SessNet: A Hybrid Session-Based Recommender System

by

Omar Nada

B.Sc. Simon Fraser University, Burnaby (BC), Canada, 2017

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the program of

Computer Science

Toronto, Ontario, Canada, 2019

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public for the purpose of scholarly research only.

I authorize Ryerson University to publish the abstract of the thesis, MRP, or dissertation in hard copy or electronic form.

Abstract

SessNet: A Hybrid Session-Based Recommender System

Omar Nada
Master of Science, Computer Science
Ryerson University

Session-based recommendation is the task of predicting user actions during short online sessions. Previous work considers the user to be anonymous in this setting, with no past behavior history available. In reality, this is often not the case, and none of the existing approaches are flexible enough to seamlessly integrate user history when available. In this thesis, we propose a novel hybrid session-based recommender system to perform next-click prediction, which is able to take advantage of historical user preferences when accessible. Specifically, we propose SessNet, a deep profiling session-based recommender system, with a two-stage dichotomy. First, we use bidirectional transformers to model local and global session intent. Second, we concatenate any user information with the current session representation to feed to a feed-forward neural network to identify the next click. Historical user preferences are computed using the sequence-aware embeddings obtained from the first step, allowing us to better understand the users. We evaluate the efficacy of the proposed method using two benchmark datasets, YooChoose1/64 and Dignetica. Our experimental results show that SessNet outperforms state-of-the-art session-based recommenders on P@20 for both datasets.

Acknowledgment

Foremost, I would like to express my immeasurable appreciation and deepest gratitude to my supervisor Dr. Cherie Ding, for her continuous guidance and support throughout my graduate studies. Throughout my research, she provided me with proper direction, feedback and helped me overcome all challenges and difficulties in my work. Her invaluable suggestions, knowledge and research skills helped me throughout my research work, and for the completion of this thesis.

I would like to thank my friends Serena, Ehsan and Preston helping me to beat the state-of-art. I would like to thank Natural Sciences and Engineering Research Council of Canada (NSERC), Yeates School of Graduate Studies, Department of Computer Science at Ryerson, and Dr. Cherie Ding for providing additional funding that helped me through my research. I would also like to thank all members of the Department of Computer Science at Ryerson University for their cooperation, help and for providing me with access to additional resources required for my research. I would also like to thank Dr. Qinmin Vivian Hu, Dr. Alireza Sadeghian and Dr. Alex Ferworn for taking the time to review my thesis and providing valuable feedback which enabled the improvement of the thesis.

Finally, I wish to express my deepest appreciation to the most important supporters. Thanks to my mom and dad for their continuous encouragement and support throughout my life. Special thanks to my wife for her support through my entire career. I wish to also thank my brother, sister and friends for their continuous encouragement throughout my studies.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Background

In recent years, the social interactions have been growing vastly over the internet. Social interactions can vary a lot in definition, where it could be considered mainly as two humans interacting with each other. With the presence of the World Wide Web, it has been much easier for people having interactions even if they don't know each other. These interactions can have different forms, one of them is interacting directly over social networks like Facebook or Twitter. The other one is indirect interaction, which usually happens on e-commerce web site (e.g., Amazon, Netflix, etc.) and is related to online transactions. In this case, if two users are similar in their behaviors and tastes based on past transactions, they may indirectly interact with each other if one selects an item purchased or chosen by the other and thus recommended by the web site. This thesis will focus on the latter form of interactions where recommender system comes in place and plays a huge role in making these interactions as personalized as possible. Many popular websites such as Netflix, Amazon, Google, and Facebook rely a lot on the quality of their recommender systems to enhance the interactions on their websites. As a result, the amount of data has exploded thus giving more opportunities for new algorithms to play a role in personalizing services for the end users.

With this growth, data does not only belong to the end users or customers, producers also have lots of data that can be utilized. For instance, Spotify has stored a massive database to collect all data related to the songs, the singers and any information related to that song and the whole casting crew. All these platforms make sharing information, collaborating with other members and connecting with different parts of the community much faster and more accessible [1,2]. This again has led to having tons of information related to products and services on the web.

Even within a single website, the number of products can be overwhelming for users. Recommender systems can help create a better user experience by helping users find what they are looking for and are interested in. These could also benefit the producers and businesses by aiding them to personalize the target ads, find the next best offer to offer to their clients and increase user interaction with their products or services.

Users are interested in finding the items or products that best suit their needs and tastes without spending too much time on the website. This is done in Spotify by having the "discover weekly" [3] feature which recommends songs that are similar to what the user has listened to before or liked. In the e-commerce website Amazon, the site displays items that are often bought together with the item the user has purchased. Users spend multiple sessions on the websites. A session is considered to be a sequence of items that the user interacted with in a specific timeframe [3]. In a session-based setting, the items that are viewed or purchased together within a session are considered to be correlated. This means that the recommender system can identify and analyze this type of customer behavior as they interact more with the website. The session information with a sequence of items can be used later by various Machine Learning and Deep Learning techniques. This is similar to Natural Language problem domains where there are ordered sequences of words to be analyzed and processed. We could treat a sequence of products similarly as a sequence of words in a sentence, where each word has a correlation with the following words within the given sentence. Session-based recommendations focus mainly on the set of sessions (the sequence of items) as the input to determine what are the product or items to be recommended next. Lately, Recurrent Neural Networks (RNN) have shown great success in dealing with session-based recommendations. RNN is naturally good to deal with and interpret sequence of data because its internal memory can store information of relevant items that have been seen earlier and discard the useless information at the current time step. The drawback of RNN is that it won't be accurate at the

beginning of the session as it utilizes the information as the sequence gets longer. The main advantage is that it utilizes the ordered sequence of information while other models fail to do so. These models often assume that the sequence doesn't play an important role in determining the next item in the sequence, which contradicts the definition of the session-based recommendations.

In one of the earliest papers that integrated RNN into session-based recommendations [4], a simple RNN implementation processed the sequence of items user has interacted with and a list of recommendations was produced as an output. This model has proven to do well in session-based datasets of item interactions and movie domain. In [5], they further improved the previous model by making it more contextual aware. The context could be the time of the day, the price, sale, etc.

The two previously mentioned papers only looked at recommendations per session basis. What has been lacking in the research on session-based recommender system until today is to utilize the history of sessions for a user if it exists. All the current research pay attention only to the current session without paying attention to the previous sessions. Intuitively, there should be a strong dependence between sessions, e.g., in e-commerce site, if a user purchased hiking boots in his last session, he will probably not be interested in another pair in his next session. However, the user might be interested in additional hiking equipment such as a primus stove.

After RNN has proven its strength in the domain of Session-based Recommender Systems (SRS), more research has been done to improve the existing models to increase accuracy and reduce running time. After being very efficient in the NLP domain, Transformers [17] has been incorporated into recommender systems and it has shown additional improvements. In this work, a bi-directional transformer will be introduced for the first time in Recommender Systems. It has been only used in natural language domain in the past. The main idea of the bi-

directional transformers is to mask random items from a given sequence and use the neighboring items to predict what the masked and tokenized item is.

1.2 Goals and Research Questions

The aim of this thesis is to introduce the Transformers in a session-based recommender system to test its effectiveness and compare its accuracies with the traditional recommendation algorithms. The model in this thesis takes into consideration the current session the user is browsing, a representation of all the previous items the user has viewed where sequence isn't preserved, and a representation of all the historical sessions where order of items is preserved.

The bi-directional transformers have claimed the state-of-art performance in Natural Language Processing (NLP) tasks such as machine translation, question answering. In academia, a technique or model successfully applied in one domain can be applied in another domain too. For example, computer vision techniques can be used for Natural Language processing tasks, Natural Language processing algorithms are being used in improving recommender systems and much more. In this thesis, our aim is to adopt the bi-directional transformers for the first time in Recommender Systems. As far as we know, till the time when this thesis was started, nobody has done similar work. In addition to using transformers, the other goal is to utilize the session histories of users to further personalize the recommendations. Our proposed approach is considered to be a hybrid approach between utilizing the features of a session in session-based recommendations and a traditional approach in recommender systems where the history of a user (without considering sessions) is analyzed to find the best recommendations for the given user. For the users who do not have histories in the system (first time login), we rely mainly and solely on the transformer to analyze that specific session to come up with the best possible item recommendations given the sequence (the current session).

In this work, we try to answer the following three research questions.

Research question 1: Can transformer improve results compared to current session-based recommendation algorithms to identify the highest probable next item recommendation given the current session?

Research question 2: Is a deep neural network that uses the history of users useful to further personalize the next item recommendations by transferring the embeddings obtained from the bi-directional layer of the transformer?

Research question 3: Is having a hybrid system combining session-based recommendations and traditional recommendations able to increase the personalization not just for users with history but even for users who have no history at all (first time logins)?

1.3 Proposed Method

The two-stage model SessNet that is proposed in this thesis are two different models, each with its own advantages and power. The first model is the bi-directional model where a transformer layer is used to analyze the sequences of items that are observed within sessions. In this part, all sessions will be passed to the model and it will be trained on cloze task. This means that some items will be randomly masked within a sequence and all the items before that masked sequence will be used to determine the masked item. Each item has a 10% chance of being masked. Once this stage is done, all the item embeddings will be extracted from the embedding layer of the transformer and transferred to the second model which is the deep profiling model. This kind of transfer learning is crucial in this thesis as the goal was to utilize the power of transformer to make the next item prediction easier for the second model. There will be transfer of the learnt embeddings from the bi-directional model to the Deep-profiling model. The deep-profiling model (the second model) consists of a feed-forward neural network where the current session items are replaced with their corresponding embeddings. If the user who is browsing in the current session has history, then all the items they have viewed and the history of their

previous sessions will be replaced with their embeddings and concatenated to the current session representation and fed to the feed-forward network. In case the user has no history (majority of the cases in our datasets), the current session will be padded to make sure all the inputs have the same size for the feed-forward network.

After building the model, it will be compared to multiple baselines on different datasets. One of the baselines should be RNN-based recommender. Our model is an extension to a standard RNN, which means that the model should be able to outperform a standard RNN. We also need to tweak our model, by adjusting parameters and testing different implementations, to perform as optimally as possible. Then we should analyze its performance, trying to find its strengths and weaknesses, and explain why it performs the way it does. Finally, we should see whether we can answer all our research questions by performing various experiments on the model. The first question will be answered by comparing transformers with RNN-based models and other baseline models [7][20][11][8][5]. The second research question will be answered by having two different datasets, one that has no user with history information and the other with some users who have history. Having these two different datasets, we can determine if the model in general is good for session-based recommendations in different scenarios or it performs well only in certain situations. The last research question will be answered by experimenting with the entire dataset with users who have history and users who don't have any history. An experiment will be conducted to run the entire two-stage model only on users with histories once and users with no histories once. These experiments could help to explain if the second-stage model (the deep profiling model) is good for only users with histories or it is good for next item recommendations in general. According to our experimental results, our model has outperformed all previous session-based recommendation algorithms on two different datasets.

1.4 Thesis Structure

In Chapter 2, a deep exploration of the current recommender systems will be discussed where session-based recommendations will be studied more for understanding the current state-of-art model's power and weakness. In Chapter 3, we will describe all the components of each model and explain how these different models are integrated into a single model that can outperform the current state-of-art models. In Chapter 4, we give a detailed discussion of the experiment design, the metrics and the datasets we used, and the results, and how we did the hyper-parameter tuning for the model to obtain the highest accuracy. Lastly Chapter 5 will discuss the conclusion and highlight the main contributions of the thesis.

Chapter 2: Related Work

In this chapter, the main concepts and algorithms of recommender systems will be explained as well as the related RNN based recommender systems. The first section will provide a brief explanation on some existing approaches that make up the baseline models for recommender systems. The second section will include recent work done by other researchers using RNN and transformers.

2.1 Recommender Systems

Recommender system combines ideas from user profiling, information filtering and machine learning to provide more intelligent and proactive recommendations that match user's preferences and needs. In general, recommender system can be categorized into three types: content-based model, collaborative filtering model, and hybrid model.

2.1.1 Content-Based filtering

These systems make recommendations using a user's profile features. They hypothesize that if a user was interested in an item in the past, they will be interested in similar items in the future. Similar items are usually grouped based on their features. User profiles are constructed using historical interactions or by explicitly asking users about their interests. There are other systems, not considered purely content-based, which utilize user's interests which can be collected by comparing the user with other users based on the interaction similarity. One issue that arises is making obvious recommendations because of excessive specialization. For example, if user A has only shown interests in categories B, C, and D in the past, the system will not be able to recommend items outside those categories, even though they could be interesting to them.

Another common problem is that new users lack a defined profile unless they are explicitly asked for information. Nevertheless, it is relatively simple to add new items to the system. We just need to ensure that we assign them a tag or assign them to a specific group or cluster according to their features. By doing so, the new items will have common features with existing items, this is much easier for users to explore these new items and therefore solving the cold start problem for items.

2.1.2 Collaborative Filtering

These kinds of systems utilize user interactions to filter for items of interest. We can visualize the set of interactions with a matrix, where each entry (i, j) represents the interaction between user i and item j. An interesting way of looking at collaborative filtering (CF) is to think of it as a generalization of classification and regression. While in both classification and regression problems, we aim to predict a variable that directly depends on other variables (features), in collaborative filtering there is no such distinction of feature variables and class variables. In short, collaborative filtering systems assume that if a user likes item A and another user likes the same item A as well as another item, item B, the first user could also be interested in the second item B. Hence, they aim to predict new interactions based on historical ones of similar users. There are two types of methods to achieve this goal: memory-based and model-based [1,2,3,4].

1. Memory-based: There are two approaches in this category. The first one identifies clusters of users and utilizes the interactions of one specific user to predict the interactions of other similar users. The second approach identifies clusters of items that have been rated by user A and finds similar items that haven't been shown to the user

before. These methods usually encounter major problems with large sparse matrices, since the number of user-item interactions can be too low for generating high quality clusters.

2. Model-based: These methods are based on machine learning and data mining techniques. The goal is to train models to be able to make predictions. For example, we could use existing user-item interactions to train a model to predict the top-5 items that a user might like the most. One advantage of these methods is that they are able to recommend a larger number of items to a larger number of users, compared to other methods like memory-based due to its scalability. We say that they have large *coverage*, even when working with very sparse matrices.

A popular approach in collaborative filtering is user-based collaborative filtering where similar users are found using similarity scores (e.g., cosine similarity) and for items that the user hasn't interacted with, the system finds similar users and predicts what rating they would give to that item based on similar users' ratings on the item. Another alternative approach is the item-based collaborative filtering. Here, a user-item rating is predicted by identifying similar items instead of identifying similar users. For a given user A, and item B, where we want to predict A's rating of B, a set of items similar to B is first found. Then the rating of B is predicted based on A's rating of other items in that set.

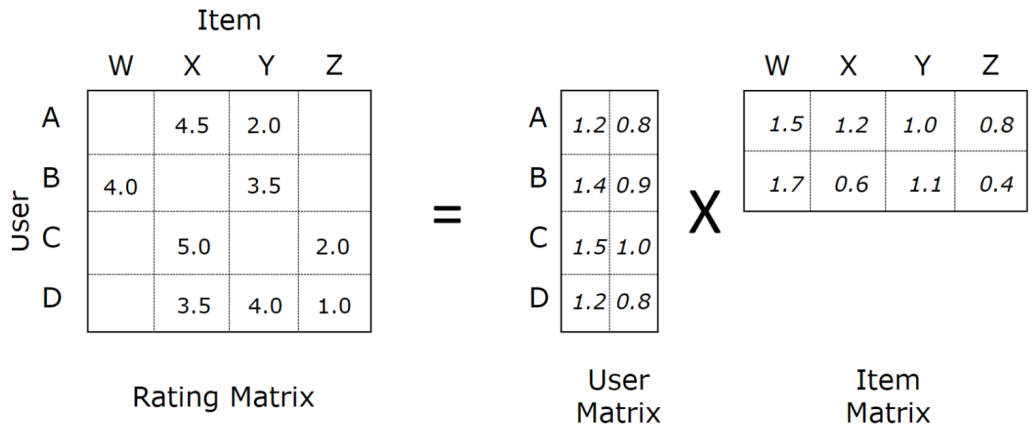
User-based and item-based collaborative filtering are both memory-based methods. They are simple to implement, but they do not work well when the user-item rating matrix is sparse. The popular model-based approach would be Nearest Neighborhood, Bayesian methods, Neural Networks (NN) and latent factor models (e.g., matrix factorization) [5].

2.1.2.1 Matrix Factorization

Matrix factorization is a type of latent factor models. The idea behind this is to represent the users and items into latent factors. These latent factors can better identify the attributes of an

item or the tastes of a user. This representation also helps a lot in finding similar users and items by comparing their latent factors.

Matrix factorization is applied on the big holding/utility matrix (R) and decomposes it into two much smaller matrices that represent users (U) and items (V) respectively as shown in Equation 2.1. After obtaining these two smaller matrices, it then applies the dot product operation on them to yield a rating prediction score for that specific user and item (represented by the two matrices).



Matrix factorizations

Figure 2.1: Matrix Factorization

$$R = UV^T \qquad (2.1)$$

Suppose U is a $m \times k$ matrix, representing m users, and V is a $n \times k$ matrix, representing n items, and k is the number of latent factors needed to represent the data. In the example shown in Figure 2.1, k=2, each row in U is the latent vector for user A, B, C and D, and similarly, each column in V is the latent vector for item W, X, Y and Z. So, R can be reconstructed if we know U and V. The problem is that we do not have access to all values in R, otherwise we would not need a recommender system, and we do not know the value of k. Fortunately, we can choose a value for k and often get a good approximation of R even when the chosen value for k is lower than the actual value [5]. Usually a lower value is chosen to reduce the dimensionality for the

purpose of more feasible and efficient computations. Once a reasonable value of k is chosen, we can use the following equation to approximate R.

$$R \approx UV^T \qquad (2.2)$$

The error between the actual R and our approximation can then be expressed as $||R - UV^T||^2$. Each row $\underline{u_i}$ in U is the latent factor vector for user i, and each entry in this vector describes the user's affinity towards the corresponding latent factor. Similarly, each row v_j of V describes the latent factors of item j. With this, we can predict how user i will rate item j, using Equation 2.3.

$$r_{ij} \approx u_i \cdot v_j \qquad (2.3)$$

To calculate the approximation UV^T, U and V can be initialized with random values. Then the error can be iteratively reduced until it goes below a specified threshold (or for a maximum number of iterations), using a gradient decent method [5,6].

2.1.3 Hybrid methods

Hybrid methods combine both content-based and collaborative filtering methods to benefit from their strengths. Hybrid recommendations can be classified as follows [1]:

1. Implement both collaborative and content-based methods independently and combine their predictions.
2. Incorporate some content-based characteristics into collaborative approach. This approach can overcome some sparsity-related issues.
3. Incorporate some collaborative characteristics to content-based approach.
4. Construct a unifying model that incorporates both content-based and collaborative filtering characteristics.

2.2 Session-based Recommender Systems (SBRS)

Users often interact with systems in sessions, i.e., they interact heavily with the system for a limited amount of time, and then become inactive for some time, before interacting with the system again. A session refers to a short period of active interaction, where the user performs multiple actions. In another explanation by Wang et al. [9], a session can be defined as a transaction with multiple purchased item in one event. Examples of user interaction in a session-based manner include listening to music on Spotify where the songs were listened to in a set window of time. Or, in an e-commerce site like Amazon, browsing through different items within a specific timeframe can be called a session. Many existing recommender systems only consider the last item clicked, such as Item-kNN [23, 26]. Some approaches might take the full user history into account, such as matrix factorization models. However, the connection between the items in a given session and the connection between different sessions are not taken into consideration to come up with the next item recommendation. In many cases, such as for small e-commerce sites, sessions are treated independently, even though they are tracked with a user id. The reason is that most users often visit the site only a few times. Also, even though a user visits the site more than once, they are often interested in something very different from last time. This lack of user profile makes it hard to apply latent factor models [21]. Thus, neighborhood models are mostly used in session-based recommender systems [1]. However, recent work has shown that RNN-based models can outperform existing session-based recommendation approaches such as KNN, Apriori algorithm, or Matrix Factorization [7,8,9,21,24].

Among these session-based recommender systems, NARM [22] uses neural attention models and STAMP [20] uses short term attention to model user's interest drift. Both of these models hold the state-of-art. STAMP claims the state-of-art for the YooChoose 1/64 dataset while NARM holds it for the Diginetica dataset. To test the precision and the effect of recommender

system, precision at *k (P@K)* is used to evaluate the model. In addition to NARM and STAMP, our model is tested against multiple other baseline methods that are very popular in the recommender system domains such as popularity, Item-KNN, FPMC [41] which is a hybrid model between matrix factorization and Markov models. The famous GRU4REC [7] and GRU4REC+ [8] models have also been evaluated on the two datasets. All the results will be discussed in Chapter 4.

2.3 Deep Learning in SBRS

Deep Learning is a field of Machine Learning. It uses computational models that are composed of multiple processing layers of representation and abstraction to help make sense of data such as images, sound, and text. These methods have dramatically improved the state-of-the-art in computer vision, speech recognition, natural language processing (NLP) and many other domains such as drug discovery and cancer cell detection [10]. They have also proven their efficiency to boost recommender system accuracies in various domains like e-commerce, music, videos, media websites, etc. [11]. Many deep learning architectures such as Convolutional Neural Networks (CNN), Recurrent Neural Network (RNN), and Deep Reinforcement Learning (DRL) have been successfully used in recommender systems. We will go in more details on how each of these networks help in recommendation tasks and explore the benefits and weaknesses of each.

2.3.1 Convolutional Neural Networks

Convolutional neural networks have been mainly used in image processing and classification tasks. These are multilayer perceptron networks where each layer involved is responsible to capture only important information and display the information in a way that the decision can

be easily identified. These networks try to mimic the organization of the Visual cortex in the eye. Firstly, the input whether it being a picture, an audio or others is converted into numbers so that they can be processed later. The numeric input is then passed into a layer where the average or the max pooling is performed to reduce the dimensionality of the input. This is very important as an image has a huge number of dimensions, where a dimension can be considered as the width of each image by channel size (which can be considered as RGB). The next layer is a convolutional layer where the output of the previous layer is multiplied by certain weights so that we can obtain the important features from the input. This again reduces the dimensionality and only relevant and important input is kept. The third layer will be a fully connected layer to perform the required classification task. The order and the repetition of each of those layers can be changing depending on the task and the given input.

In the field of recommender systems, CNN has been used in many ways and forms. It is mainly seen in music platforms to enhance music recommendations and provide a personalized list of songs for a user. In 2018, Spotify released a dataset of existing playlists of music and the challenge is to add more songs to each playlist for the user. The winning paper [12] has used CNN and they have won the two awards at that challenge (RecSys 2018). In their work, Maksims et al. [12] came up with a two-stage model where the first stage retrieves 20K relative songs using a Weighted Regularized Matrix Factorization (WRMF) followed by a CNN-based latent model using convolutional neural network. WRMF model accurately obtained a candidate list of songs with high accuracy but without taking into consideration the order of songs. This means that the last song might be more relevant than the first song. This problem is solved by the CNN layer. The CNN applies convolutions on the song embeddings to find the localized pattern with high correlation. To capture patterns that might have been missed by the CNN, an additional neighborhood-based CF model is incorporated. The output of this model is the most relevant 20K songs for that given playlist. The second stage is re-ranking so that

the candidate songs can be ordered to maximize the accuracy at the top-n recommendations. The input to the second-stage model is divided into five groups where the first is the input of the first stage, the second the playlist features, the third the song features, and the fourth creative track where extra features are extracted from Spotify API stating the energy, instruments, danceability, loudness and more features. The last and the most important feature is the playlist-song feature where a pairwise similarity score is calculated between each playlist and the song generated. This two-stage model has successfully captured the personalized next playlist continuation.

Another approach that has been used in the music domain was proposed by Chang et al. [13]. They have used CNN to classify each song based on the audio signals into different genres. A collaborative filtering approach is then used to predict the rating given for each song by the user. The input to the CNN is the song and the output is the classified music for each user where a score of the user and genre are being stored to be used later by the CF algorithm.

Another way to increase the accuracy of personalized music recommendation is to identify user's mood. Abdul et al. [14] proposed an emotion-aware personalized recommendation system (EPRMS). The aim is to find the correlation between user's data such as location, listening history, time of the day, emotion etc. and the songs being listened to. Computing this correlation is done by combining the outputs of two models. The first is Deep Convolutional Neural Networks (DCNN) and the second is Weighted Feature Extraction (WEF). Like their previous paper [13] they have used DCNN to extract music latent features of the song such as the genre, the mood related data such as energy and the pace. The DCNN uses both the songs' meta-data and the audio to classify the song into different genres. The second model (WEF) uses the user features such as the listening history, the categorical location (work, park, school, etc.), as well as the time of the day to compute the correlational score.

CNN has proven its advantage to find the local important features. It can be used in various types of recommender systems. This being said, it has shown average results for session-based recommendations compared to other models. One of the reasons is that CNN ignores the order of items. However, in some cases, sequence is a very important attribute to be taken into consideration as it reflects the change in user's behavior and interest over time. RNN has been proven to do well in such systems.

2.3.2 Recurrent Neural Network (RNN)

RNN has been proven especially effective for session-based recommender systems. Many researchers have done extensive studies to apply different RNN techniques coupled with attention and many other recent technologies. The main advantage of attention is instead of paying attention to each item equally (like in traditional RNN), the model only focuses on relevant items in the sequence. They have achieved good performance in many fields like natural language processing, question answering and language translations. With small modifications in the techniques changing from paying attention to every item to only focusing on relevant items, they can be used to identify and find the next item to be recommended for a given session. Zhou et al. [18] have used attention to determine user behavior. They claimed that RNN by itself have problems in long term dependencies, emphasizing that user's current behavior can be determined from the items that are relevant in their history. It is also claimed that user's behavior drift over time and only certain behaviors can determine their current interests, so it is important to focus only on the important ones through attention mechanism. Their system models interactions of users and items differently for different behaviors. Different behaviors refer to user actions such as clicks, adding to wish-list, purchases, etc. Results of the behavior model are used in a downstream prediction task where vanilla attention is used on the behavior embedding for prediction. In another study by Zhang et al. [19], they

claimed that sequential recommender systems have failed to explicitly capture item-item relationship within a sequence where a regular sequential recommender system focuses more on long term intents rather than the current short term. In order to solve this problem, they have combined regular Collaborative Filtering techniques (CF) along with attention to have a representation for both short-term and long-term intents. The model utilizes item-item relationship during a sequence within a given window to focus on item relevancy and similarity for the next item recommendations. Their model has performed great. It has outperformed baseline scores for 12 different datasets by constructing self-attention of all items with each other, which is achieved by creating a score between all the items and themselves. Liu et al. [20] have mainly focused on user's short- and long-term intentions. A novel short-term attention/memory priority model is proposed to capture user's general interest from the long-term memory of a session context while paying attention to last click of the user. The model has been used on two datasets from RecSys challenge 2015 and the CIKM Cup 2016, scoring the state-of-art results for these datasets.

The models that have been proposed in the past using RNN are extremely efficient in determining the next item to be recommended given a specific sequence. The model discussed in this thesis is built up on this concept by creating a novel approach to further improve and personalize the recommendation results. The idea of the model in this work revolves around obtaining the embeddings of each item learnt from the bi-directional model. The embeddings will be used in the deep profiling model, creating representations of the current session and user's history, which are then fed to a feed-forward network. The output of the feed-forward network is the probability of each item being the next item to be recommended. The output list of probabilities will be then sorted in descending order, where the top-n recommendations will be given (the highest n probabilities). In addition to identifying the next item to be recommended given the current session, this model also incorporates the user preference

(obtained by the history) in predicting the next item to be recommended. This is a mixture between a session-based recommendation and generic recommendations focusing only on the user preference and history.

Traditional neural networks expect all the inputs and outputs to be independent of each other, and they usually require fixed size inputs. In an NLP task such as predicting the next word in a sentence, previous words play a vital role in deciding the words to appear next. However, since sentences have varying number of words in them, it could be problematic for both traditional neural networks and CNN. RNN overcomes this with the help of hidden layers, with the most important feature being the hidden states. These hidden states can act as a memory with some information of the inputs that were passed into the network earlier. The reason it being called Recurrent is that the same step gets computed multiple times, each time depending on the hidden state of the previous step. As shown in Figure 2.2, at each time step, the network takes in external input and input from the last time step, and two outputs are created.

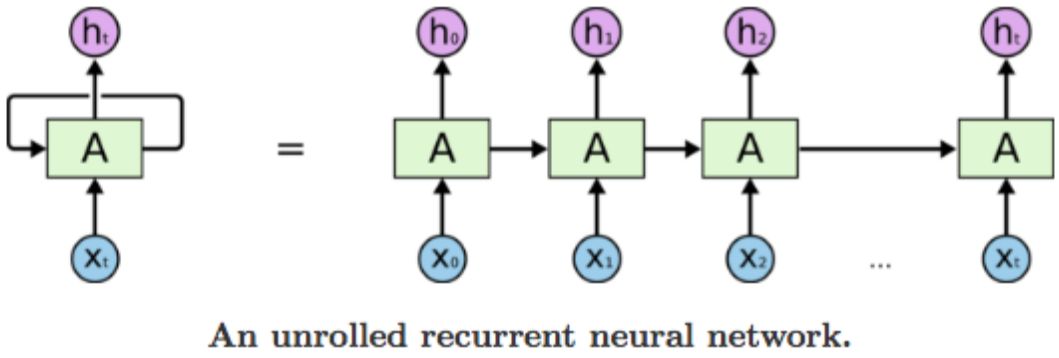


Figure 2.2: Unrolling of a Recurrent Neural Network [7]

One output is passed on to the output layer, or the next hidden layer if there is one. The other output is the state of the RNN, which is passed on to the next time step. Each time step does not have to be separated by a fixed amount of time, and it is just when input arrives. It is fully possible to apply deep learning to RNNs. One can stack multiple RNN layers, or add other types of layers such as a feedforward layer. RNNs are not constrained to either fixed input sizes

or output sizes. Although the size of each of these input vectors has to be fixed, RNNs are not limited to a fixed number of such vectors as they can take as many words as needed. RNNs can be applied both to domains where it is natural to treat the data as sequences, and to domains where the amount of input is fixed. With these features of RNN, it makes sense to tackle session-based recommendations using RNN. In this case, sessions replace sentences where the words are items.

2.3.2.1 Long-Short-Term Memory (LSTM)

In the very beginning of RNN, training was difficult due to the vanishing and exploding gradient. When the number of steps increases, the gradient often grows very large (explodes) or approaches zero (vanishes). This is due to the usage of the same weights, which causes the pattern to stay the same. This can be solved by introducing the concept of memory and the model is called LSTM. In this model, each node is replaced by a memory cell. The cell can control what to remember and what to forget from the previous time steps. This is done by the help of “gates” which regulates the information flow of the network passing only relevant information from the recent inputs and keeping the important information from the past.

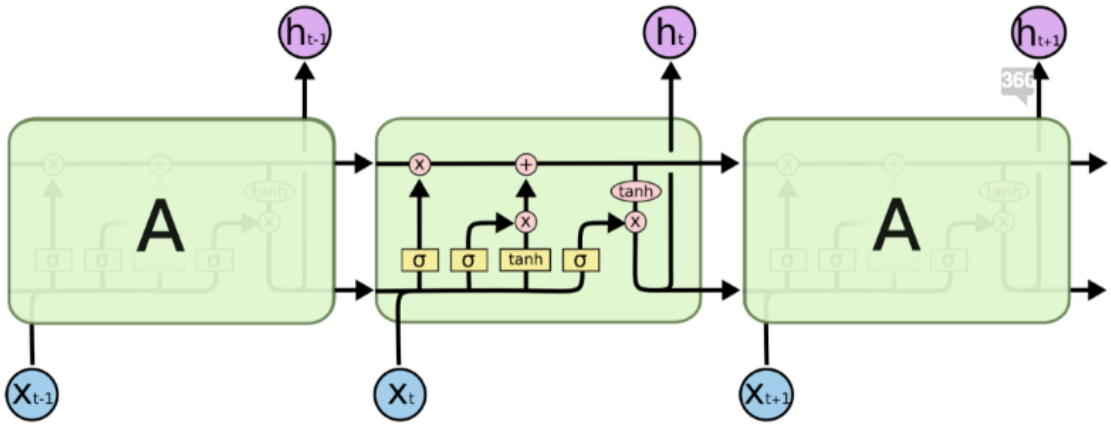


Figure 2.3: LSTM unrolled nodes [7]

There are three gates in LSTM node as shown in Figure 2.3. First is the forget gate which controls what information to be kept and what information to be discarded when moving to the next time step. The second gate is the update gate/input gate, which updates the cell's hidden state. The third gate is the output gate where the next hidden state containing information from previous time steps will be determined. In these gates, the two main activation functions are Sigmoid and tanh.

There are multiple steps that happen in each node. Firstly, the previous hidden state and information from the current input are passed through the first gate which is the forget gate using Sigmoid function. If the value is closer to 0, the information is to be forgotten and if it is closer to 1, the information is passed along. In the second step, the previous hidden state and the current input pass through the input gate using both tanh and sigmoid functions in parallel resulting in two sets of values. These two values are then multiplied to produce a single output. In the third step, the cell state is calculated. This is done by multiplying the cell state with the result of step 1 (forget gate) and then added to the result of step 2 (input gate). The last step happens in the output gate where the previously hidden state and the current input are passed through a sigmoid function. The resulting value is multiplied with the value obtained by passing the new cell state (from step 3) through tanh function. The output of this is the hidden state.

The output of step 3 (new cell state) and step 4 (hidden state) is then carried over to the next time step. Figure 2.3 earlier, clearly indicates the flow of information and the activation functions applied to each set of inputs.

2.3.2.2 Gated Recurrent Unit (GRU)

With a very similar structure as the LSTM, GRU has proved its success in various situations. While LSTM has 3 gates, GRU has only 2 gates, which are the Update and the Reset gates. The update gate decides how much of the previous memory to keep around while the reset input gate defines how to combine the new input with the previous value. Another difference between the GRU and LSTM is that in GRU there is no persistent cell state distinct from the hidden state. Most of the other functionalities are common between LSTM and GRU. Figure 2.4 shows the different paths for the input and the hidden state obtained from the previous time step. It also shows all the activation functions being applied to these inputs. Advantages of GRU over LSTM is that GRUs are faster to train and need fewer data to generalize. That being said, if there is no enough data LSTM's extensive computation could potentially lead to better results. Both of the models have very comparable performances, which therefore makes it hard to recommend one or the other for a specific task.

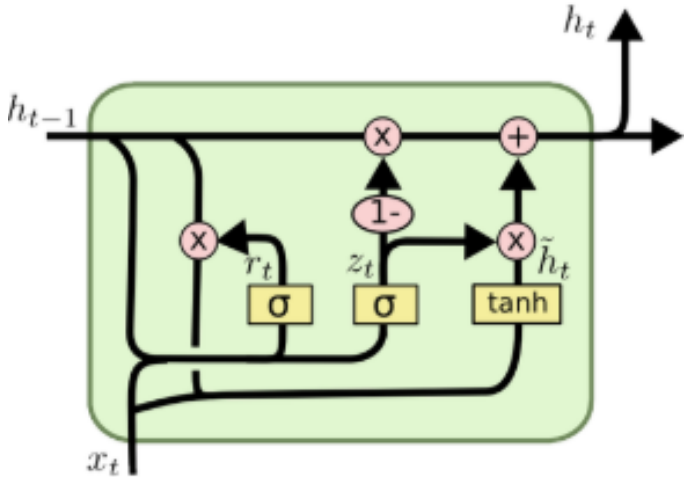


Figure 2.4 A single GRU node where the r_t resembles the reset gate and the z_t resembles the update gate [7]

In the next chapter, there will be deeper explanation of how RNN and its variations help to enhance modelling user's behavior and finding patterns that could possibly improve the accuracies in the next item recommendation domain.

Chapter 3 – Methodology

In this chapter the model will be explained in more details. The chapter will start with reviewing the deep learning models for NLP tasks and how the field has been evolving. Deep learning models including GRU, LSTM and transformers have been used in NLP tasks to identify the next word in a given sentence, which is similar to the problem being solved in this thesis. The similarity can be seen if we treat words as items and sentences as sessions, where the model's aim is to predict the next item to recommend given the current session. The sessions are considered to be sequence of items that a user has interacted with. An interaction is any browsing, purchasing, adding-to-cart or other action taken by a user towards a specific item. In later sections, the main components of the model will be explained followed by a detailed description of how the model is divided and the main goal of each part of the model.

3.1 Related Models

In this section, we will start by describing the relevant models and their limitations before describing our solution.

The first architecture to start with would be transformers [17] where the idea of attention is being utilized. It is mainly used to solve the problems of sequence transduction and neural machine translations. For such tasks, it is required to have memory presence to be able to link future words that will be seen later in the sentence to the right entity that was mentioned earlier and to resolve future dependencies. This would be a difficult task for LSTM or GRU when the sentence is too long. This is because when a sentence is too long, these two models forget the content of the distant positions in the sequence. Another big problem that faces LSTM and

GRU is that it is hard to parallelize the work for processing sentences since the processing happens word by word.

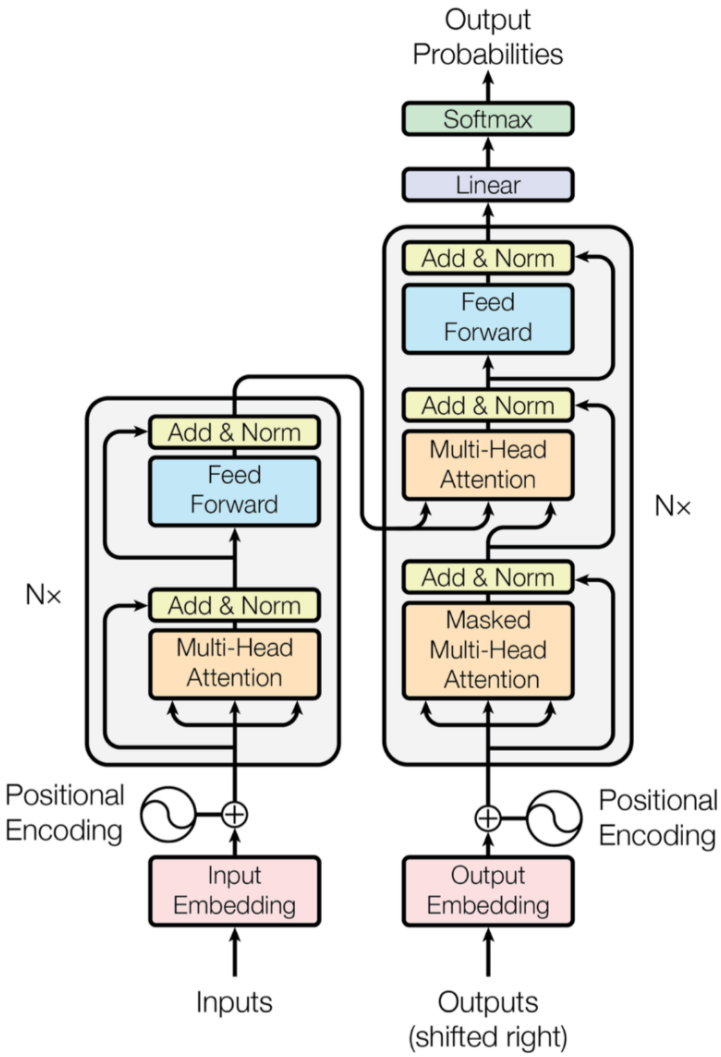


Figure 3.1 Transformer structure [16]

In an attempt to solve these problems, researchers have introduced the technique of attention [16]. In this case, the model focuses on part of a sentence where there is a lot of information. In attention, each word is encoded separately and passed all the way to the decoder. As shown in Figure 3.1, there are several encoders stacked over each other and the same number of decoders for the decoder stack. Each encoder is similar in structure, but they do not share weights. For the encoding step, each word is used to obtain the hidden state for that word and

then passed to the decoding stages. The decoding stage uses that hidden state to pay "attention" to the position (word or item) with the highest relevance in the input (sentence/session).

Self-attention is another form of attention where a score is given between all the words in an input. As each word enters the encoder, the first step in self-attention is to create three vectors from each of the encoder's input vectors. For each word, a Query vector, key vector and value vector are created by multiplying the embedding by three matrices that contain all the weights for the corresponding 3 vectors (Query, Key and value) that were obtained during the training process. All the Query vectors are stacked together to make the W^Q. The same will be applied to create the W^K and W^V. These new vectors are smaller in dimension than the embedding vector, while it is not mandatory that they must be smaller. As shown in Figure 3.2, multiplying X_1 with the matrix W^Q will produce q_1, the query vector associated with that word. The same process will be applied for the Key and the Value matrices thus producing a key and value vector for each word. These three vectors are abstraction for the word, which will be later used to calculate the attention between this word and all the other words in the input sentence. The second step is to calculate the self-attention score for each word against all the other words. The score determines how much focus to place on other parts of the input sentence as a certain word being encoded at a specific position. The score is calculated by taking the dot product of the query vector with the key vector of the remaining words. The third step is to divide the score obtained from step 2 for each word by the square root of the dimension of the query vector (in the example, 64 was the dimension used for the vector) leading to stabilizing the gradients. The fourth step is to perform softmax function on all the words in the dictionary of all possible words and then get the normalized scores. This score determines how much each word will be expressed at this position. Clearly the word at this position will have the highest softmax score. The fifth step is to multiply the score obtained from step 4 with the value vector. The intuition behind this is to only focus on the relevant words and multiply the other words

with values like 0.001 (from the softmax score) thus ignoring irrelevant information. The resulting vector will then be fed into the neural network to determine the next word in a given sequence. This process is shown in Figure 3.3.

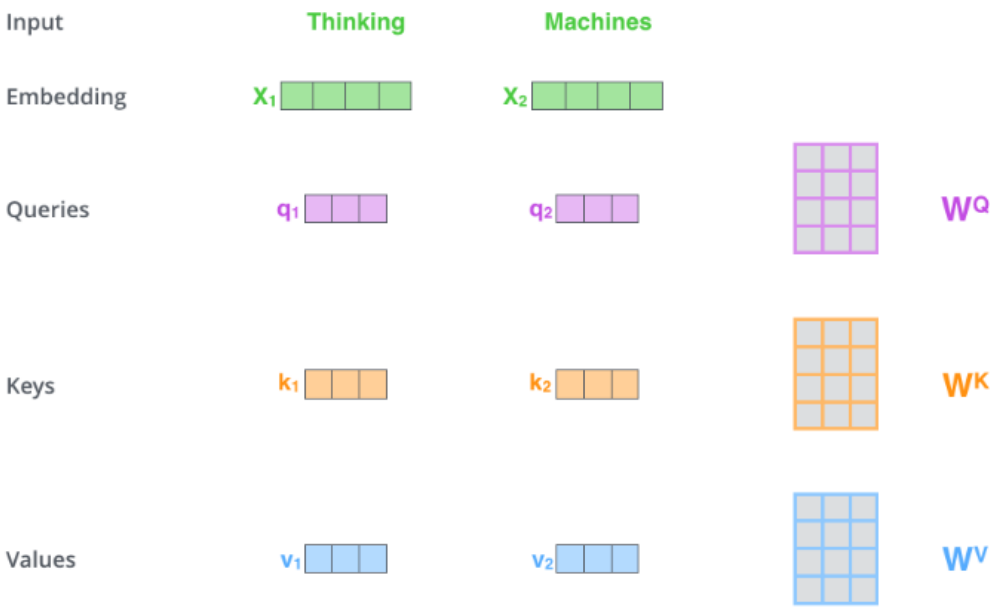


Figure 3.2 “Query”, “Key” and “Value” vectors [16]

In addition to the mechanism that is explained, transformers also use the multi-head attention. The reason is to pay different attention to different word based on the position of the word. Another useful technique that was used in this thesis is the positional encoding where the position of each word plays a part in the encoding step by using a sinusoidal function.

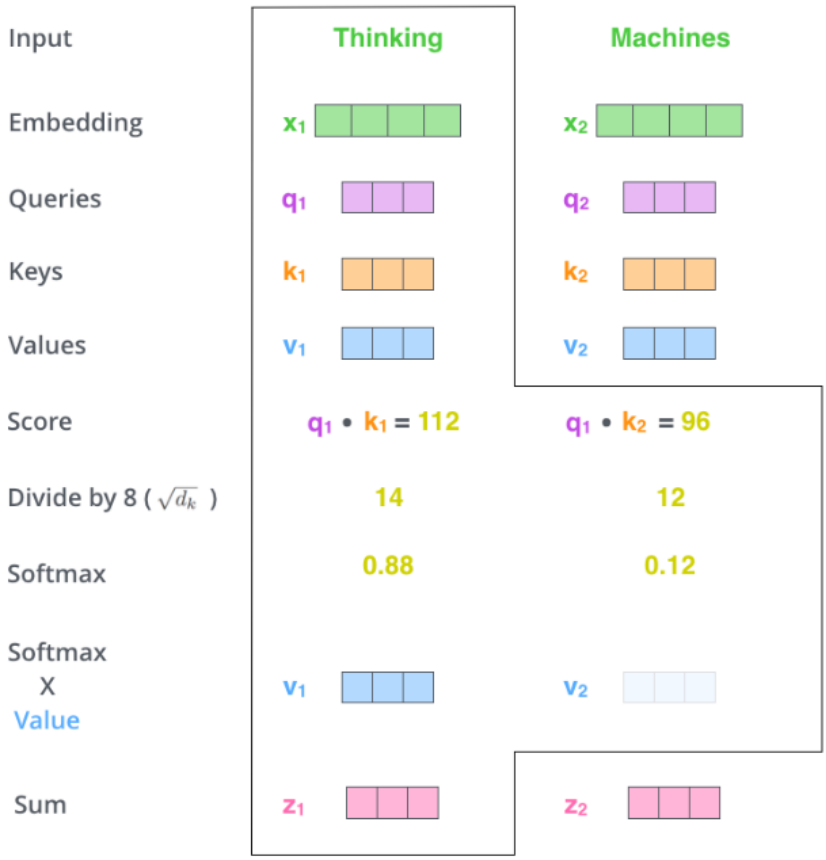


Figure 3.3 showing the calculations to obtain the self-attention between each word and the remaining input sequence. [16]

3.2 Model Overview

Now that the idea of transformer and self-attention has been discussed and explained, this section will explain briefly the architecture and structure of SessNet, which is our proposed model. The goal of this model is to predict the next item recommendation given a sequence. The model is a two-stage system as illustrated in Figure 3.4. The first component bi-directional transformer is responsible for applying Transformers encoders to provide embedding for each item. Until the time this thesis is written, transformers have been only used in NLP tasks predicting the next word and has never been introduced to session-based recommendations domain. The second component, Deep Profile will utilize these embeddings to perform better

and more accurate profiling for each user that will be later fed to a fully connected layer to produce a score for each given user and a product.

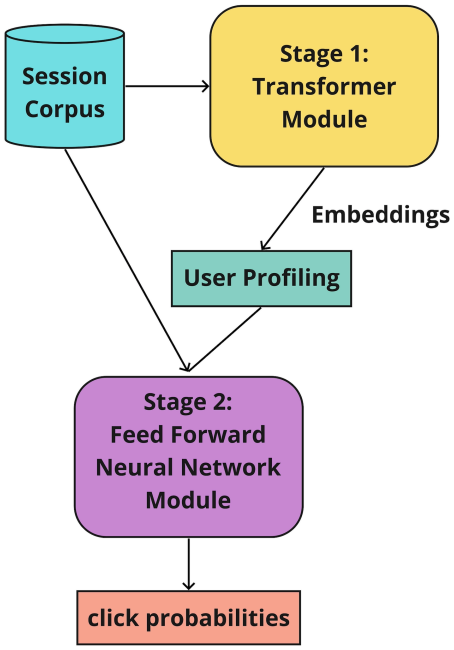


Figure 3.4: The overall Model with Stage 1 (Bi-directional) and Stage 2 (Deep Profile)

As mentioned earlier, the goal of this model is very similar to next word prediction. Therefore, in this model, the words are replaced with items and the sentences are replaced with sessions and the goal would be to predict the next click item given a list of items in the current session for a particular user. The input to the first model will be sequence of items and the main task is to accurately obtain an embedding for each item, where items with high co-occurrence frequencies will have closer embeddings. The task of the second model is to use those embeddings and provide more personalized results for all users that have historical interaction data recorded (more than one session). Those two models are trained and tested independent of each other. This means that each model will be optimized separately and the result of the first model will be propagated as reference for the second model. The first model should provide an embedding for each item that will be later used to provide the user representation and session representation in the second model. The second model, Deep Profile personalizes

the recommendation for a given sequence using the history of users and their previous session representations.

3.2.1 Bi-Directional Transformers (First Model)

The first model's task is to utilize all the sequences of items and come up with an embedding for each item given the co-occurrences that happen within the sessions. This model is identical to BERT model where several encoder transformer layers are stacked on top of each other. In BERT [17] there are two different models, the BERTbase where there are 12 Transformer layers stacked on top of each other, and the BERTlarge where there are 24 transformer layers stacked. In our model we use a very similar structure to BERT except that we are only using 4 transformer layers as there are much more words than items. The number of transformer layers has been decided after we did a preliminary test trying different values (2, 3, 4 and 5). The results will be explained more in the next chapter. The input to our model will be all the sequences of items while the first input token is a special [CLS] token. This token is the classification token. The model takes sequence of items as input which keeps flowing up the stack. Each layer applies self-attention and passes its results through a feed-forward network and then hands it off to the next encoder.

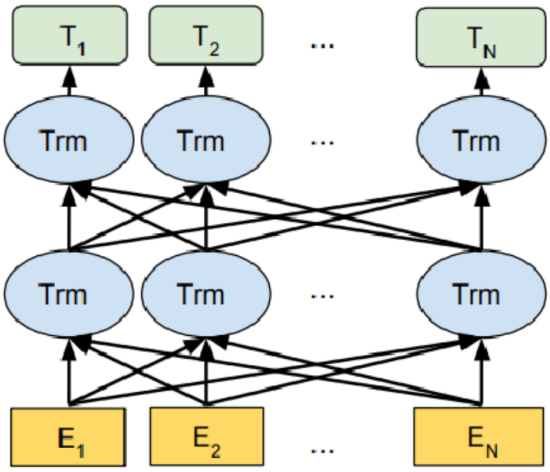


Figure 3.5 Tokens process [17]

Just like BERT, our model makes use of Transformer encoding layer where attention mechanisms learn contextual relations between items in a session. Before feeding item sequences into the model, 25% of the items in each session are replaced with a masking token [MASK]. The model attempts to predict the original value of the masked items, based on the context provided by the other non-masked items in the sequence. An item starts with its randomized initial embedding representation from the embedding layer. Every layer does some multi-head attention computation on the item representation of the previous layer to create a new intermediate representation. All these intermediate representations are of the same size. There will be 4 intermediate representations as this is a 4-layers model. Each layer has 512 units with 8 attention heads in total. As shown in Figure 3.5 E_n is the embedding representation of each item, T_n is the final output of that word and T_{rm} is the intermediate representation of the same token.

In order to predict the next item, there must be a classification layer on top of the encoder output where the output vector is multiplied by the embedding matrix transforming them into dimensions. Lastly using softmax, probability of each of the items in the entire collection will be computed and the one with the highest probability will be recommended next. The BERT

loss function takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked items. As a consequence, the model converges slower than directional model, a characteristic which is offset by its increased context awareness.

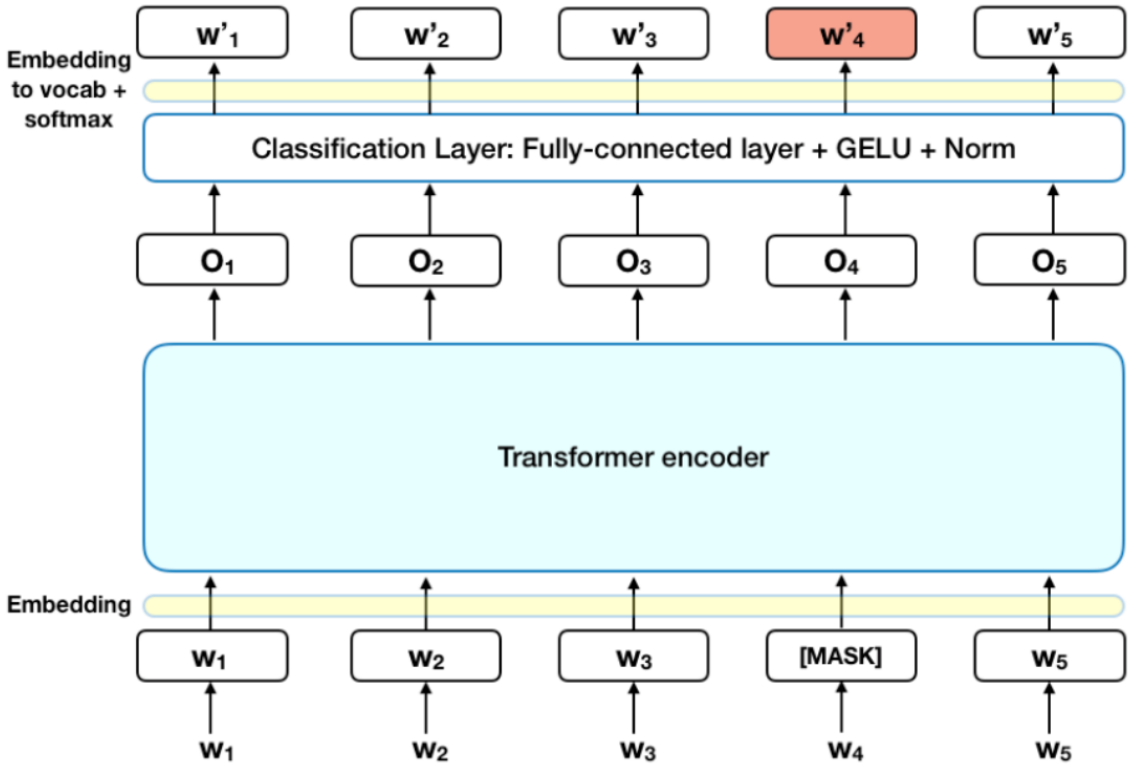


Figure 3.6 The full structure of BERT [17]

Now that the architecture and the working procedure of the model has been explained (illustrated in Figure 3.6), training happens and those trained embeddings are extracted. After training finishes and all the embedding weights are updated, the embedding layer (the first layer) is obtained and those weights are then extracted to create the lookup dictionary which maps every item using the item id to the trained embeddings. This lookup dictionary will be later used to convert all the user profile, the session history and the current session into list of embeddings to be fed to the Deep Profile model (second component of this model). As mentioned earlier, training for the first stage is independent from training the second stage,

because our main goal is to extract the best item representation given the sequence of items observed by the bi-directional transformer.

3.2.2 Deep Profiling (Second Stage Model)

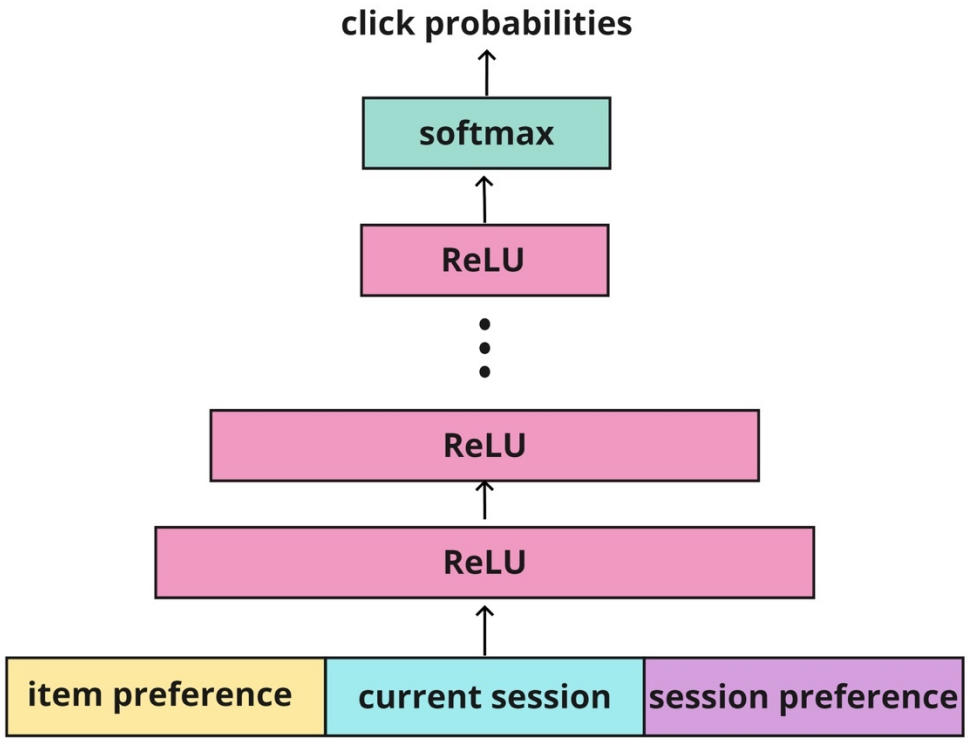


Figure 3.7 The Deep Profiling Network structure

Now that all our items have corresponding embeddings provided by running the first model and after extracting those trained embeddings from the embedding layer, we can use those embeddings to start the profiling stage. There are three types of profiling that will be done: (i) Item Preference/User Profiling, (ii) Session Preference/Session History, and (iii) Current Session. To the best of our knowledge, up to the date of writing this thesis, this profiling model is novel in the field of session-based recommendations. All the existing research works in this field have been solely focused on identifying the next-item based on the sequence of items that

precedes the next-item to be recommended. The idea here is that, in addition to identifying the next item recommendation given the sequence, we want to personalize each recommendation given users' past interactions on items. The next-item recommendation can be achieved by relying solely on the bi-directional transformer layer (first stage model). This second stage focuses on the personalization part. In case where there is a user with no history (first time user) or users that are not logged in, the first stage model will be sufficient to recommend the next-item. The deep profile will further personalize the next-item recommendations for those users who have history by giving more preference to the items they have interacted with in their previous sessions. In this section, a more detailed discussion is given on how the profiling is done and why it helps.

User Profiling

In the data pre-processing step, a bag-of-items is constructed by iterating through all the sessions for a given user. This helps in determining the user's interest based on the items they have viewed. Since we are more interested in what the user's taste is and what they like, the sequence and the repetition of the items haven't been taken into considerations in this profiling step. The bag of items in this case is just a set of all the unique items that the user has interacted with in their entire experience with the items in scope. This process will help to capture user's preference and taste, which will help greatly in personalizing the recommendations made to that user. When two users have the same sequence of items, usually with a standard session-based recommender system, they will probably be recommended the same item to click on next, which may not be desirable in many situations. Although sessions from two users are almost identical in this case, the history of each user might be totally different, thus recommending the same item is not considered to be personalized. Due to this reason, in this work, we have decided to use the embeddings obtained from the first model to represent the

users using the bag of items they have viewed. This will lead to difference in size of each bag of items for different users, because each user most likely will view different number of items compared with other users. The way we handle it is that each item for a given user is first replaced with its embedding with dimension *d*. All the n items that user U_x has viewed will then be transformed to a list of n embeddings, each one with dimension d. Max pooling will be then applied to this list of embeddings, thus at the end yielding only one d-dimensional array representing the user. Obviously if the user has only one session, the user representation will be identical to that one session (current session). A very important note is that the bag of items is being constructed given a specific timestamp. This means that all the items that are viewed by the user before a given timestamp will be collected and all the later ones are considered outside the scope. This will be crucial for training and testing in later stages.

Session History

In addition to representing users using their bag of items, we will also consider their session histories. The bag of items simply shows the items that users were interested in, without taking into consideration of the sequence, the number of times that they viewed an item and the number of sessions they had with the system. All of these aspects are very important for recommender systems, especially session-based recommender systems, while they are often overlooked in previous research works in this area. In this work, we want to take sessions that a user has interacted with as additional input to the Deep Profile network to make sure that these aspects are considered in the recommender system. Each user may have multiple sessions. For example, user U_x has a set of 4 sessions $\{S_1, S_2, S_3, S_4\}$. Obviously, each session includes a set of items that the user has interacted with. In this case S_1 for example is $\{I_1, I_2, I_3, I_4, I_5, I_6\}$. Each of these items will have specific embedding/representation obtained from the first model. Each of these items will be represented with the corresponding embedding. Therefore, we will have 6 different embeddings for session S_1. Similarly, we have different number of embeddings

for all other sessions. Since it is clear that each session might have different number of items, max pooling will be applied again to make sure that all the sessions will have the same number of dimensions. Once we have a unified representation for each session, max pooling will be applied to get the representation for the history of sessions for each user. This is because each user will have different number of sessions. It is also important to keep in mind that the history of sessions for each user will be collected before time ***t***, where ***t*** is the time when the user representation is given to the model for training. This representation will help identify the type of items that the user's previous sessions have been revolving around. It also helps partially solve the interest shift problem in recommender systems.

Current Session

Last input to our model is the current session. This is the session that the user is undergoing at a given moment. Our aim is to predict the next item given the items viewed so far in the session. Again, each session may have different sizes as they have different number of items. Similar to the previous two representations, the items will be first represented with their corresponding embeddings followed by max pooling to get a singular representation for all the items in that session.

Now that we have these three representations, all with the same size *d* where *d* is the embedding size, each of the inputs are fed directly to a feed-forward fully-connected neural network as shown in Figure 3.7. Lots of testing has been done trying different sets of parameters and hyper-parameters for both networks, which will be further explained in Chapter 4. The final network consists of three layers. The first layer has 512 nodes with ReLU as the activation function. ReLU has been used as it is known to be the best activation function that provides non-linearity. The second layer contains 256 nodes with ReLU as activation function too. The last layer is the output layer with 64 nodes and uses Softmax as the activation function to give probability

for each given item in the collection. From there, the top 20 items with the highest probability values will be recommended to the user.

Representing the user history, the session history and the current sessions with embeddings obtained from the bi-directional transformer layer is a novel approach that has led to state-of-art on two different datasets. More explanation about the datasets and the results will be provided in the next chapter.

Chapter 4: Results and Discussion

In this chapter, we will explain our experiment design and discuss and analyze the results we have achieved. Different hyperparameters are tested for both models. As mentioned in the earlier chapters, the two networks are trained and tested separately. This means that there is no cross-training happening between the two models. The first model is trained first and the best hyperparameters for that stage is found on the dataset. The embeddings learned from the first tuned model is then used to start the training process of the second model.

In our experiment, we consider the number of transformer layers, the batch size, dropout rate, the dimension size of each session and number of epochs as the hyperparameters in the first model. For the second model, we consider the size of each layer, the number of layers, the number of epochs, the dropout rate and the batch size as the hyperparameters.

We will first perform the hyperparameter tuning for the first model. Once we find optimal values for all the hyperparameters except one for the first model, the same process will be repeated for the second model. We divide the dataset into three parts – 50% as the training set, 30% as the validation set and 20% as the testing set. All the hyperparameter tuning has been done on the validation set and tested only once on the testing set.

4.1 Dataset and Evaluation

4.1.1Dataset

We evaluate SessNet on two datasets. The first dataset is YooChoose from the RecSys 2015 Challenge [21], which consists of six months' session data obtained from an e-commerce website of the same name. The second dataset is Diginetica, which is from the CIKM Cup 2016 [22], from which we use transaction data for our experiments. The transaction data consists of

the sequence of items a user browsed with no detailed actions provided. This is only the item sequence sorted by timestamp. Both datasets follow the same pattern where the only data provided is sequences of items along with the corresponding timestamps.

As in STAMP [20], we follow the same pre-processing steps to make sure that the comparison is fair and valid. We filter out sessions of length 1 from both datasets, as well as items that appear less than 5 times. We preprocess YooChoose dataset so that the sessions in the test set happen after those in the training set with respect to time, therefore removing any lookahead bias, and we also filter out items that appear only in the test set. We preprocess Diginetica dataset so that the test set consists of sessions that occur in the subsequent week, with respect to the sessions in the training set. After pre-processing, the YooChoose dataset has 7966257 sessions with 31637239 clicks and 37483 items, and the Diginetica dataset has 202633 sessions with 982961 clicks and 43097 items.

Similar to STAMP, we preprocess sequences so that for an input session $C = \{c_1, c_2, \ldots, c_n\}$, we generate sequences and corresponding labels $([c_1], c_2), ([c_1,c_2], c_3),\ldots, ([c_1, c_2, \ldots, c_{n-1}], c_n)$, for the training and test sets. A label in this case will be the true item that comes after the sessions in the test set. Our model's task is to precisely predict that item from the top n recommendations. Since YooChoose dataset is very large, similar to what is done in STAMP, we choose to experiment on the most recent 1/64 of the data.

4.1.2 Evaluation

We focus on precision at K (P@K), as the metric for evaluating the performance of our solution. P@K is a common measure of predictive accuracy [24, 27] for session-based recommender systems. It represents the proportion of relevant items of the top-K recommended items in a recommendation set. P@K is defined by:

$$P@K = n_{\{hit\}} / N \qquad (4.1)$$

where N is the number of recommendations, and $n_{\{hit\}}$ is the number of recommended items in the top-K that are relevant.

4.1.3 Configuration of the experiment environment

The two stages of SessNet are trained separately. For both stages, we used P3.2xlarge AWS instances with 61 GB of RAM and a single Tesla V100 GPU with 16 GB of VRAM.

4.2 Results

4.2.1 Bi-directional model Tuning (First model)

In this section we will test the effect of each hyperparameter and how their values affect the accuracies. The first test is on the dimension size, where we test different sizes for item representation. The purpose of the first model is to represent each item as a *d* dimensional vector. This is an important hyperparameter as we want to make sure that the similar items are represented close to each other in the *d*-dimensional space and dissimilar items are far from each other. If the dimension size is too small, the difference between items won't be represented well. On the other hand, if the dimension size is too large, similar items will be represented farther apart in the *d*-dimensional space. Therefore, it is essential to find the best dimension size. For testing purposes, we have tried values in the range from 20 to 140 as the dimension size and the results are shown in Figure 4.1.

In this stage, we will fix some of the hyperparameters. One of those fixed hyperparameters will be the sequence length, as this number is deteremined by the biggest sequence of items. All other sessions/sequences that are smaller in size, are padded with 0 to make sure all sessions have fixed size.

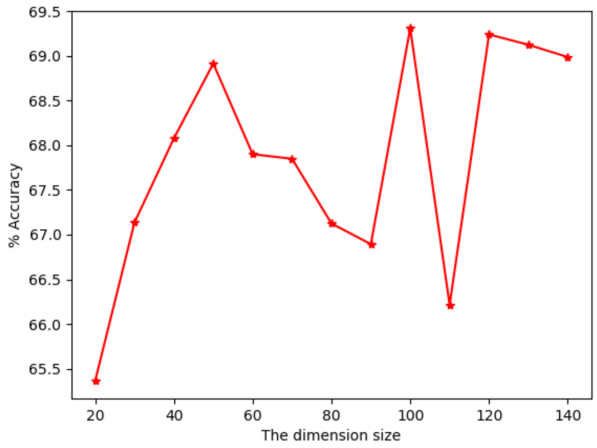


Figure 4.1: change of P@k with different dimension sizes on YooChoose 1/64 dataset

In Figure 4.1, we can see that the dimension size of 100 gives the highest accuracy where it is clearly not overfitting. Small sizes like 50 will be overfitting as the small size will force some items that have different embeddings to be placed closer together. On the other hand, big sizes have two drawbacks, the first being that similar items will be placed much farther in that dimension space than they are supposed to be. Second drawback is that as the dimension size increases, the computation power increases as well, requiring more resources. Therefore, 100 was chosen to be the size to represent items and sessions.

In the second experiment, we test the number of transformer layers. We tested only 2, 3 and 4 transformer layers and the accuracies are calculated. It was hard to test different numbers of heads due to limitations in computation power during our experiment. In our experiment, we tested with the number of heads being fixed to two as shown in Figure 4.2.

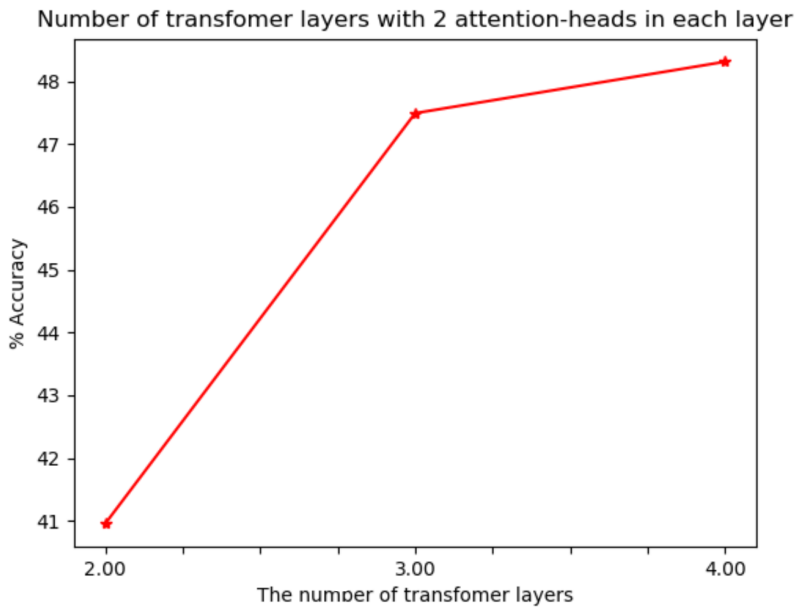


Figure 4.2: Number of layers and corresponding accuracies using 2 attention heads in each layer

As we can see, as the number of layers increases, the accuracy increases. This is expected because the more layers the network has, the more attention can be paid to relevant items, and therefore the next item is predicted more accurately. 4 has been chosen as it is the most optimal number to balance between the computation power needed and the accuracy obtained.

The next experiment is done on the dropout rate to make sure that the network is not overfitting on the training set while not being generalized enough for different test set. Dropout rate is introduced to ensure the training is generalized for any kind of sessions in the future and not only the training set. The dropout rates we have chosen are 5%, 10%, 20% and 30% and the results are shown in Figure 4.3.

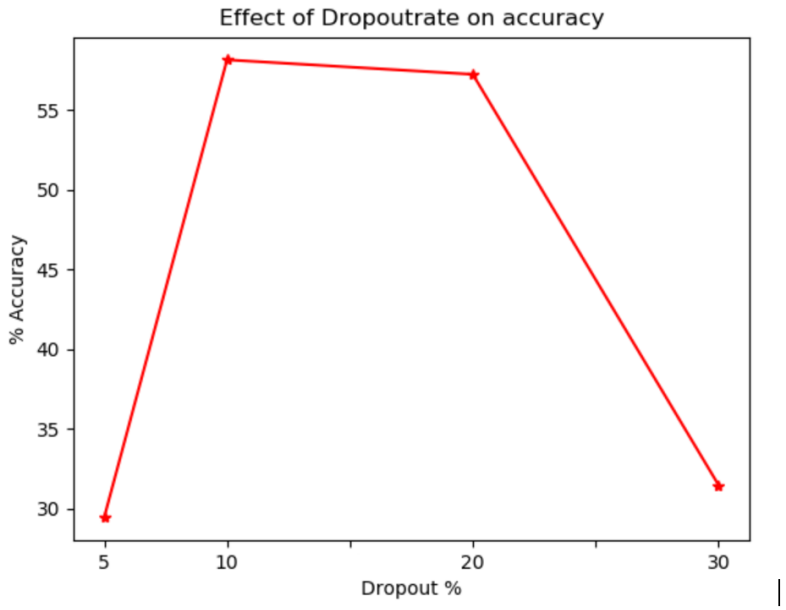


Figure 4.3: The dropout rate after each network and the corresponding accuracies.

As we can see, 5% dropout rate had the least accuracy as the model was overfitting on training data. Therefore, any higher dropout rate led to higher accuracy. 10% and 20% had very similar accuracy while anything more than that (30%) led to a big drop in accuracy as there was essential information not being forwarded between the layers.

The effect of the batch size has been tested to determine the best batch size to be used for training and testing. Batch size is one of a few hyperparameters that can't be determined without testing it. Three different batch sizes were tested, including 64,128 and 256. Anything more than that would require more computation resource, as there are more items being trained at once. The results are shown in Figure 4.4. The batch size of 128 was the worst size while 256 had the highest accuracy.

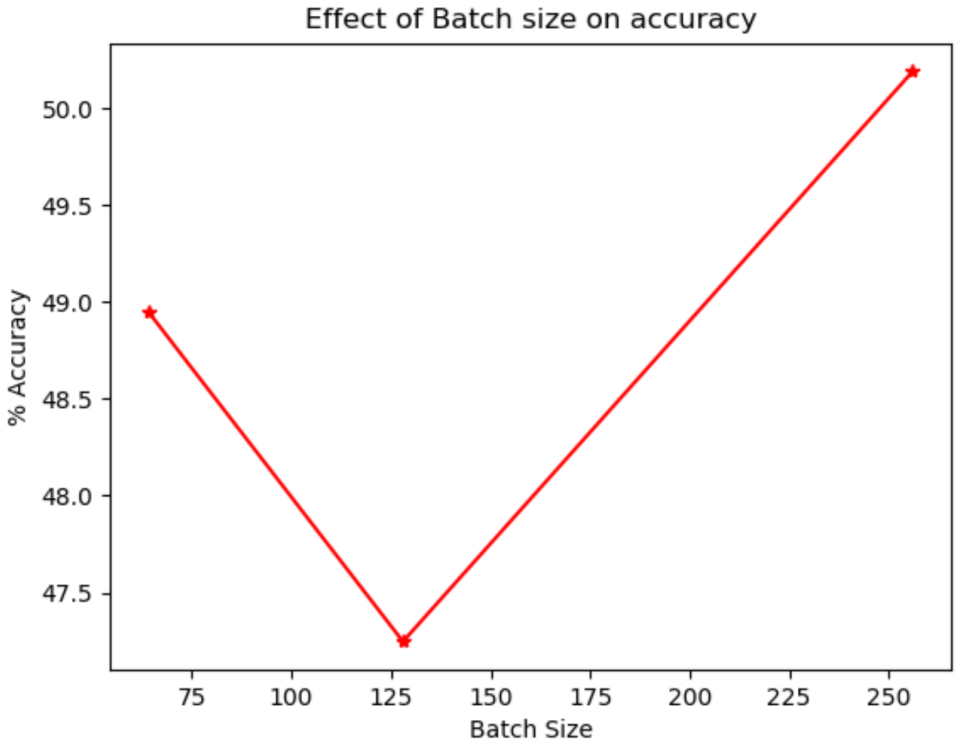


Figure 4.4: Batch size and accuracy

The last experiment is done on the number of epochs. We tried 10, 20, 30 and 40 and the results are shown in Figure 4.5. The training time exponentially grew as the number of epochs increased. Therefore 40 was the maximum number of epochs that was performed. It shows that the more epochs the better the accuracy. We chose number of epochs as 30 instead of 40 in the final model as it takes much less time (the combinations will be shown later) with very small increment in precision.

Now that we have tested the effect of each of the hyperparameters alone with fixating the others, it makes more sense to test combinations of these parameters to determine which combination will lead to the best accuracy for this task.

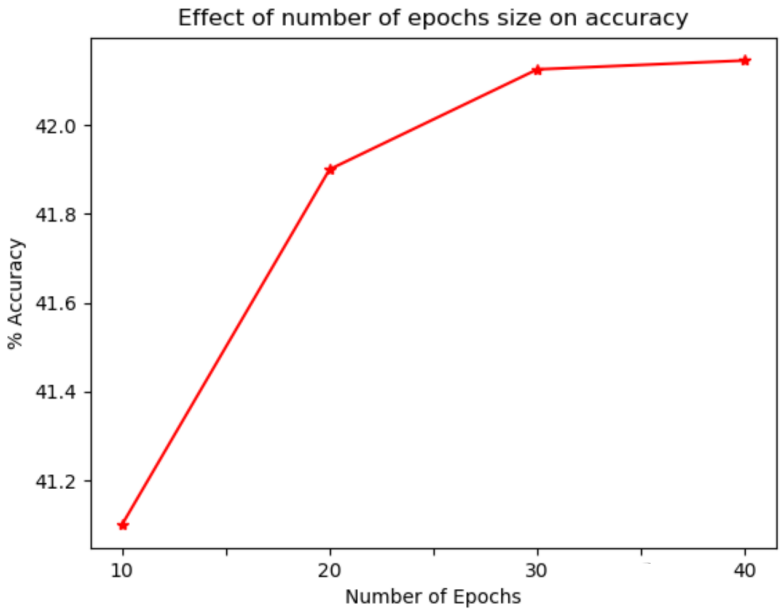


Figure 4.5: Number of epochs and the accuracy

The goal of the first model is to achieve the highest prediction accuracy on masked tokens, where each token has a 10% chance of being masked. The model will utilize the sequence prior to the masked item/token and predict what is that corresponding item. The same session could be seen multiple times by the model but each time with a different masked item thus having an accurate embedding representation for items. This is called cloze learning. All the hyperparameter tuning for this model has been done to achieve highest accuracy in predicting the masked tokens. Table 4.1 shows the combinations used and the accuracies are determined using the precisions at k, where k is 5, 10 and 20 respectively. These metrics were used to compare our work with previous work. For this, two different datasets were tested, and the results are shown below.

Epochs	TRANSFORMER NUMBER	EMBEDDING DIMENSION	Dropout Rate	P@20	P@10	P@5
30	2	20	0.05	65.36	54.52	41.34
30	3	30	0.1	67.14	56.44	42.90
30	3	40	0.1	68.08	57.76	44.30
30	3	50	0.1	68.91	58.55	45.43
30	**4**	**100**	**0.1**	**69.30**	**58.96**	**46.30**
30	4	50	0.1	68.64	58.34	45.08

Table 4.1: all the combinations of hyperparameters and the precision for each of the combination on YooChoose 1/64 dataset

The best combination of hyperparameters were obtained by choosing the one that resulted in the highest P@20. This combination result in the state-of-art for P@20 and P@10 which was held before by STAMP [20]. The results here are obtained on the YooChoose 1/64 dataset. This clearly shows that the bi-directional model produces comparable results to the state-of-art without yet including the deep profiling (second) model.

The same model has been applied on the Dignetica dataset and the results are shown in Table 4.2. Although these two datasets are quite different, the parameters that achieved the best accuracy on the YooChoose 1/64 dataset also produced the best results on the Dignetica dataset. According to these results, we consider these parameter values as the best options for predicting the next item given the current session. This being said, the bi-directional model was not enough to perform the highest accuracies for the Diginetica dataset.

Epochs	TRANSFORMER NUMBER	EMBEDDING DIMENSION	Dropout Rate	P@20	P@10	P@5
30	2	20	0.05	40.95	30.46	21.57
30	3	30	0.1	43.59	32.75	23.40
30	3	40	0.1	46.01	34.72	24.71
30	3	50	0.1	47.49	35.69	25.22
30	**4**	**100**	**0.1**	**48.31**	**35.82**	24.66
30	4	50	0.1	47.51	35.68	**25.39**

Table 4.2: all the combinations of hyperparameters and the precision for each of the combination on Dignetica dataset using bi-directional model

4.2.2 Deep Profiling Model (Second model)

The main usage of the second model is to pass all the current session embeddings along with any history of the users given the items they viewed and the history of their sessions. This stage uses the embeddings obtained from the best model from the bi-directional layer and replaces all the sessions with these embeddings.

In this stage, testing was done as well to ensure the maximum accuracy using the metric P@K. We tested on the effect of batch size, the number of epochs and the dropout rate. The embedding dimension was fixed as it is being obtained from the bi-directional model. Any change in that will give a bias and unfair treatment of items as they will be represented differently in each model. We made sure to have one representation for each item to ensure validity. There is no transformer head in this stage, therefore it was dropped out from test as well for this stage. After finding the best hyperparameters, the learning rate scheduler and the

size of the layers were tested to ensure the maximum accuracy for that model. Below, the result for each test will be explained.

The first test is done on the effect of batch size with all the other variables (listed above) fixed. The batch size taken was 64, 128 and 256. As shown in Figure 4.6, the bigger the batch size, the higher the accuracy.

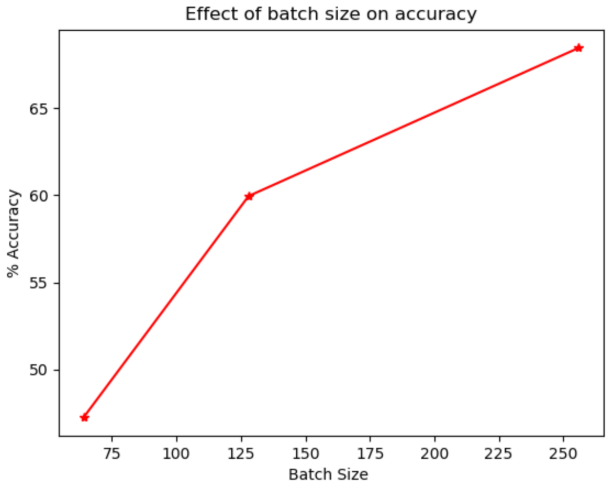


Figure 4.6: The batch size vs Accuracy

The second experiment was done on the number of epochs. In this part, it takes much shorter time for training than in the bi-directional model, therefore it was more viable to test a bigger range of values. In the bi-directional model, a very detailed attention between each item is computed to determine relevance, therefore there is much more computation power and time required for that step. In this experiment we tested the range between 20 to 100 and the results are plotted in Figure 4.7.

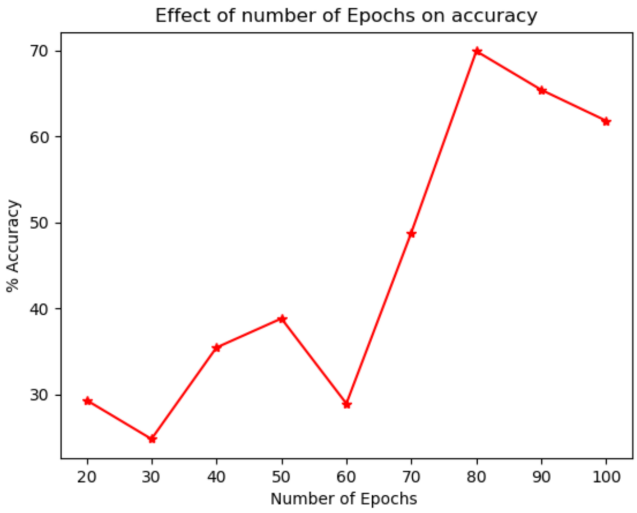


Figure 4.7: The number of epochs vs Accuracy

Next hyperparameter that was experimented was the dropout rate. We tested 3 different dropout rates, including 10%, 20% and 30%.

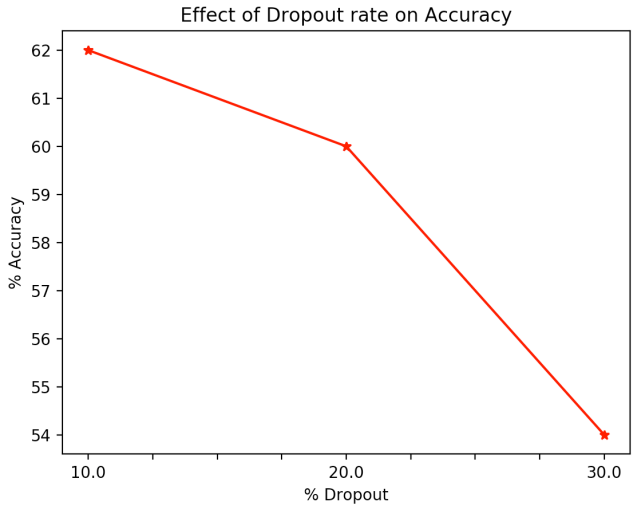


Figure 4.8: Effect of dropout rate on accuracy

After testing each of those hyperparameters individually, different combinations of hyperparameters are tested. Table 4.3 shows the different combinations tested and the

corresponding accuracy. Network size represents the number of nodes in each layer separated by a comma for each layer.

Epoch	BATCH SIZE	Dropout Rate	Network Size	P@20	P@10	P@5
50	128	0.1	{1024,512,256}	30.92	20.54	11.57
50	256	0.2	{1024,512,512,256}	33. 19	29.75	24.40
80	128	0.1	{1024,512,256}	46.00	31.27	26.81
80	256	0.2	{1024,512,512,256}	48.69	37.78	25.87
100	**128**	**0.1**	**{1024,512,256}**	**54.29**	**41.82**	**30.78**
100	256	0.2	{1024,512,512,256}	49.51	38.72	27.24

Table 4.3: all the combinations of hyperparameters and the precision for each of the combination on Dignetica dataset using deep profile model

As mentioned earlier, the YooChoose dataset didn't have any user information. Therefore, the deep profile model wasn't applied as there was no user's history to process. The bi-directional model was sufficient to score the highest precision for this dataset. This being said, the effect of the deep profiling model was proven to be effective as it increased the accuracy on the dignetica dataset from the bi-directional layer by more than 10% for the P@20 and significant increase for the P@10 and P@5. Table 4.4 shows the comparison between SessNet, STAMP, NARM and other baseline models. The SessNet outperforms all the previous models for both datasets and a paper has been written and soon to be submitted. Further experimentation can be performed to check the effect of deep profiling model on data with no user history. Another experiment that can be considered is to run sequence information with user history only using the deep profiling model once and without it once. The difference in the result will determine how effective the deep profiling model is to incorporate the history of sessions and the history of items for each given user.

Models	P@20 for Yoochoose 1/64	P@20 for Diginetica
POP	6.71	0.91
Item-KNN	51.60	28.35
FPMC	45.62	31.55
GRU4REC	60.64	43.82
GRY4REC+	67.84	57.95
NARM	68.32	*62.58*
STAMP	*68.74*	62.03
SessNet	**69.23**	**62.91**

Table 4.4: Indicating the P@20 for baseline models for both State-of-art models STAMP for Yoochoose and NARM for Diginetica along with different baseline models

In both datasets, it is clear that SessNet outperforms the state-of-art models by a good margin while using even less computation power and requirements than NARM and STAMP as stated in their papers [22,20]. It not only produces comparable results but as well performs much better if more user history is stored as it will be utilized by the Deep Profile stage.

Chapter 5 Conclusion

5.1 Summary

SessNet has proven its effect in the next-item recommendation for datasets with and without user history. This is done through a two-stage system that firstly utilizes the sequence of items in a given session and then uses the embeddings to personalize the recommendations for users. SessNet has achieved higher precision the state-of-art models on two different datasets. The first is the YooChoose [1/64] and the second is Dignetica. Yoochoose [1/64] stores only the session data with the timestamps and no user history. Digentica stores the items in a session, timestamp and the users id.

Embedding of those items are constructed using the item co-occurrences information, by taking all the sessions (item sequences), and passing these sessions into the bi-directional model. This model uses transformers and is trained using cloze task. In the bi-directional transformers stage (first stage), random masks are applied to items. These masks hide the items and the model's task is to predict what the hidden items are given the sequence of items before and after the masked item. Each item has 10% chance of being hidden. This structure is similar to the model applied on the original BERT paper [17]. After passing the sessions to the bi-directional model, the embedding of each item is obtained and transferred to the second stage. The bi-directional model has four transformer layers and each has two attention heads in them.

The second stage is the Deep Profile layer. The Deep Profile layer mainly uses the previous sessions that the user has had to ensure that the items recommended are personalized. This is done by taking all the embeddings from the bi-directional model and using them to create different representations that will be fed to a neural architecture to determine the next-item to be recommended. There are three different representations that are fed to the network. The first

is the current session, which is the session where a next-item will be recommended to the user. The second is the user representation, which is determined by replacing all the items the user has previously interacted with their corresponding embeddings. The third representation is the history of all the sessions the user has had. These three representations will have different numbers of items involved in each. To solve this problem, a max pool will be applied to each to ensure a fixed size input being passed to the network. The Deep Profile network has 3 layers of network with 1024, 512 and 256 neurons in each layer respectively. 10% dropout rate is applied between each layer to make sure only relevant information is passed to the next network. The activation function used is ReLU in all the layers' training using sparse categorical cross entropy.

We claim the state-of-art using the SessNet. SessNet had a better score for P@20 for both the YooChoose and Diginetica dataset. This model has two main contributions. Firstly, it outperforms the state-of-art in the next item recommendation task by a wide margin in both datasets than the two models. This implies that the model is much better in determining the best items to be recommended to a user in a specific session by focusing only on relevant items adapting to any interest drift. The second main contribution is that the novel approach of combining a traditional recommender system that analyzes the user's history along with a session-based approach that determines the relevance of an item in a given session. This hybrid is essential in many different domains where recommender systems lacks the accuracy. This type of hybrid systems can be used in e-commerce websites where the users can have different sessions while their history plays important role in what to recommend next.

This model can be further improved by combining the two stages into a single training procedure. The current model optimizes each stage independently. This is basically by obtaining the highest precision in each model separately. After optimizing the bi-directional model, the embeddings are carried to the Deep Profile model and it is optimized to score the

highest precision. Other than co-training the two stages together, more feature engineering can be performed to feed the network more data related to the users and the sessions such as number of sessions, the duration of each session and more.

5.2 Contribution

This model has proven its effectiveness through the results we achieved on two different datasets, each of them with different information. The main contribution of the model lies with the fact that it achieves high accuracies comparable with the state-of-art no matter the user information is present or absent. The first dataset is YooChoose [1/64] which only contains sessions with no any user linkages. For this dataset, only the bi-directional transformer stage is applied since there is no user information for the profiling. The second dataset is the Diginetica where both stages are applied. This shows that this model is very robust to the different dataset and it can always produce comparable results to state-of-the-art.

This model has been the first ever model to use bi-directional transformers in session-based recommendation systems. Bi-directional transformer has been achieving state-of-art for different NLP tasks and it has been used in recommender systems since then. Using such a model has clearly shown promising results.

5.3 Future Work

This model could have a lot of further improvements that could potentially lead to better results. Firstly, the training procedures could be done together rather than separately. The current model optimizes each step individually. Something that could be changed is to have one pipeline that optimizes the backpropagation step together. This will guarantee a higher accuracy. In addition to co-training of the two stages, different model architecture could be

used for the first step to obtain the embeddings in a more efficient way if possible. Lastly the deep profiling step could be modified by using a different RNN network that utilizes the old history of the user's purchases instead of a fully connected neural network.

References

[1] G. Adomavicius, and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. Knowledge and Data Engineering, IEEE Transactions on, 17(6):734-749, June 2005.

[2] P. G. Campos, F. Diez, and I. Cantador. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. User Model. User-Adapt. Interact., 24(1-2):67-119, 2014.

[3] X. Luo, Y. Xia, and Q. Zhu. Incremental collaborative filtering recommender based on regularized matrix factorization. Knowledge-Based Systems, 27(0):271-280,2012.

[4] W. Zhang, J. Wang, B. Chen, and X. Zhao. To personalize or not: A risk management perspective. In Proceedings of the 7th ACM Conference on Recommender Systems, RecSys '13, pages 229-236, New York, NY, USA, 2013. ACM.

[5] C. Aggarwal. Recommender Systems: The Textbook. 1st. Springer Publishing Company, Incorporated, 2016. ISBN: 3319296574, 9783319296579.

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

[7] B. Hidasi. Session-based Recommendations with Recurrent Neural Networks. In: CoRR abs/1511.06939, 2015. URL: http://arxiv.org/ abs/1511.06939.

[8] Y. Zhang et al. “Sequential Click Prediction for Sponsored Search with Recurrent Neural Networks”. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence. AAAI’14. Québec City, Québec, Canada: AAAI Press, pp. 1369–1375, 2014.

[9] S. Wang, L. Cao, and Y. Wang. 2018. A Survey on Session-based Recommender Systems. 1, 1, February 2018, 34 pages. https://doi.org/10.1145/1122445.1122456

[10] Y. LeCun, Y. Bengio, G. Hinton. Deep Learning. Nature volume, 521, pp. 436–444, 28 May 2015

[11] S. Zhang, L. Yao, A. Sun, and Y. Tay. 2018. Deep Learning based Recommender System: A Survey and New Perspectives. ACM Comput. Surv. 1, 1, Article 1, 35 pages, July 2018. DOI: 0000001.0000001

[12] M. Volkovs, H. Rai, Z. Cheng, Ga Wu, Y. Lu, and S. Sanner. Two-stage Model for Automatic Playlist Continuation at Scale. In Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys Challenge '18). ACM, New York, NY, USA, Article 9, 6 pages, 2018. DOI: https://doi.org/10.1145/3267471.3267480

[13] S. Chang, A. Abdul, J. Chen and H. Liao, "A personalized music recommendation system using convolutional neural networks approach," 2018 IEEE International Conference on Applied System Invention (ICASI), Chiba, pp. 47-49, 2018.

[14] A. Abdul, J. Chen & H. Liao & S. Chang. An Emotion-Aware Personalized Music Recommendation System Using a Convolutional Neural Networks Approach. Applied Sciences. 2018. 8.1103.10.3390/app8071103. doi: 10.1109/ICASI.2018.8394293

[15] S. Zhang, Y. Tay, L. Yao, and A. Sun. 2018. Next item recommendation with self-attention. arXiv preprint arXiv:1808.06414, 2018.

[16] A. Vaswani , N. Shazeer , N. Parmar , J. Uszkoreit , Llion Jones , Aidan N. Gomez , Łukasz Kaiser , Illia Polosukhin, Attention is all you need, Proceedings of the 31st International Conference on Neural Information Processing Systems, p.6000-6010, December 04-09, 2017, Long Beach, California, USA

[17] J. Devlin, M. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018

[18] C. Zhou, J. Bai, J. Song, X. Liu, Z. Zhao, X. Chen, and J. Gao, Atrank: An attention-based user behavior modeling framework for recommendation, arXiv preprint arXiv:1711.06632, 2017.

[19] S. Zhang, Y. Tay, L. Yao, A. Sun.: Next item recommendation with self-attention. arXiv preprint arXiv:1808.06414, 2018.

[20] Q. Liu, Y. Zeng, R. Mokhosi, and H. Zhang. STAMP: ShortTerm Attention/Memory Priority Model for Session-based Recommendation. In Proceedings of KDD. ACM, New York, NY, USA, 1831–1839, 2018.

[21] D. Ben-Shimon, A. Tsikinovsky, M. Friedman, B. Shapira, L. Rokach, J. Hoerle. RecSys challenge 2015 and the YOOCHOOSE dataset. In ACM RecSys, 2015.

[22] J. Li, P. Ren, Z. Chen, Z. Ren, and J. Ma. Neural Attentive Session-based Recommendation. In Proceedings of ACM CIKM'17. Singapore, Singapore, 1419–1428, 2017.

[23] D. Jannach. Keynote: Session-Based Recommendation–Challenges and Recent Advances. In Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz). Springer, 3–7, 2018.

[24] D. Jannach and M. Ludewig. Determining characteristics of successful recommendations from log data: a case study. In Proceedings of the Symposium on Applied Computing. ACM, 1643–1648, 2017.

[25] D. Jannach and M. Ludewig. When recurrent neural networks meet the neighborhood for session-based recommendation. In Proceedings of the Eleventh ACM Conference on Recommender Systems. ACM, 306–310, 2017.

[26] D. Jannach, M. Ludewig, and L. Lerche. Session-based item recommendation in e-commerce: on short-term intents, reminders, trends and discounts. User Modeling and User-Adapted Interaction 27, 3-5 (2017), 351–392, 2017.

[27] B. Twardowski. Modelling Contextual Information in Session- Aware Recommender Systems with Neural Networks. In Proceedings of ACM RecSys'16 (September 15 - 19). ACM, Boston, MA, USA, 273–276, 2016.

[28] P. Wang, J. Guo, Y. Lan, J. Xu, S. Wan, and X. Cheng. Learning Hierarchical Representation Model for Next Basket Recommendation. In Proceedings of ACM SIGIR'15. ACM, Santiago, Chile, 403–412, 2015.

[29] F. Yu, Q. Liu, S. Wu, L. Wang, and T. Tan. A Dynamic Recurrent Model for Next Basket Recommendation. In Proceedings of ACM SIGIR'16 (July 17 - 21). ACM, Pisa, Italy, 729–732, 2016.

[30] Y. Zhu, H. Li, Y. Liao, B. Wang, Z. Guan, H. Liu, and D. Cai. What to Do Next: Modeling User Behaviors by Time-LSTM. In Proceedings of IJCAI'17 (August 19 - 25). IJCAI, Melbourne, Australia, 3602–3608, 2017.

[31] M. Quadrana, A. Karatzoglou, B. Hidasi, and P. Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In Proceedings of the Eleventh ACM Conference on Recommender Systems. ACM, 130–137, 2017.

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is All you Need. In Advances in Neural Information Processing Systems 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 5998–6008, 2017. http://papers.nips.cc/paper/7181-attentionis-all-you-need.pdf

[33] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. arXiv preprint arXiv:1906.08237, 2019.

[34] I. Yeo and R. Johnson. A New Family of Power Transformations to Improve Normality or Symmetry. Biometrika 87, 4(2000), 954–959, 2000. http://www.jstor.org/stable/2673623

[35] H. Ying, F. Zhuang, F. Zhang, Y. Liu, G. Xu, X. Xie, H. Xiong, and J. Wu. Sequential recommender system based on hierarchical attention, 2018.

[36] T. Armstrong, A. Mofat, W. Webber, and J. Zobel. Improvements That Don't Add Up: Ad-hoc Retrieval Results Since 1998. In Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09). 601–610, 2009.

[37] G. Moreira, F. Ferreira, and A. Marques. News Session-Based Recommendations using Deep Neural Networks. In Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems (DLRS) '18, 2018.

[38] H. Fu, J. Li, J. Chen, Y. Tang, and J. Zhu. Sequence-Based Recommendation with Bidirectional LSTM Network. In Advances in Multimedia Information Processing (PCM '18). 428–438, 2018.

[39] M. Quadrana, P. Cremonesi, and D. Jannach. SequenceAware Recommender Systems. Comput. Surveys 51, Issue 4, 1–36, 2018.

[40] B. Hidasi and A. Karatzoglou. Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM '18). 843–852, 2018.

[41] S. Rendle, C. Freudenthaler, and L. Thieme. Factorizing personalized Markov chains for next-basket recommendation. In Proceedings of WWW'10. ACM, Raleigh, North Carolina, USA, 811–820, 2010.