulate the language to perform tasks such as information retrieval, speech recognition, text generation, and others. The foundations of NLP lie in a number of disciplines that range from mathematics and engineering to linguistics and psychology [4].

One recurrent problem in NLP is how to properly and efficiently represent the text. A very popular technique is to extract n-grams of the sentence, where $n \in \mathbb{N}^*$. An example of this technique can be seen in Table 1.

Table 1: N-Grams transformation.

Sentence	1-Gram	2-Gram
The cat is furry	The, cat, is, furry	The cat, cat is, is furry
I am going	I, am, going	I am, am going

While many NLP applications use n-grams, increasing n also increases the number of tokens found, and while this can lead to better performance, it also raises the cost. Another common representation is the 1-of-V vector, also called 1-of-N and One-Hot encoding, where each word is mapped into a V-sized vector in which V is the number of unique tokens (i.e. the number of unique words), and only one of the elements is 1, representing the word in that position. Considering an example with the sentence "I went to London" being the corpus, the 1-of-V representation would be a sequence of vectors such as Eq. (3).

$$\begin{array}{r} I \\ London \\ to \\ went \end{array} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{3}$$

However, while being simple, there still are problems in these representations: when applied over a large corpus, each vector has high dimensionality, thus requiring more memory; these vectors do not extract information such as the similarity between words nor the context in which they were used, and others [15].

2.2.1 *Word Embeddings.*

Word embeddings is a set of techniques and methods that map each word of the corpus into a real valued vector of d dimensions. The recently proposed model families for learning word vectors are:

(1) **Local Context Window Methods**. These methods usually map each word to a vector via a learned lookup-table. Then a sequence of w word vectors, also called context window,

is used as input to a shallow NN applied to a task. When the NN parameters are adjusted in response to a particular word or word-sequence, the improvements will carry over to occurrences of similar words and sequences, updating those vectors. Examples of these methods are the Continuous Bag-of-Words (CBOW) and the Skip-Gram models [6, 14, 15];

(2) **Global Vectors (GloVe)**. This method considers not only the context window, but the global co-occurrence counts, making a better use of global corpus statistics. However, instead of using preliminary values as input to a NN, this method has predefined functions to calculate the word vector [17].

2.3 Neural Networks

Neural networks (NNs), or Artificial Neural Networks (ANNs), are one of the basis of DL. The goal of a NN is to find some function $f*$ that approximates an existing function f based on given examples [9]. A NN can be composed of several layers, each layer containing a number of neurons. Each neuron receives a vector x of inputs, a vector w of weights, and a scalar bias b, and applies an activation function θ to generate an output y, as shown in Figure 1.

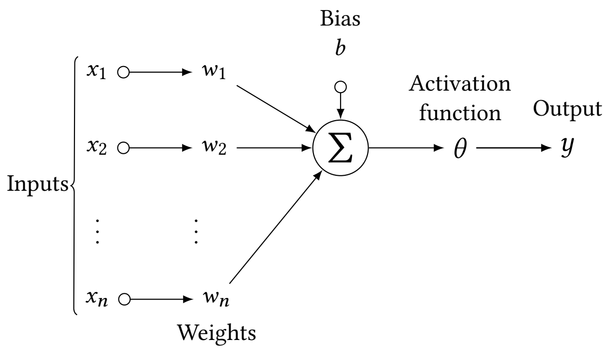


Figure 1: A NN unit, also called a neuron.

Mathematically, each neuron can be seen as in Eq. (4):

$$y = \theta\left(\sum_{i=0}^{N}(w_i * x_i) + b\right) \tag{4}$$

When the NN has multiple layers, the outputs of the neurons in the first layer becomes the inputs of the neurons in the second layer and so on. An example of this can be seen in Figure 2. These neural networks are often called **Deep feedforward networks**, or **Multilayer Perceptrons** (MLPs) [9]. If these layers are part of a larger NN, they are called **Fully-Connected layers** or **Dense layers** [12]. When approximating $f*$ the NN adjusts its weights to produce a closer result, in a process referred to as **learning**, and when multiple layers are stacked the NN is called **Deep NN** (DNN), thus the term **Deep Learning**.

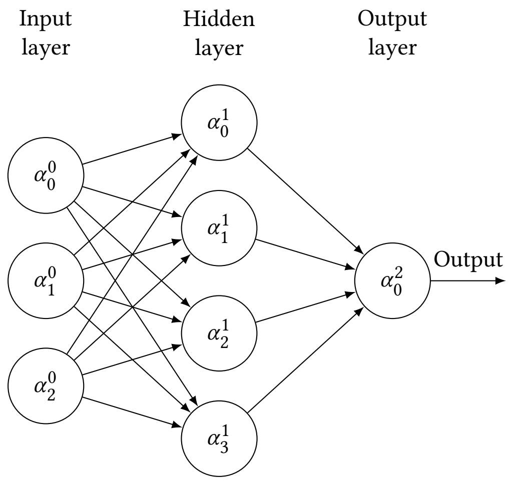


Figure 2: An example of MLP.

2.3.1 *Recurrent Neural Networks.*

However, the inputs of a MLP have no relation between themselves, and there is no feedback between them, making it difficult to solve problems with sequential input, such as text or sound. When MLPs are extended to include feedback connections, they are called **Recurrent Neural Networks** (RNNs) [9]: the input at time t can have influence over the input at $t + 1$ and so on. An example of a single layer RNN can be seen in Figure 3.

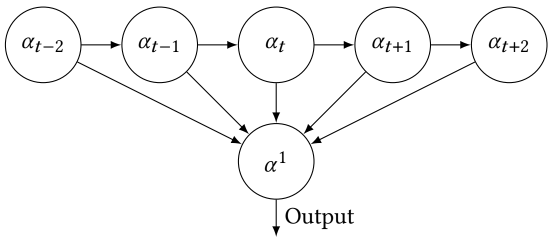


Figure 3: An example of a single layer RNN.

Since RNNs were created, more advanced layers that include feedback were proposed, such as the Long Short-Term Memory (LSTM) [13] and the Gated Recurrent Unit (GRU) [3], taking advantage over the classical RNN unit by using memory and forget gates, advancing even more the field [7].

3 RELATED WORK

Many of the methods and techniques previously mentioned have been applied to Sequential Recommendation tasks, achieving or overcoming state-of-the-art performance. In this section we will discuss about these works, the models proposed, the datasets used, and how they were evaluated.

3.1 Naïve Models

A model can be considered naive when it does not consider the input before accomplishing its task or performs an obvious and predefined inference based on the input. In the recommendation field, a model is naive when it always recommends the same item or set of items regardless the user-item matrix or the history of actions, and their use as baselines can be considered a sanity check for the proposed models. Two of these methods are frequently used as baselines:

- **Top Popular**. Also referred as Pop, this frequency-based method always returns the L most frequent items in the entire training collection.
- **SPop**. Another frequency-based method, SPop is an adaptation of Top Popular for Session-based recommendation: It always returns the most frequent items of the current session.

3.2 ItemKNN

Based on the k-Nearest Neighbors algorithms, ItemKNN is a Memory-Based collaborative filtering model, that receives an user-item ratings matrix, shown by Eq. (1), and its goal is to complete the missing $r_{p,q}$ values, introduced in Section 2.1 as the matrix-completion problem.

Since the Sequential Recommendation problem does not have explicit feedback (i.e. a rating given by an user to an item), the user-item matrix is a binary matrix indicating whether the user interacted or not with that item, thus losing the temporal aspect of the data. Using this representation, the ItemKNN algorithm finds the k most similar items by using a similarity measure, such as the cosine distance or adjusted cosine distance, and uses the ratings of these k items to predict the current item missing ratings [1].

3.3 MLPs and GRUs

The effectiveness of MLPs and GRU-based RNNs in Sequential Recommendation was also put to the test using Word2Vec and GloVe embeddings as input: their experiments used a single layer with 200 neurons, either a fully connected layer in MLP or a GRU in RNN [10].

The models were evaluated using an e-commerce website dataset containing 946 thousand sessions with 15 thousand unique items for training, and 54 thousand sessions for testing; the number of events was not specified. The performance was reported using Precision, Recall, and MRR@10, and compared to a single baseline: a public implementation of Singular Value Decomposition (SVD).

3.4 GRU4Rec

Introduced by Hidasi *et al.* [12], GRU4Rec is a Sequential Recommender that advances the idea of using RNNs in Sequential Recommendation. This GRU-based model creates an embedding from a 1-of-V representation of an item to predict the next item. They evaluated stacking multiple GRU layers, but their experiments showed that a single GRU layer with 1000 units produces better results. The architecture of the model can be seen in Figure 4.

In their work, they evaluate the models using two different datasets:

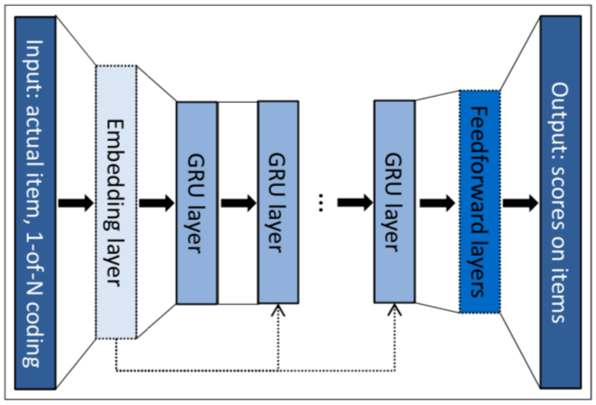


Figure 4: GRU4Rec [12].

(1) **The RecSys '15 challenge dataset**. Contains almost 8 million sessions of 31.6 M events and 37.4 thousand unique items for training, and 15.3 thousand sessions of 71.2 thousand events for testing.
(2) **Youtube-like platform dataset**. Consists in almost 3 M sessions of 13 M events with 330 thousand unique items for training and 37 thousand sessions of 180 thousand events for testing.

The performance of the model was evaluated in comparison against a few baselines: Pop, SPop, ItemKNN and BPR-MF. The comparison was made by calculating the Recall and MRR@20.

3.5 Caser

ConvolutionAI Sequence Embedding Recommendation, simplified as *Caser*, is a model that maps the input sequence H into a matrix $M_{|H|\times d}$, where each interaction h is mapped into a row and the columns are the d dimensions of that interaction's embedding. This matrix can be thought as an image, making it possible to use Convolutional Neural Networks (CNNs). Another proposed advance is to look not only to the next item, but also adopt a skip technique and predict $L*$, where $L*$ is $L_{1:k+1}$. With this techniques, the CNNs can learn both users' general preferences and sequential patterns.

Their model applies horizontal and vertical convolutions in the history input sequence H to create an embedding, which is then combined with the user embedding and used as input to fully connected layers. The architecture of the model can be seen in Figure 5.

The authors evaluate the models using four different datasets: Movielens-1m, Gowalla, Foursquare and Tmall. Since *Caser* is not a Session-based recommender, the number of sessions and events is not reported. The recommendations are evaluated using Precision, Recall and Mean Average Precision (MAP), and their results achieve state-of-the-art performance.

4 METHODOLOGY

The goal of this work is to make a comparison between different algorithms for sequential recommendation of music tracks. To do so, we used one dataset with real-world examples and considered

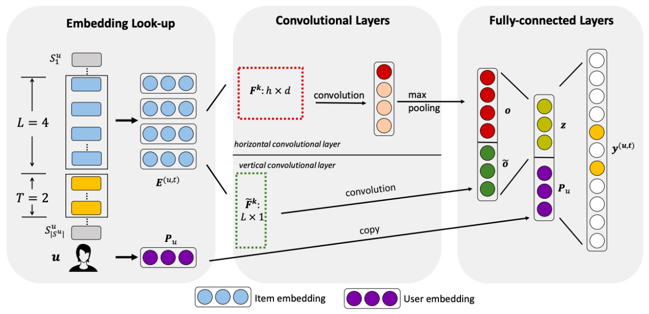


Figure 5: Caser model [22].

In the figure, the input H is represented as L and contains the 4 most recent items. The embedding of these items is generated using a lookup table and they are formatted into a matrix, in which the horizontal and vertical convolutions are applied. The result of the convolutions and an embedding that represents the user are given as input to Fully-Connected layers, to predict the T next items.

different strategies for recommendation, such as: TopPopular, SPop, ItemKNN, GRU4Rec, and Caser. In particular, ItemKNN and Caser were obtained from DRecPy [5], implemented using Tensorflow. In the case of GRU4Rec, we implemented its alternative version using Tensorflow Recommenders.

The following section describe the datasets, evaluation metrics, and implementation strategies with more details.

4.1 Experimental Data

In this work, we used a single dataset in our experiments: The Last.fm 1k Users dataset, that represents the listening habits for nearly 1000 users and contains the records of more than 19 M listening events [2]. The dataset description is depicted in Table 2.

Table 2: Statistic description of the dataset

Dataset	Last.fm 1k
# Users	992
# Items	1,083,472
# Events	19,150,868
# Sessions	1,040,226

However, the sessions do not have the same number of tracks, and many of them only have a few tracks. We used the distribution of session-lengths of each dataset before choosing a minimum number of tracks per session. This distribution is shown in Figure 6.

As we can see in Figure 6, the dataset has many sessions with a single track. By removing the sessions with less than 10 tracks, the Last.fm dataset was reduced to 47% of its total. However, in order to obtain performance metrics, a higher cut is needed, and we are only using sessions with at least 20 tracks.

One thing to mention is that the interactions are recorded and timestamped, but a definition of session is still needed. Based on the recent work of Hansen *et al.* [11], considering sessions in a Spotify private dataset, we defined a session to be a sequence of tracks with

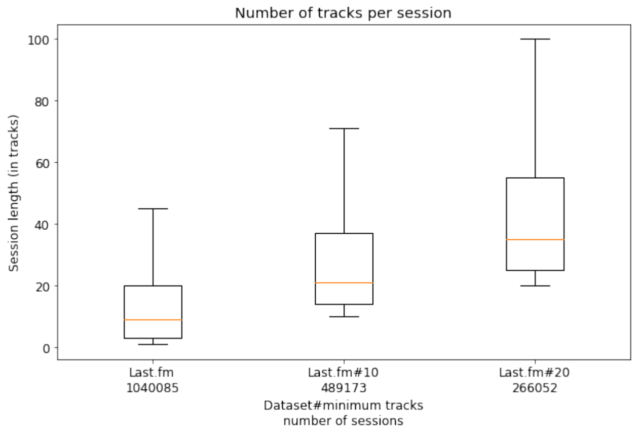


Figure 6: Dataset distribution. # means the minimum number of tracks allowed in a session.

a maximum period of 20 minutes of inactivity between two tracks; if this period is longer than 20 minutes, the tracks are split into two sessions.

Having a clear definition of sessions, we proceeded with a pre-processing step, transforming the data into a sequential string-formatted input. Based on a recent survey in the literature [19], three different strategies were adopted in our scenario:

- **Previous**. The algorithm uses only the last user action to predict L. Considering our previous notation, the input would be $[h_{|H|}]$.
- **Last N**. The algorithm uses the last N user actions to predict L. Considering our previous notation, the input H would be shortened to $[h_{|H|-N}, h_{|H|-N+1}, \dots, h_{|H|}]$. Since N is one of the parameters in this setting, it is possible that H is shorter than N; to solve this problem we apply a left padding, adding a default token until $|H| = N$. In this case, the input would look like this: $[0, 0, \dots, h_1, h_2, \dots, h_{|H|}]$.
- **Sliding Window**. This strategy, also considered to be a data augmentation technique, maps the history of an user into many histories, based on a window size s, given as parameter. Considering our previous notation and $s = 3$, this technique would map $[h_1, h_2, \dots, h_{|H|}]$ into a list of histories $[[h_1, h_2, h_3], [h_2, h_3, h_4], \dots, [h_{|H|-2}, h_{|H|-1}, h_{|H|}]$, with a different L for each element: for example, the first would target $[h_4, h_5, \dots, h_{4-1+k}]$, and so on. This technique is quite frequent in NLP problems, such as Text Generation, and can be compared to the N-gram generation. Although this process is able to enhance the performance of the RS, it is very expensive.

For the Last.fm dataset, we considered a time defined split, in which only the sessions of the last time quarter are used for testing. The time defined cut can be seen in Figure 7, in which the black line represents the split. The resulting datasets are presented in Table 3. However, considering the computational cost of training and testing models on very large datasets, we decided to randomly draw a sample of the training and testing splits, as shown in Table 3.

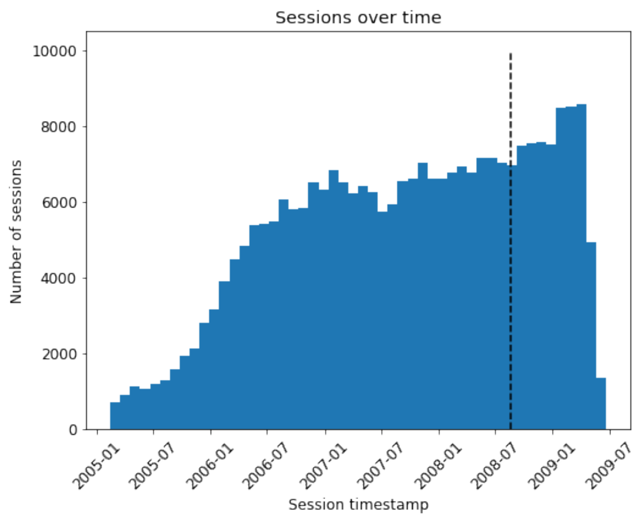


Figure 7: Last.fm dataset time split.

Table 3: Description of the dataset samples used.

Dataset	Last.fm	
	Training	Testing
Percentage	7.5 %	2.5 %
# Users	629	568
# Items	240,195	240,195
# Events	880,992	299,106
# Sessions	18,917	6305

4.2 Evaluation Metrics

In the field of RSs, the quality of the outputs of a model can be evaluated using metrics like Normalized Discounted Cumulative Gain (NDCG), Root Mean Squared Error (RMSE), or Precision and Recall. However, when dealing with Sequential RSs, ranking metrics can be used for performance evaluation. In this work, the performances will be evaluated using two different metrics:

(1) **Mean Reciprocal Rank (MRR)**. This metric evaluates the Reciprocal Rank (RR) for each test item and returns the mean of these values. The RR is the inverse rank of the first predicted item l_1 in the list of true next items; if l_1 is not in this list, the RR is 0. The MRR can be represented by Eq. (5).

$$MRR@N = \frac{1}{N}\sum_{i=0}^{N}\frac{1}{rank_i} \tag{5}$$

(2) **Recall**. This metric denotes the number of relevant items in L by evaluating the number of predicted items that are in the list T of true next items. The Recall can be represented by Eq. (6).

$$Recall@N = \frac{|L_{1:N} \cap T_{1:N}|}{N} \tag{6}$$

4.3 Implementation and Hardware

Upon investigating related work, we could see that contributions to the problem were implemented using different technologies: GRU4Rec was originally implemented [1] using the framework Theano [23], while Caser was originally implemented [2] using Matlab, but the authors also released a version using PyTorch [16].

However, when looking at different applications of NNs, Tensorflow and PyTorch are the most used libraries, and dispose of several popular architectures, making the process of developing and using a NN much simpler. Tensorflow, in particular, has recently released a new extension called Tensorflow Recommenders[3] that focus in making the development of Recommender Systems much easier. Given its integration and speed up with GPUs, we adopted Tensorflow Recommender in this project. By doing so, we also implemented the features of GRU4Rec with this technology.

We performed this analysis using a computer with an AMD Ryzen 3600x CPU, 32GB RAM, and an NVIDIA GeForce RTX 2080 Ti. However, the training time is proportional to the number of training instances, and strategies such as the sliding window may greatly increase the time.

5 RESULTS AND DISCUSSION

Considering the performance reported by Hidasi *et al.* [12], the average values of Recall@20 and MRR@20 for GRU4Rec are around 0.6158 and 0.2888. Tang and Wang [22] evaluated Caser using GRU4Rec as baseline, and while Caser had an average performance of 0.1042 for Recall@10, GRU4Rec only achieved 0.0802. Since neither of these works used the Last.fm dataset, we evaluated their performance once more, and by using MRR@10 and Recall@10, we obtained the results for each model, shown in Table 4.

Table 4: Performance evaluation.

Model	Last.fm	
	MRR@10	Recall@10
TopPopular	0.00040	0.00200
SPop	0.18250	0.06340
ItemKNN	0.07530	0.07990
GRU4Rec	0.14570	0.19610
Caser	0.36170	0.55540
GRU4Rec*	0.00005	0.00001

* Our best implementation of this model.

TopPopular and SPop are the algorithms considered to be naïve, and while they cannot be considered intelligent models, they follow logical heuristics. The music domain that we are working with is not that complicated, and its understandable why an algorithm like SPop works very well: the users listen to a small set of songs in each session, but there are songs that everyone likes or are trending and may be useful to that user, creating an artist or genre popularity bias, natural in this domain.

[1] https://github.com/hidasib/GRU4Rec
[2] https://github.com/graytowne/caser
[3] https://www.tensorflow.org/recommenders

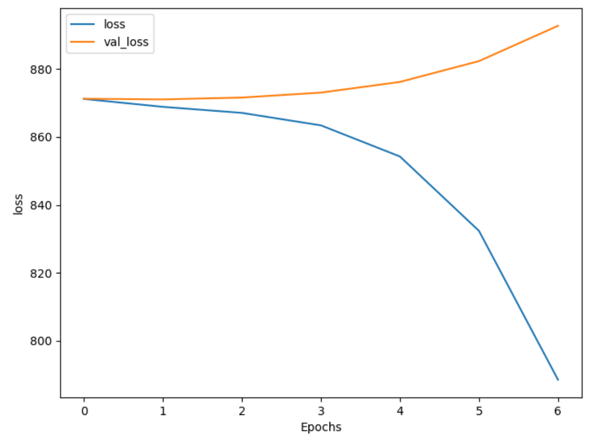
ItemKNN was the algorithm with the highest runtime: 24 hours per run. This long period made impossible to tune the k parameter properly and, among many errors and failed executions, we are reporting the results of the only well succeeded execution. However, some facts are aligned with its performance: it takes a binary matrix instead of a ratings matrix as input, therefore not weighting items according to the interest of the user; while there are only 629 users, there are 240,195 items, making it difficult and slow properly calculate the similarities in this very sparse matrix; at last, the time sequence of events is ignored, and repeated consumption of the item does not contribute to the problem, since the binary matrix is not affected by a higher frequency.

GRU4Rec was the easiest and fastest model to run , taking around 10 minutes to train and evaluate the results. Considering the best architecture and original implementation by the authors, GRU4Rec did not achieve such remarkable results, but also surpassed the baselines mentioned in the proposing paper. Considering that GRU4Rec was one of the first Session-based algorithms, its performance is quite remarkable.

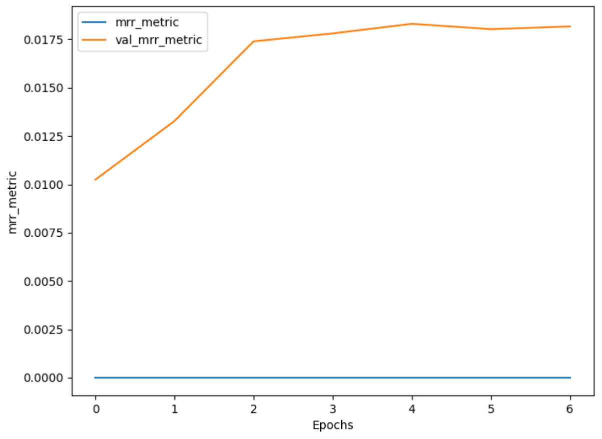
Differently from GRU4Rec, Caser is a Sequential Recommender, taking advantage from some user history to predict the next items. However, training this model needed some previous effort in transforming the data into a specific format and encoding, which was not obvious. After many tries, we were able to run it properly, achieving the remarkable results previously shown. Some characteristics of the model come to mind when we think about its performance: each one of the last n history items is transformed into an embedding, then used as rows of a matrix, which is used as input to a CNN that performs horizontal and vertical convolutions. This sophistication to represent the data is the brightest insight of Caser, and probably the reason behind the good results.

Despite the use of new and recent hardware, when implementing GRU4Rec with Tensorflow Recommenders, we ran into many issues, but the most frequent was insufficient memory, leading us to the dataset reduction depicted in Table 3. Besides this reduction, we also needed to slightly change the NN architecture: while the original work proposed using an 1-of-V vector as input, it was not possible in our experiment considering the large number of unique items, which led us to use Embeddings, a strategy also tested by the GRU4Rec authors but reported to have worse results. We trained the model after implementing this changes, but each input strategy had different results and pointed towards different thoughts:

- **Previous**. When training the model, the epochs after the first already showed signs of overfitting: when looking at the training and validation losses, they were close at first, but while the training loss improved (lower values), the validation loss deteriorated (higher values). This result is shown in Figure 8. Our hypothesis is that the model had not enough variance in the data to predict the next item. Another problem in using only 1 item as input is that during training the same item can be used to predict different items, and the NN cannot approximate $f*$ correctly, since it is not an injective function. This latter problem may also be present in the original GRU4Rec algorithm. The metric values for this strategy are: MRR@10 = 0 and Recall@10 = 0.



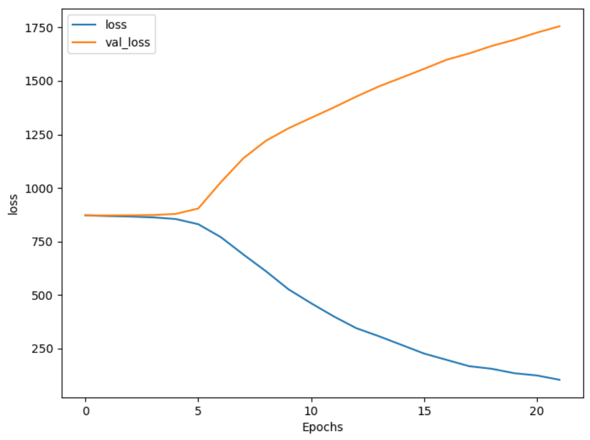
(a) Loss.



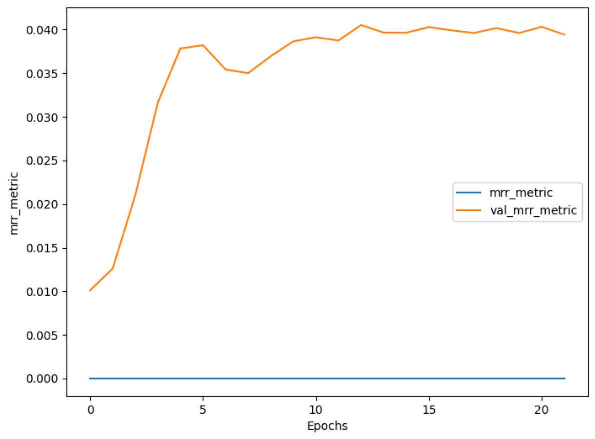
(b) MRR.

Figure 8: GRU4Rec using *Previous* strategy.

- **Last N**. While this solves the variance issue of the *Previous* strategy, the mapping is done slightly differently: the history H is mapped into a single Embedding and not a sequence of Embeddings, something called Session Embedding or Union-level Embedding [11, 22]. We tried this strategy using $N = \{200, 50, 10, 5\}$, but $N = 10$ achieved the best results. As shown in Figure 9, the model is still not learning. The metric values obtained for this strategy are: MRR@ 10 = 0 and Recall@ 10 = $3.17 * 10^{-5}$.
- **Sliding Window**. At first, we evaluated this algorithm using $s = 1$ but it already showed better results, and was the first time that the MRR was better than 0, and a possible insight to understand this improvement is the higher number of training instances when compared with the *Previous* and *Last N* strategies. For this value of s, the metrics obtained were: MRR@10 = $5 * 10^{-5}$ and Recall@10 = $1.5 * 10^{-5}$.

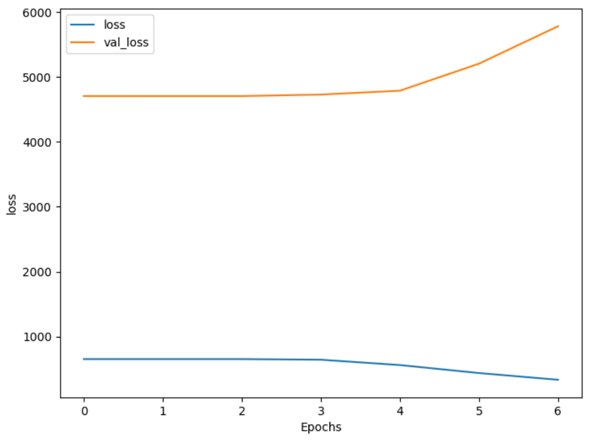


(a) Loss.

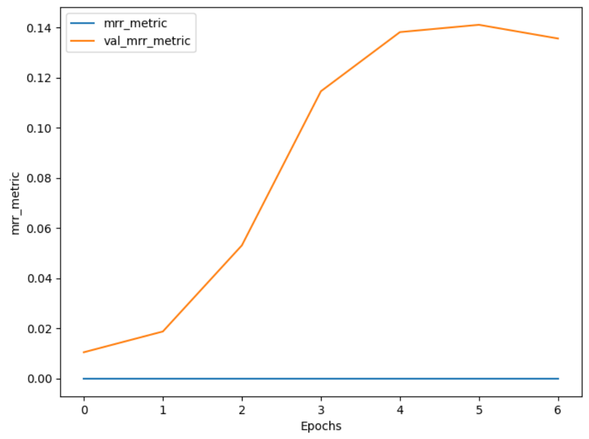


(b) MRR.

Figure 9: GRU4Rec using *Last N* strategy.



(a) Loss.



(b) MRR.

Figure 10: GRU4Rec using *Sliding Window* strategy.

However, the loss was not improving differently than the other methods, and $s = 1$ leads to the same thoughts we had about the *Previous* strategy: when the number of training instances increases, the algorithm begins to learn something, but the non-injective function damages the process. Figure 10 shows these results.

5.1 Combining Best Strategies for Fine Tuning

Having realized which input strategy performs the best, we can apply a process called Fine Tuning: we apply minor changes in the model parameters and hyperparameters, comparing the current and previous results, seeking better settings.

One thing to notice is that the s parameter of *Sliding* can be thought as the N parameter of the *Last N* strategy, solving its lack of training examples while also transforming the *Sliding* strategy into an injective function, a combination that can lead to better results. In addition to that, it is also possible to use different parameters and hyperparameters for the NN, evaluating differences in the learning process. Examples of these parameters and hyperparameters are the number of GRU layers, the embedding and GRU layers dimensions, the dropout probability, etc. The parameters and hyperparameters are depicted in Table 5.

As shown in Table 5, the different settings do not show influence in the model performance, something unusual for NNs. However, considering that Tensorflow Recommenders was recently released, and our implementation is one of the earliest using this library, the lack of performance is understandable.

Table 5: Fine Tuning.

Window	Layers	Dropout	MRR@10	Recall@10
1	[100]	0.2	0.00005	0.00001
1	[100]	0.0	0.00000	0.00001
3	[100]	0.2	0.00000	0.00003
1	[100, 100]	0.2	0.00000	0.00004
1	[32, 32]	0.2	0.00000	0.00003

6 CONCLUSION AND FUTURE WORK

In this work, we applied NLP techniques to Session-based RSs in a case study scenario of music recommendation. In order to provide a baseline for comparison, we considered two naïve methods based on item frequency, Top Popular and SPop, and also two established Sequential Recommendation models, GRU4Rec and Caser. Our proposal consists of a modification of inputs to GRU4Rec, by using the *Previous*, *Last N*, and *Sliding Window* strategies.

Our case study scenario considered a real-world database extracted from Last.fm platform. Our modifications to GRU4Rec were implemented using Tensorflow Recommenders, a recently released framework that had no Sequential or Session-based models, our implementation being the first. By using different parameters and hyperparameters, we evaluated our solution and provided a comparison with existing strategies. The results obtained show that our proposed strategies struggle to make relevant recommendations in a user-independent scenario where the number of items is very large when compared to the number of events. As a consequence of the latter, few examples of item recommendation patterns are available to effectively train the proposed models.

When carrying this work out, we faced many challenges: the datasets were not structured, making it not trivial nor fast to transform the data into useful formats to be handled by NNs, black-box models, and models made available by other authors; lack of clarity regarding configurations in the proposing papers, such as the number of epochs or hyper-parameters; the diversity of programming languages, frameworks, and code patterns in repositories that implement RSs; and, high demand of computational resources, etc.

This work and the challenges faced were beyond what I have previously experienced in classes or other projects, but it helped me to integrate many abilities developed in my undergraduate course in a unique and extensive way. In particular, concepts from Scientific Methodology, Artificial Intelligence, Descriptive and Predictive Data Science, and Natural Language Processing classes were substantial to successfully complete this work.

In future work, we aim at analyzing the performance of other Sequential Recommendation models, enhancing the depth of investigation over datasets, model architectures, cost functions, among other characteristics, while also standardizing their implementations under the same programming language and framework.

ACKNOWLEDGMENTS

I would like to thank my professor advisor Leandro Balby, who presented the opportunity for me to start working with data science, and was open to discussions regarding the most recent research in Recommender Systems and about how I should have done this work. To the research group of the Data Mining Laboratory, which included me in the discussions of the papers and always showed exemplary work.

I also would like to thank the Universidade Federal de Campina Grande, more specifically to the Unidade Acadêmica de Sistemas e Computação, where I had the first contact with Computer Science and provided many opportunities to learn from excellent professors and colleagues, while also providing an excellent infrastructure. To the Brazilian people, always strong and persevering in search of a better quality of life, but that creates opportunity for many with a quality public education service. May we remain resilient in these difficult times.

I would like to dedicate this work to my parents, Edilson and Tamar, who did not fail to support and help in everything they could. To my sisters, Thaís, Elloá and Laura, who inspired me to research, seek excellence, and always do better than yesterday.

I also thank each one of my friends and colleagues, who never stopped teaching me and clarifying my questions, with special thanks to Leonardo Lima, who lent me the hardware to run this analysis, and Ítalo Oliveira, who initially reviewed my writings.

REFERENCES

[1] Charu C. Aggarwal. 2016. *Recommender Systems: The Textbook* (1st ed.). Springer Publishing Company, Incorporated.
[2] O. Celma. 2010. *Music Recommendation and Discovery in the Long Tail*. Springer.
[3] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv:cs.CL/1406.1078
[4] Gobinda G. Chowdhury. 2003. Natural language processing. *Annual Review of Information Science and Technology* 37, 1 (2003), 51–89. https://doi.org/10.1002/aris.1440370103 arXiv:https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/aris.1440370103
[5] Fábio Colaço, Márcia Barros, and Francisco M. Couto. 2020. DRecPy: A Python Framework for Developing Deep Learning-Based Recommenders. In *Fourteenth ACM Conference on Recommender Systems (RecSys '20)*. Association for Computing Machinery, New York, NY, USA, 675–680. https://doi.org/10.1145/3383313.3418483
[6] Ronan Collobert and Jason Weston. 2008. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*. Association for Computing Machinery, New York, NY, USA, 160–167. https://doi.org/10.1145/1390156.1390177
[7] Shaveta Dargan, Munish Kumar, Maruthi Rohit Ayyagari, and Gulshan Kumar. 2019. A survey of deep learning and its applications: A new paradigm to machine learning. *Archives of Computational Methods in Engineering* (2019), 1–22.
[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:cs.CL/1810.04805
[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.
[10] Asnat Greenstein-Messica, Lior Rokach, and Michael Friedman. 2017. Session-Based Recommendations Using Item Embedding. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces (IUI '17)*. Association for Computing Machinery, New York, NY, USA, 629–633. https://doi.org/10.1145/3025171.3025197
[11] Casper Hansen, Christian Hansen, Lucas Maystre, Rishabh Mehrotra, Brian Brost, Federico Tomasi, and Mounia Lalmas. 2020. Contextual and Sequential User Embeddings for Large-Scale Music Recommendation. In *Fourteenth ACM Conference on Recommender Systems*. 53–62.
[12] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. arXiv:cs.LG/1511.06939
[13] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-term Memory. *Neural computation* 9 (12 1997), 1735–80. https://doi.org/10.1162/neco.1997.9.8.1735
[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:cs.CL/1301.3781

[15] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013. Linguistic Regularities in Continuous Space Word Representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Atlanta, Georgia, 746–751. https://www.aclweb.org/anthology/N13-1090

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[17] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162

[18] Tu Minh Phuong, Tran Cong Thanh, and Ngo Xuan Bach. 2018. Combining user-based and session-based recommendations with recurrent neural networks. In *International Conference on Neural Information Processing*. Springer, 487–498.

[19] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-Aware Recommender Systems. *ACM Comput. Surv.* 51, 4, Article 66 (July 2018), 36 pages. https://doi.org/10.1145/3190616

[20] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2015. *Recommender Systems Handbook* (2nd ed.). Springer Publishing Company, Incorporated.

[21] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM '19)*. ACM, New York, NY, USA, 1441–1450. https://doi.org/10.1145/3357384.3357895

[22] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM '18)*. Association for Computing Machinery, New York, NY, USA, 565–573. https://doi.org/10.1145/3159652.3159656

[23] Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688 (May 2016). http://arxiv.org/abs/1605.02688

[24] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Comput. Surv.* 52, 1, Article Article 5 (Feb. 2019), 38 pages. https://doi.org/10.1145/3285029