2019

Explainable Neural Attention Recommender Systems

Omer Tal
talx6630@mylaurier.ca

Explainable Neural Attention Recommender Systems

By:

Omer Tal

Master of Applied Computing, Wilfrid Laurier University, 2018

THESIS

Submitted to the Department of Physics and Computer Science

Faculty of Science

in partial fulfillment of the requirements for the

Master of Applied Computing

Wilfrid Laurier University

Abstract

Recommender systems, predictive models that provide lists of personalized suggestions, have become increasingly popular in many web-based businesses. By presenting potential items that may interest a user, these systems are able to better monetize and improve users' satisfaction. In recent years, the most successful approaches rely on capturing what best define users and items in the form of latent vectors, a numeric representation that assumes all instances can be described by their respective affiliation towards a set of hidden features.

However, recommendation methods based on latent features still face some real-world limitations. The data sparsity problem originates from the unprecedented variety of available items, making generated suggestions irrelevant to many users. Furthermore, many systems have been recently expected to accompany their suggestions with corresponding reasoning. Users who receive unjustified recommendations they do not agree with are susceptible to stop using the system or ignore its suggestions.

In this work we investigate the current trends in the field of recommender systems and focus on two rising areas, deep recommendation and explainable recommender systems. First we present *Textual and Contextual Embedding-based Neural Recommender* (TCENR), a model that mitigates the data sparsity problem in the area of point-of-interest (POI) recommendation. This method employs different types of deep neural networks to learn varied perspectives of the same user-location interaction, using textual reviews, geographical data and social networks.

We then suggest two novel, explainable, frameworks. *Dual Attention Recommender with Items and Attributes* (DARIA) is based on the emerging neural attention technique, where latent representations are effected by the importance of items' features and users' past preferences. *Self-Attention Recommender based on Attributes and History* (SARAH) further adopts self-attention, an extension to the standard neural attention paradigm. By utilizing this latest concept, SARAH is able to represent users and items by their most relevant input features.

A series of experiments demonstrate that both DARIA and SARAH consistently outperform state-of-the-art baselines in diverse recommendation scenarios, while being able to justify their suggestions. Furthermore, we analyze the effects of different hyper-parameters and design selections, shedding light on the impact attention networks have in the area of recommender systems.

Acknowledgments

I would like to express my very great appreciation to Dr. Yang Liu for her support and guidance throughout the research work as my advisor. Her broad knowledge, vast experience and valuable feedback allowed me to pursue the topics I am most interested in, and to greatly expand my professional horizons. I would further wish to thank our collaborators, Dr. Jimmy Huang and Dr. Xiaohui Yu from York University for their helpful comments.

In addition, I like to express my gratitude to Dr. Chính T. Hoàng and Dr. Hongbing Fan for their work in creating the Master of Applied Computing program that provided me a great learning opportunity. Finally, my family and friends deserve a special thanks for their encouragement and moral support throughout this period.

Various chapters in this thesis include material that has been submitted for publications as it may appear in 2018 ICDM Workshops, 2019 Complexity journal special edition and SIGIR 2019. The thesis author was the primary author of these papers.

Contents

List of Tables **VII**

List of Figures **VIII**

1 Introduction **1**

1.1 Problem Definition 1

1.2 Major Challenges 2

1.3 Our Method and Contribution 3

2 Literature Review and Background **5**

2.1 Collaborative Filtering 6

2.2 Deep Recommender Systems 8

2.2.1 Multi Layer Perceptrons 9

2.2.2 Auto Encoders 10

2.2.3 Recurrent Neural Networks 11

2.2.4 Convolutional Neural Networks 13

2.3 Point-Of-Interest Recommendation 15

2.3.1 Challenges in POI Recommendation 15

2.3.2 Solutions and Limitations 16

2.4 Neural Attention 18

2.5 Explainable Recommendations 21

3 Deep POI Recommendation 24

3.1 Recommendations In LBSNs 24

3.2 Textual And Contextual Neural Recommender 26

3.2.1 Context-based network layers 27

3.2.2 Textual modeling network layers 29

3.3 Training the Network 32

3.4 Textual Modeling Using Word Sequences 34

3.5 Model Evaluation 36

3.5.1 Experimental Setup 36

3.5.2 Baselines 37

3.5.3 Performance Evaluation 39

3.5.4 Model Design Analysis 41

4 Explainable Neural Attention Frameworks 45

4.1 Motivation 46

4.2 Self-Attention With Attributes And History 48

4.2.1 Item Self-Attention 51

4.2.2 User Self-Attention 53

4.2.3 Rating Prediction 54

4.3 Dual Attention Recommender 55

4.3.1 Past Items Importance 58

4.3.2 Item Features Attention 59

4.3.3 Comparison to SARAH 61

4.4 EXPERIMENTAL SETTINGS 62

4.4.1 Datasets 62

4.4.2 Evaluation Metrics 63

4.4.3 Baselines 64

4.4.4 Parameters Settings 66

4.5 EXPERIMENTS AND EVALUATION 66

4.5.1 Overall Performance (RQ1) 67

4.5.2 Hyperparameter Analysis (RQ2) 69

4.5.3 Case Studies (RQ3) 75

4.5.4 Comparison To TCENR (RQ4) 78

5 Conclusion and Future Work **80**

Bibliography **82**

List of Tables

3.5.1 Performance Comparison for TCENR 39

3.5.2 TCENR Accuracy with Different Layer Designs 43

4.1.1 Parameter notations used in Chapter 4 48

4.4.1 Datasets' Statistics for Attention Experiments 63

4.5.1 Performance Comparison for Attention Models 67

4.5.2 Performance Comparison between SARAH, DARIA and TCENR . . 79

List of Figures

3.2.1 TCENR Framework Architracture 27

3.4.1 Textual Modeling Alternatives In TCENR 35

3.5.1 Runtime Analysis for TCENR 41

3.5.2 TCENR Merging Layer Comparison 43

3.5.3 Number Of Words Impact In TCENR 44

4.2.1 SARAH Framework 49

4.2.2 SARAH Sample Scenario 50

4.3.1 DARIA Framework 57

4.5.1 Runtime Analysis for SARAH 69

4.5.2 Number Of Features Analysis in SARAH 71

4.5.3 Number Of Past Items Comparison in SARAH 73

4.5.4 Item Embedding Size Evaluation in SARAH 74

4.5.5 RNN Layer Units in SARAH 75

4.5.6 Item Attention Scores Case Study 76

4.5.7 User Attention Scores for a Positive Sample 77

4.5.8 User Attention Scores for a Negative Sample 78

Chapter 1

Introduction

1.1 Problem Definition

In today's age of information, it has become a prerequisite to have reliable data prior to making any decision. Preferably, we expect to obtain opinions from like-minded users before investing our time and money in any product or service. However, with increasing variety of available items it is extremely tiresome for customers to rely on methods such as browsing or searching in order to find these opinions. Such is the case for e-commerce websites as eBay with 1.1 Billion listings[1] or location-based social networks as Yelp with 171 million reviews[2].

Personalized *recommender systems* (RS) are therefore a vital component in many on-line businesses, websites, social networks and media services. They consist of proposing a tailored list of relevant items, such as movies, songs, products, locations and more, to an end user, allowing her to make the best selection out of an almost endless list of possibilities. More specifically, a recommender system is a method of supervised learning, that usually takes tuples of users and items as input and predicts the probability a user will be interested in the given item. It is trained and tested

[1] https://www.ebayinc.com/our-company/who-we-are/

[2] https://www.yelp.ca/factsheet

by having ratings given by the users (e.g. 5 stars scale, 1-10 numeric grade, binary score, etc) alongside the reviewed item and user data.

1.2 Major Challenges

A prominent technique partly responsible for the rising success of recommender system is *collaborative filtering* (CF). An item-based CF model attempts to identify similar items to those previously liked by the user. User-based methods, on the other hand, assume that users who shared resembling interests in the past are likely to do so in the future. The availability of many distinct items, however, results in few users who share the exact same past experiences. It is more likely that similar users interact with resembling items (e.g. watched an action movie created by the same director). Latent-based collaborative filtering, therefore, is a recommendation paradigm that attempts to represent users and items in a way that captures their preferences and attributes, respectively, in a way that is both accurate and allows generalization.

Latent CF-based methods, however, have some drawbacks making their implementation challenging in many scenarios. First is data sparsity, usually referred to as the cold-start problem, due to the missing data for new users and items. This issue however is not limited to new components in the system. As the range of available items and the size of systems' user base increases, less similar they become. The more varied past activities are, the fewer users and items have sufficient data to result in meaningful representations. In addition, methods based on latent-features are commonly perceived as a ”black-box” that is only responsible for outputting recommendations. In case an end user does not agree with the given suggestion, she is unable to learn the reasoning behind it and her motivation to use the system decreases.

The rise of deep learning has had a broad impact over recommender systems in general, and specifically over latent-based CF methods. The ability to represent users and items using vectors allows deep neural networks to be employed by stacking multiple layers, in order to learn the user-item interactions [20, 65] or to learn the representations themselves [15, 60]. Along with the explosion of available data, these methods achieved a great success in previous years. Different deep neural network architectures have been introduced to improve the recommendation performance, such as *multi-layer perceptrons* (MLP) [12,32], *convolutional neural networks* (CNN) [25, 71] and *recurrent neural networks* (RNN) [1, 3]. However, by applying multiple nonlinearities over its inputs, recommendation techniques based on deep learning are usually considered extremely prone to the explain-ability problem.

1.3 Our Method and Contribution

In this work we start from reviewing recent developments in deep recommender systems, and then introduce multiple frameworks that attempt to mitigate the aforementioned challenges. To tackle the data sparsity and cold-start problems we will first focus on the area of *points-of-interest* (POI) recommendation. In this field, locations are suggested to users of *location-based social networks* (LBSN), such as Yelp and TripAdvisor. We present *Textual and Contextual Embedding-based Neural Recommender* (TCENR), a framework that takes users' social network, locations' geographical data and textual reviews along with historical activities, to produce personalized POI recommendations. While most works focus on only a single type of deep neural network, TCENR combines multiple techniques, i.e., MLP, CNN and RNN, to provide suggestions using various types of inputs. By successfully integrating relevant paradigms, TCENR is shown to outperform multiple key baselines over the *Yelp* dataset for restaurant recommendations in terms of accuracy, MSE, precision and recall.

We further adopt the novel technique of *neural attention networks*, where dedicated layers are responsible for identifying the most important components of their inputs [2, 43], and propose two frameworks meant to produce relevant explanations along with accurate suggestions: i) *Dual Attention Recommender with Items and Attributes* (DARIA) utilizes two neural attention mechanisms, where each user is represented by her most relevant past items with regard to the recommended item, while a second attention layer determines which attributes best contributed to the reported items' similarity. ii) *Self-Attention Recommender based on Attributes and History* (SARAH) attempts to solve the same task as DARIA by employing the self--attention paradigm, a recent extension to neural attention. Based on identifying the most important components depicting an input, self-attention is applied to construct items' and users' latent vectors. The use of self-attention allows SARAH to determine which historical activities are most relevant to represent each user, independently of the candidate item, along with identifying the input features that best capture a suggested item.

Our proposed methods are evaluated in various settings by utilizing four datasets and demonstrate significant improvement over six varied baselines, including methods of classic CF [36, 58], deep learning [20, 25] and neural attention [11, 16], using the popular metrics hit ratio and normalized discounted cumulative gain.

This thesis is organized as follows. In Chapter 2 we review notable works in deep recommender systems history, and describe recent developments related to the suggested methods. Chapter 3 extends the discussion on POI recommendation and presents TCENR along with its sequential variation. In Chapter 4 we focus on explainable recommender systems and introduce our two novel attention-based approaches, SARAH and DARIA. We further perform extensive empirical evaluation on our proposed methods and provide case studies to demonstrate their explain-ability. Chapter 5 concludes this work along with directions to future research. In the following chapters, we denote matrices in upper-case letters, while vectors and scalars are presented as lower-case.

Chapter 2

Literature Review and Background

In this chapter we discuss relevant background and present notable advancements in the area of recommender systems.

Recommender system is a sub-field of machine learning, where a model is developed to provide item suggestions for a given user. As a method of supervised learning, recommender systems are given past user-item interactions to learn from, and based on the type of target feature can be classified as having either *explicit* [25, 67, 71] or *implicit* feedback [3, 64, 69]. Algorithms that take input ratings based on a numeric scale, such as 1-5 stars or 1-10 score, are considered explicit, since this preference was directly given by the user as part of a rating or review. A distinction between liked and disliked samples can be then determined according to a predefined range (e.g. items with an estimated scores of 4-5 stars should be recommended).

On the other hand, methods based on implicit data adopt input features collected as part of a user's usage, such as clicking on an item or making a purchase. While implicit-based methods usually have more available data, they are only aware of items the user interacted with and lack indication towards the disliked or negative instances. Recommender systems can be further classified by their type of output. Models that estimate a score for a given user-item tuple are defined as *point-wise ranking* [12,20,62] while methods that attempt to rank relevant items over irrelevant ones are denoted as *pair-wise ranking* [35, 53, 70]. Nonetheless, the goal for most recommender systems is to generate a list of personalized recommendations with as many relevant items.

An important factor in classifying different recommender systems is the type of architecture, where the two most popular categories are *content-based* and *collaborative filtering* (CF) [47–49],. While a content-based system relies on the similarity of items using their attributes, in this work we will focus on its more popular alternative.

2.1 Collaborative Filtering

CF-based recommender systems have become extremely common [22, 56], due to their ability to outperform competitive methods while being efficient and easy to implement. They are based on the assumption that users who shared similar preferences in the past will continue to do so in the future. Collaborative filtering approaches can be further divided to neighborhood-based and model-based methods [28]. Neighborhood-based recommender systems explicitly compute the similarities between different users. In prediction time, they produce an estimation based only on the likings of users that most resemble the target user. In contrast, the model-based approach constructs latent representations of users and items using a machine learning algorithm. These latent descriptions are used to provide a concise representation for users' preferences and items' attributes.

A well-known implementation of the model-based approach is the *Matrix Factorization* (MF). This method gained much popularity due to its simplicity and success in the famous Netflix challenge [29]. Using MF, the predicted rating for a given user u and item i can be defined as:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T \cdot p_u \ , \tag{2.1.1}$$

where $\cdot$ is the dot-product operation, μ is the mean rating, q_i and p_u are item i's and user u's latent vectors, respectively, while b_i and b_u are their bias factors. These five parameters are learned using the following point-wise loss function:

$$\min_{b_*,q_*,p_*} \sum_{u,i \in Y} \left(r_{u,i} - \hat{r}_{u,i}\right)^2 \ , \tag{2.1.2}$$

where Y is the test set, $r_{u,i}$ is the ground truth and the model is penalized for differences between it and the predicted score $\hat{r}_{u,i}$. Alternatively, the same model could be optimized by employing pair-wise ranking:

$$\min_{b_*,q_*,p_*} \sum_{u} \sum_{i \in Y} \sum_{j \in Y^-} \log \alpha\left(\hat{r}_{u,i} - \hat{r}_{u,j}\right)^2 \ , \tag{2.1.3}$$

where $j \in Y^-$ is a negative item and therefore should be ranked lower than the positive item $i \in Y$. The advantage of pair-wise ranking is that we are only required to know what items are preferred by the user compared to others, rather than an explicit ground truth. However, unlike models based on point-wise ranking, following the pair-wise technique does not allow a recommender system to generate a score for a single given item. This can limit the ability to properly evaluate the model. Either way, the two alternatives can be optimized similarly using a method based on *gradient descent*.

While the inclusion of bias vectors allows methods such as MF to consider variations between estimated latent representations and real life scenarios, probabilistic methods [1,13,58] further extend the assumption of uncertainty in the learned model. Probabilistic matrix factorization [42] assumes the user and item latent vectors can be drawn from a normal distribution, as well as their estimated rating:

$$\begin{aligned} p_u &\sim \mathcal{N}(0, \lambda_u^{-1} I_K) \ , \\ q_i &\sim \mathcal{N}(0, \lambda_v^{-1} I_K) \ , \\ \hat{r}_{u,i} &= \mathcal{N}(q_i^T p_u, c_{u,i}^{-1}) \ , \end{aligned} \tag{2.1.4}$$

where I is the identity matrix and c is a confidence parameter set to distinguish between different rating classes according to the certainty we have towards them.

Although methods based solely on user-item interactions achieve relative success, some recommendation scenarios can be too complex for such an approach. In the case of a sparse dataset, an unbalanced distribution of items per user or simply lack of scores, ratings can be insufficient in capturing an accurate representations for users and items. Additional features such as time [38], spatial location [10], users' demographic data [12] or items' meta-data [32] provide additional insight towards the factors that contribute to a user's interest or define an item. [58] extends the probabilistic MF method and samples an item latent vector where the mean is derived from its respective topics distribution. The topics in turn, are learned from textual inputs (e.g. reviews or description) using the popular *latent dirichlet allocation* (LDA) model [6].

2.2 Deep Recommender Systems

Deep learning is a sub-field of machine learning where latent input representations are developed using multiple layers of non-linearities. Due to the vast availability of data and computational resources, methods based on deep learning paradigms have become increasingly popular in recent years. The use of latent vectors to represent users and items proved recommender systems to be a highly relevant field to adopt deep learning techniques.

2.2.1 Multi Layer Perceptrons

Probably the most general approach to implement a deep recommender system is based on multi-layer perceptrons (MLP), where a neural network with multiple hidden layers is employed. In [12], a "wide" component, a simple linear transformation on various input features, is combined with a "deep" component that learns a concise representation of a given user-item interaction. The model is fed with tuples of N users and M items, each in the form of a *one-hot encoding.* Such encoding is a transformation of the categorical value to a vector in the size of N or M, where 1 indicates the given user or item, respectively, while 0 denotes the rest. The deep component then transform each input to a latent vector using a lookup function denoted as its *embedding*, e_u for the user and e_i for the given item. A combined vector with all the user and item latent factors is formed by concatenating the two embeddings. To estimate whether u is interested in i, the model learns the interactions between different factors:

$$\begin{aligned} H_0 &= [e_u, e_i] \ , \\ H_1 &= a_1(W_1 \times H_0 + b_1) \ , \\ &\vdots \\ H_l &= a_l(W_l \times H_{l-1} + b_l) \ , \\ &\vdots \\ H_L &= W_L \times H_{L-1} + b_L \ , \\ \hat{r}_{u,i} &= \sigma(H_L) \ , \end{aligned} \tag{2.2.1}$$

where [ ] is the symbol for concatenation, while W_* and b_* are the weight and bias terms, respectively, optimized by the model as parameters. a_* is a nonlinear activation function, which is commonly either the *sigmoid*, *hyperbolic tangent* (tanh) or *the rectified linear unit* (ReLU):

$$\sigma = \frac{1}{1+e^{-x}} \ . \tag{2.2.2}$$

$$tanh = \frac{sinh_x}{cosh_x} \ . \tag{2.2.3}$$

$$ReLU = max(0, x) \ . \tag{2.2.4}$$

By setting W_L in Eq. 2.2.1 to be a one-dimensional vector and adopting the sigmoid function, the MLP last hidden layer transforms its concise input to a predicted score in the range of $[0, 1]$. In a similar fashion, [20] utilizes MLP to learn a concise representation of the given user-item interaction. The authors further concatenate the resulting vector with the dot product of the two embedding, before feeding it into another neural network with one layer. By doing so, they are able to combine the strengths of the deep MLP and vanilla MF. A different adoption of multi-layer perceptrons is demonstrated in [15]. By applying multiple ReLU layers over the concatenated embeddings of various user attributes, a user latent vector is learned. A *softmax* function transforms the user representation to a distribution over the number of items to determine the most probable recommendations.

2.2.2 Auto Encoders

A common alternative to MLP for the task of learning user or item representations is the use of *auto-encoder* (AE). This technique attempts to reconstruct the input using hidden layers in a neural network, while the input and output layer are of the same structure. By penalizing the network for differences between its output and input, the more concise hidden layer is optimized to store an accurate representation of its input. A denoising auto-encoder assumes the input is noisy or corrupt, and its goal is to learn a clean version of it using the AE framework. Due to the partial availability of data and the method's robustness, denoising auto-encoder have found relative success in learning user or item embeddings in recommender systems. [59] extend the probabilistic method developed in [58], by stacking multiple hidden layers within the denoising AE. The model's first half encodes the item textual attributes while the remaining layers are responsible do decode it and reconstruct the item. The middle layer is therefore the most concise item representation. While [59] adopts a probabilistic approach, [33] utilizes a similar architecture to learn both the user and item latent vectors simultaneously.

Some methods however apply auto-encoders to learn the full user-item interaction. [64] first modifies its inputs to introduce real-world noise to the system before feeding the corrupted items and user vector to an AE, responsible to reconstruct the original input. By combining both user and item data in the auto-encoder, the method is able to regard the output layer as the list of recommended items.

2.2.3 Recurrent Neural Networks

While MLP and AE based methods were proven to be successful in learning latent representations from past activities, they are not optimized to capture dependencies between different inputs. Moreover, treating user purchases independently might not be suitable in all scenarios and may lead to data loss. For example, a system able to recognize that users usually go to the cinema right after visiting a restaurant might provide more relevant suggestions following a user interest in a movie. Recurrent neural networks (RNN) however, are based on a specialized architecture capable of capturing such sequentiality by including an internal hidden state in each input and considering the impact of its predecessors.

The *long-short term memory* (LSTM) framework is employed in [63] to represent each user and item by their latest activity sequence. Assuming inputs are fed to an LSTM model as $\{z_0, z_1, ..., z_{t-1}, z_t, z_{t+1}, ..., z_T\}$, a time state t will be updated by the following operations:

$$\begin{aligned} f_t &= \sigma[W_f[h_{t-1}, z_t] + b_f] \ , \\ i_t &= \sigma[W_i[h_{t-1}, z_t] + b_i] \ , \\ o_t &= \sigma[W_o[h_{t-1}, z_t] + b_o] \ , \\ l_t &= tanh[W_l[h_{t-1}, z_t] + b_l] \ , \\ \tilde{c}_t &= f_t \cdot \tilde{c}_{t-1} + i_t \cdot l_t \ , \\ h_t &= o_t \cdot tanh(\tilde{c}_t) \ , \end{aligned} \tag{2.2.5}$$

where W_* and b_* are the weight and bias parameters, σ the Sigmoid activation and f_t the forget gate that indicates to which degree should previous inputs be integrated into the current input, z_t. i_t is the input gate that determines the current input's impact and o_t the output gate which effects the current hidden state, h_t, signal strength. Finally, $\tilde{c}_t$ represents the current candidate state, before applying the output gate. Employing these multiple gates allow inputs in different states to have a respective weight in the representations of successive inputs. In [63] LSTM units are used to learn user and item representations, where the latent vectors are the final states, h_T, allowing important past interactions to be captured while potentially giving more weight to recent activities.

While LSTM is highly capable in capturing the most significant states of an input, it is extremely inefficient due to its reliance on a large number of learned parameters. A more common alternative is the *gated reccurent units* (GRU) [14]:

$$\begin{aligned} \tilde{c}_t &= tanh(W_z z_t + f_t \cdot W_h h_{t-1} + b_{\tilde{c}}) \ , \\ h_t &= (1 - o_t) \cdot h_{t-1} + o_t \cdot \tilde{c}_t \ , \end{aligned} \tag{2.2.6}$$

where f_t and o_t are the same formulas as in Eq. 2.2.5. GRU ignores the input gate and its parameters, and therefore allows faster convergence.

Following its original success in tasks of textual modeling [41], recurrent neural network are a popular solution to recommender systems aiming to utilize available textual data. Since words are sequential by nature, treating each word as an input state allows the RNN to capture semantics found in the original text, whether it is an item description or a user written review. In some cases, however, the meaning of a word can only be realized by other words following it rather than preceding it. [3] adopts the bi-directional GRU where one layer learns to represent each word in an item description by maintaining the context from previous words to its left, while a second GRU layer process the same text from right to left. Concatenating the two representations for each word allows the network to fully capture the semantic meaning found in the text. The model exploits the resulting word vectors to learn an item representation derived solely from text, before enriching it with additional attributes and user data.

2.2.4 Convolutional Neural Networks

Originating from tasks of image processing [30, 31], a convolutional neural networks (CNN) treats its input as pixels and attempts to identify the most relevant sliding windows over it. While images are not a common input feature in recommender systems, integrating the CNN framework is considered a common alternative to RNN in textual modeling tasks [26]. First, a convolution operation is applied on a sliding window of concatenated words to generate a feature map:

$$\begin{aligned} D_{1:n} &= [w_1, w_2, ..., w_l, ...w_n] \ , \\ z_l^j &= a(W^j * D_{l:l+ws-1} + b^j) \ , \\ z^j &= [z_1^j, z_2^j, ..., z_l^j, ..., z_n^j] \ , \end{aligned} \tag{2.2.7}$$

where w_l is the latent embedding of a word in position l, $*$ the convolution operation, ws the window size hyper-parameter, z_l^j a contextual feature, W^j the filter and b^j the bias.

Eq. 2.2.7 results in z^j which represents a single feature vector, determined by the filter W^j. To extract multiple features from the given text, more filters can be added so $Z = [z^1, ..., z^j, ..., z^m]$. Since CNN usually takes the full text as input, it can identify many signals as part of each feature vector, where some can be redundant or even contradicting. Therefore it further requires a *max-pooling* operation, that enables it to keep only values with the highest scores:

$$o_f = [\max(z^1), \max(z^2), ..., max(z^j), ..., max(z^m)] \ . \tag{2.2.8}$$

To fit the high level textual representation to the recommendation task, [25] feed o_f, learned from an item description, to a two layer neural network, activated by the *tanh* function. The resulting vector is employed as the item latent representation integrated into the probabilistic MF approach. In [71], two CNN layers are integrated to learn both the user and item representations jointly. All words written in user u's reviews are concatenated and kept in their original order, as well as all words written about item i. The two word vectors are fed each to separate CNNs, following the operations in Eq. 2.2.7 and Eq. 2.2.8. After applying the hidden layers to each vector, the user and item representations are combined in a shared layer to generate a point-wise prediction.

Although [71] is able to produce relevant latent representations from text, it introduces a bias by feeding all words written by the user and about the item, including those that describe the trained object. In other words, the network is exposed to data that is only available in training, resulting inferior performance in test time. [7] proposes to separate the words describing each trained interaction and learn two networks simultaneously. A source network is based on [71] without the current review, while the target network utilizes standard CNN over the specific review. By penalizing the difference between the learned representation in the source network and the vector produced by the target, the model learns to predict the review written by the user on the candidate item and therefore its respective rating.

In our proposed methods we intend to rely on multiple deep learning techniques to fit each recommendation scenario and data type with the according paradigm. We will utilize multi-layer perceptrons to model past user-item interactions, convolutional neural networks to represent textual reviews and recurrent neural networks to capture the sequentiality of both words and user activities.

2.3 Point-Of-Interest Recommendation

Locations-based social networks (LBSN), such as Yelp, TripAdvisor and Foursquare are environments that allow users to share experiences about the places they visit. Point-of-interest (POI) recommendation, a sub-field of RS, attempts to provide LBSN users with personalized suggestions. Properly exploited, it can save time and effort for the end user, and encourage her to make future use in the location-based social network both as a consumer and content provider.

2.3.1 Challenges in POI Recommendation

Similar to standard recommender system, POI recommendation takes tuples of users and items as input and estimates their rating. There are, however, some challenges that exist to a higher degree in this scenario, making it a dedicated research field. While data sparsity is a recurring issue in all RS, LBSN users are unable to physically visit most locations due to geographical distance, an issue that is even worsened for out-of-town users [61]. Second, in contrast to other recommendation scenarios, the decision whether to visit a location depends not only on the target user's preferences, but on the that of her friends [32]. They might share different interests that are unknown to the system, resulting in a more complex decision process for the RS to learn. Finally, most location-based social networks rely on implicit feedback, derived from geo-spatial coordinates and defined as check-in. It allows the POI recommender to learn a user has visited a location but not the extent to which she liked it. The lack of explicit data requires the RS to identify what locations should be regarded as the negative instances for each user.

2.3.2 Solutions and Limitations

The availability of contextual data within LBSNs provides an opportunity to utilize features such as social networks [37], geo-spatial locations [10] and time [38] to mitigate the data sparsity issue and to gain insights towards users' interests and locations' attributes. These contextual features are usually incorporated into RS either as part of the input or as a regularizing factor. Following previous RS techniques, MLP-based networks are a popular choice in modeling contextual data by concatenating respective embeddings before stacking the non-linear layers [15], while sequential data is often introduced by utilizing recurrent neural networks [5].

The use of spatial data is often done by dividing the input space into roles and regions. Assuming users' behavior varies when traveling far from home, previous works [61, 66] generated two profiles for each user, one to be used in her home region while another in more distant locations. A recent approach [67] attempts to divide the input space into geographical regions before incorporated into the model, often by hierarchical structures. Although methods based on regions and roles are able to better distinguish user behaviors in varied locations, they do not provide a personalized user representation and can ignore potential shifts of preferences from one region to another. For example, a user might prefer to visit a Starbucks location in different regions, close or far from home. Enriching geographical features with additional data is demonstrated in [68] where the next location prediction is partially determined by past sequences of check-ins. However, these methods are not generic and cannot be extended to include additional features, derived from social networks or textual data.

Furthermore, in case of tasks in highly sparse environments, such as POI recommendation, adding user or item specific inputs may diminish the model's ability to generalize. However, applying the same data as a regulating factor can enhance the model's performance and reduce over-fitting. Such has been done in [37], where the similarity between connected users in the social network was used to constrain a matrix factorization model. [65] utilized social networks and geographical distances to enforce similar embeddings for users and locations in an MLP, thus improving the model's ability to generalize for users and locations with few historical records.

Since many websites encourage users to provide a written explanation to their numeric ratings, textual reviews are one of the most popular types of data to be integrated into RS. By expressing each review as a bag of words, LDA-based models are able to extract topics which can be used to represent users' interests and locations' characteristics [66]. These probabilistic methods are usually successful in handling issues that standard CF approaches struggle with, such as out-of-town recommendations where similar users lack sufficient historical data. However, as demonstrated in recent works [52], failing to preserve the original order of words and ignoring their semantic meaning prevents the successful modeling of a given review. On the other hand, adopting deep methods such as RNN [1] and CNN [25] over reviews allows such learning without the loss of data.

In this work we claim that by jointly learning contextual and textual based deep models a POI recommender system can better exploit the strengths of collaborative filtering, while being more resilient to its shortcomings in sparse scenarios. This will be achieved by proposing TCENR, which learns users' and locations' representations as similarities in direct interactions, along with the correlation in underlying features extracted from their written reviews. The notable work of [70] proposed JRL, a framework that similarly attempts to jointly optimize multiple models, where each is responsible for learning a unique perspective of the same task by focusing on different inputs. However, while JRL is a general framework, focused on extendability to many types of input, we propose a dedicated framework for the task of POI recommendation.

2.4 Neural Attention

Defined in the field of neural science [23], attention describes the ability to focus on what perceived as the important part of an input. By integrating attention in neural networks, this concept has seen rising success in computational tasks such as image processing [43], textual translation [2], summation [45] and additional language modeling tasks [50].

Recent works have been introducing attention-based neural networks to recommender systems. In [18], the authors computed attention weights to identify the most important words in an input micro-blog within a CNN. [54] reasoned that users are interested in various features of the same product, represented as different keywords in its reviews. The authors therefore employed attention to identify the most important reviews and words written on a candidate item, based on their relevance to the user. This was achieved by first representing each user and item as three-dimensional matrices, denoted as a_i and b_j, respectively, that hold the word embeddings for each review. Important reviews were extracted by calculating an affinity matrix between a_i and b_j and taking only the maximum row to represent the most important review written by the user and column for the item. This process was done again over the selected reviews' word embeddings to identify the most important words.

The use of attention networks is not limited to textual modeling, though. In [11], a memory matrix saves the last k items each user u interacted with. The model then recognizes which of a user's past interactions can best help to predict the preference towards a candidate item i:

$$w_{i,j} = q_i^T \cdot m_j^u \ , \tag{2.4.1}$$

$$z_{i,j} = softmax(w_{i,j}) = \frac{\exp(\beta w_{i,j})}{\sum_l \exp \beta w_{i,l}} \ , \forall j = 1, 2, ..., k \ , \tag{2.4.2}$$

where q_i is the candidate item embedding, m_j^u the j's item embedding in user u's memory matrix and β a strength parameter. $z_{i,j}$ is therefore an importance weight, indicating the level of similarity between a candidate item i and each past item the user interacted with. These scores are normalized as an importance distribution and used to weight the k past items impact when constructing the user latent vector, p_u:

$$p_u = \sum_{j=1}^{k} z_{i,j} \cdot m_j^u \ , \tag{2.4.3}$$

By applying the attention mechanism, the model generates a user latent representation dedicated to suit the candidate item. The embeddings of past items that are closely related to item i will be further emphasized in the user representation.

In a similar fashion, [60] constructs a user embedding by focusing on the most relevant historic news items compared to a candidate article. The attention weights, however, are generated by feeding the concatenated item embeddings to an MLP:

$$w_{i,j} = H([q_i, q_j]) \ , \tag{2.4.4}$$

where q_* is an item embedding and H a multi-layer perceptron that outputs a score in the range of $[0, 1]$. The resulting weights are normalized and used to generate the user embedding as done in Eq. 2.4.1 and 2.4.2. A different weighting function is introduced in [16], where the authors integrate opinions from other users regarding the candidate item. Instead of combining past interactions to construct a user latent vector, attention scores are learned based on the similarity between user u and other users in her neighborhood, along with the opinion of these users:

$$w_{i,u,v} = m_u^T m_v + e_i^T m_v \ , \tag{2.4.5}$$

where m_* is a user memory matrix with k past items, e_i the candidate item's embedding and v is a user who had a previous interaction with i.

While the standard attention network is based on comparing tuples of inputs one at a time to determine importance, self-attention, the most recent development in attention-based learning, is composed of measuring the internal components of the input to identify the most valuable ones [34, 57]. Originally introduced in tasks of textual summation [45] and translation [2, 14], this technique was successfully implemented in encoder-decoder models, where one network is responsible to encode a textual input as a concise vector while the other to decode it as a different output. However, self-attention was found to be applicable in other frameworks as well. In [34], it was used to label important words in tasks involving textual embedding. By applying self-attention over the collection of n word embeddings, denoted as $V \in \mathbf{R}^{n \times d_1}$, the following importance distribution of words can be learned:

$$a = softmax(w_2 tanh(W_1 V^T)) \ . \tag{2.4.6}$$

The use of $W_1 \in \mathbf{R}^{d_2 \times d_1}$ and $w_2 \in \mathbf{R}^{d_2}$ transforms the input matrix V to the vector a of size n, where each cell represents the importance weight of the respective word.

Due to its novelty, there are only few works using self-attention in recommender systems. [72] generates a user embedding vector by employing self-attention in an encoder framework over the sequence of items the user interacted with. However, the model does not output an item recommendation, but only the user latent vector. [24] utilizes self-attention to identify the most important past items in the user's history. The authors adopt a similar method to the one proposed in [57] by generating three transformations of the input embedding and performing a weighted average.

In this work, we propose two self-attention frameworks, SARAH and DARIA. While the former implements self-attention to learn the user and item latent representations, the latter captures their combined interaction by employing standard neural attention. Although each suggested method adopts a different technique, both introduce a novel approach. To the best of our knowledge, SARAH is first to represent each user and item by feeding past interactions and attributes to separate self-attention layers. Unlike other works that utilize attention to construct latent representations, the use of self-attention allows SARAH to model users and items independently. DARIA on the other hand, will be composed of two stacked neural attention layers, where one identifies the most relevant user past items with regard to a candidate item and the other compares these items' attributes.

2.5 Explainable Recommendations

Although matrix factorization based recommender systems are getting more accurate and are able to efficiently process more data, the heavy use of latent factors results them in being perceived as a black box, only able to predict a recommendation. However, as found in previous work [21], the ability to justify a suggestion is vital and improves the trust users have in the system and the likelihood to retain them. Explainable recommender systems are therefore an important component of RS, dedicated to generating recommendations along with human-interpretable reasoning. They are often classified as either post-hoc or embedded models. Post-hoc methods [55] are usually model-agnostic and based on generating explanations from an already trained model using pre-defined candidates, extracted from external data, i.e. tags and text. Embedded models [39] learn explainable factor while simultaneously providing recommendations. Although they are less flexible, embedded methods have a broader range of explainable factors to choose from, and they can not only optimize the given suggestions, but also the accompanying reasoning. We therefore intend to focus on embedded explainable models in this work.

Facing with the increasing popularity of deep learning, the challenge of explainable deep recommender systems is to utilize the strengths of neural networks in modeling users and items using multiple layers, while retaining enough data to explain the system's output. An approach that was found to be successful in previous works [4, 44] is the use of knowledge graphs to define the system's inputs and their relation. This is usually done by extracting known entities from the input and representing their relations in the form of graph edges. However, knowledge graphs cannot be successfully applied to all fields in an equal manner. For example, while it may be possible to extract entities from news items, it is considerably more challenging to do so for restaurants and venues.

The successful implementation of attention networks in recommender systems introduced a new opportunity for providing explanations, by enabling a deep model to be composed of vectors representing the importance of various inputs. [8] identifies the most valuable user reviews to be accompanied with the recommended items. First, a concise representation of each past review is learned using CNN. Then, attention scores are generated by combining each review with the user who wrote it. The resulting importance distribution is used to construct an item representation, where the embeddings of valuable reviews get the higher impact.

In [62], the authors embed user and item properties using a tree-based model. The most relevant attributes to the given interaction are then determined by an attention network. User and item features are therefore compared jointly, resulting some recommendations generated from user preferences, while some rely strongly on the item attributes. Similarly, [9] compares item features along with the target user to find the traits that are most important to her. These features are then presented to the end user either as an importance distribution to justify generated recommendations or as a personalized message. [17] integrates neural attention and a neighborhood-based CF technique, by replacing the similarity parameter between two items with an attention score. More specifically, it generates attention weights between the candidate item and each of the target user's past items. The obtained weights are multiplied by the rating previously given by the user for each respective item, to result in a weighted average of past ratings, altered by the degree of relevance to the candidate item.

Unlike previously described models, in this work we introduce two explainable recommender systems composed of neural attention and self-attention layers over the user's context and the item's attributes. By implementing self-attention, SARAH is able to learn an interpretable representation of its inputs with no direct dependence to the current interaction, while DARIA utilizes neural attention to explain its recommendations. To the best of our knowledge, SARAH is the first method to exploit self-attention over users and items for the task of explain-ability. A resembling work is [51], where the model's reasoning is based on a movie's content and user's past activity, however this is done by clustering and not by methods of collaborative filtering or deep learning.

Chapter 3

Deep POI Recommendation

In this chapter we extend the focus on point-of-interest recommendation and propose Textual and Contextual Embedding-based Neural Recommender (TCENR), a framework for deep recommendation in location-based social networks. TCENR attempts to learn two perspectives of the same user-location interaction, where one is centered around past interactions while the other adopts textual reviews. Although numerous works established the potential of recommender systems based on deep learning, most have focused on only a single type of neural network that best suited their given task. However, in this work, we intend to incorporate multiple paradigms, each best suited to its given input, to provide POI recommendation.

Section 3.1 presents the motivations for developing TCENR, which is described in Sections 3.2 and 3.3. An extension that employs recurrent neural networks is developed in Section 3.4, before providing experimental results and intermediate conclusions in 3.5.

3.1 Recommendations In LBSNs

While POI recommendation methods usually attempt to solve a similar problem as the standard recommender system, they are required to overcome additional challenges.

Due to the physical distance between users and most locations, the list of possible recommendations is extremely narrow. However, when users go to infrequently visited areas, their representation might not be suited to the new environment. This is in fact a worsened case of the data sparsity and cold-start problems.

A common method to mitigate the sparsity issue is data extraction from additional sources. External inputs can allow a model to learn more general representations of users and items, while focusing on available features. Our proposed framework will take as input the historical user visits, reviews written by the user and about the location, user's friends in the social network and locations' geographical area. By fitting the different inputs to appropriate sub-models that are optimized jointly, TCENR learns more accurate representations for users and locations. In addition, implementing different techniques allows the network to focus on the more significant inputs in diverse scenarios. More specifically, textual reviews are described as word vectors, while the data from social networks and geo-spatial distances is captured in the form of a graph. Each user or location is defined as a graph node and edges determine the social connection between two users or locations' geographical proximity.

We adopt implicit feedbacks as the trained target feature, allowing TCENR to be applied in various recommendation scenarios. This enable our framework to make recommendations when no rating data is available, by defining a user visit as the positive instance in a two-class prediction scenario. In other words, we will attempt to determine whether a given location is likely to be visited by the target user or not. However, this results in another challenge to the POI recommender. While all observed interactions are positive, training a two-class model requires negative instances as well. Defining all unobserved interactions as negative, though, will result in an unrealistic distribution, due to the high concentration of such instances in our sparse setting. We therefore follow previous work [53, 70] and apply *negative sampling*, where a pre-determined number of unobserved items are sampled for each known user-item interaction.

The main contributions of the work described in this chapter are as follows:

1. We present TCENR, a framework that jointly trains MLP and CNN to provide POI recommendations, as well as a variation, $\text{TCENR}_{\text{seq}}$, that performs the same task while adopting RNN instead of the CNN component.

2. To the best of our knowledge, no work has been done in jointly training MLP and CNN for the task of POI recommendation using social networks, geographical locations and natural language reviews as inputs.Although the proposed solution has been developed to provide recommendations for specific types of inputs (i.e. reviews, social network and geo-spatial), we claim it can be easily generalized to a framework able to support additional features.

3. Evaluated over our the Yelp dataset, our proposed frameworks were found to consistently outperform seven state-of-the-art baselines in terms of accuracy, MSE, precision and recall. By comparing the two alternatives to our suggested model, we provide insight towards the impact gained by analyzing textual reviews as a secondary input to the common past interactions, as well as a comparison of CNN and RNN for the task of sentiment analysis in the same experimental settings. We further present comprehensive analysis over the most important hyper-parameters and design selections of our proposed networks, shedding some light over the different components of deep neural networks in the task of POI recommendation.

3.2 Textual And Contextual Neural Recommender

The following recommender system aims to improve the POI recommendation task by learning user-location interactions using two parallel neural networks, as shown in Fig. 3.2.1. The context-based network, presented in the left part of the figure, is designed to model the user-POI preferences using social and geographical attributes as regularizing factors [37,65] and based on a multi-layer perceptron structure [12,20]. Shown in the right side of Fig. 3.2.1, the convolutional neural network is responsible for the textual modeling unit [25, 71]. It attempts to learn the same preference by analyzing the underlying meaning in users' and locations' reviews. Each of the two networks is based on modeling the user and POI input individually with regard to their shared interaction, defined in the merge layers. The resulting concise vectors, representing the user-location interaction learned from each perspective, are then combined in a final layer, responsible for generating a recommendation.

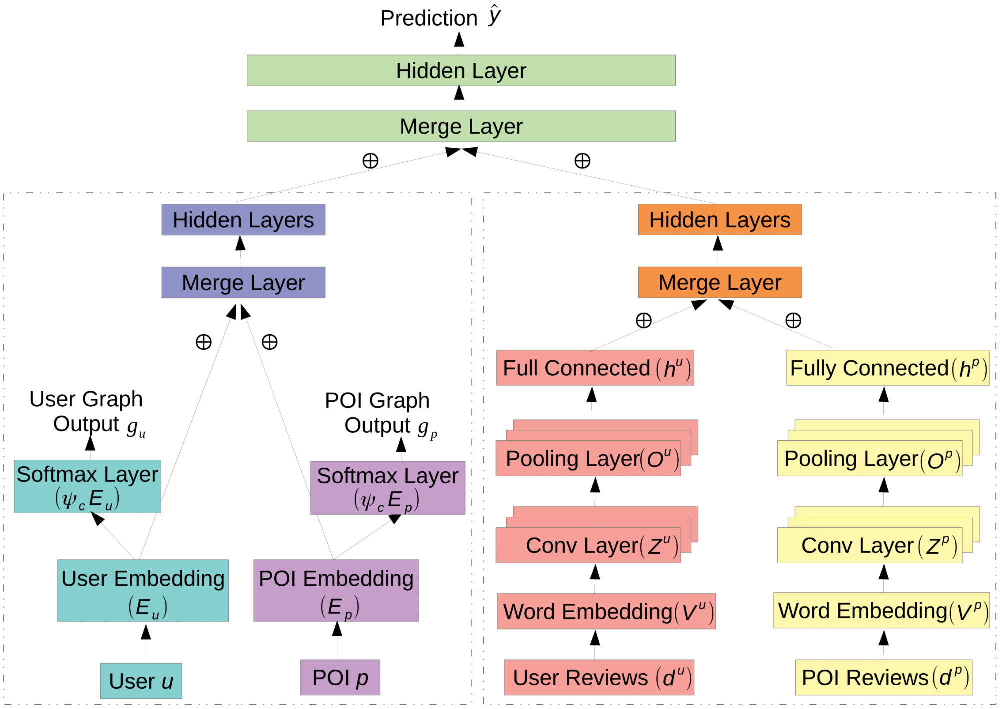


Figure 3.2.1: TCENR Framework

3.2.1 Context-based network layers

To better capture the complex relations between users and locations in a LBSN, we chose to adopt the multi-layer perceptron architecture. By stacking multiple layers of nonlinearities, MLP is capable of learning relevant latent factors of its inputs. It is first fed with user and location vectors of sizes N and M, where each input tuple $< u, p >$ is transformed into sparse one-hot encoding representations. The two fully connected embedding layers, found on top of the input layer, project the sparse representations of users and locations into smaller and denser vectors. For u and p, the respective embedding matrices are $E_u \in \mathbf{R}^{k_u \times N}$ and $E_p \in \mathbf{R}^{k_p \times M}$, where k_u and k_p are the corresponding dimensions.

We assume that friends usually share similar preferences towards places they visit, and therefore exploit the users' social networks to constrain their representations within the network. The same reasoning is inferred for locations in the same geographical region, as they might be different, but yet share many attributes in common compared to far away venues. We apply this logic to the contextual MLP in the form of smoothing, where the embeddings of connected users and locations in their respective graphs are constrained to be relatively similar. Two softmax layers take the user and location representations, E_u and E_p, as input and transform them back to N and M sized vectors, respectively. The user output layer, $\psi_c E_u \in \mathbf{R}^N$ can be formally described as:

$$\psi_c E_u = a(W_c^u \times E_u + b_c^u) \ , \tag{3.2.1}$$

where W_c^u and b_c^u are the layer's weight matrix and bias vector and a is a non-linear activation function. Due to the similarity between the user and POI specific layers, the location output layer, $\psi_c E_p$, will not be developed in this section. As part of its optimization, the proposed network will penalize differences between the two softmax outputs, $\psi_c E_u$ and $\psi_c E_p$, to the user and location respective graphs, g_u, and g_p. This in turn enforces the smoothing and similarity between connected embeddings.

The two representations main purpose is, however, to learn the user-location interaction, and in turn provide an implicit prediction. The two embeddings are therefore projected to a merge layer and combined by the concatenation operator. Using concatenation instead of dot-product allows the embeddings to be in different dimensionality, which in turn improves the generated representation [20]. By feeding the two latent vectors to an MLP, the network attempts to learn the interactions between different user and location hidden features. This in turn, allows it to eventually describe the interaction using a small number of parameters. As the input for the following neural network, the merge layer can be represented as H_0 where:

$$H_0 = [E_u, E_p]. \tag{3.2.2}$$

Since simple concatenation of the user-location embedded vectors does not allow for interactions to be modeled, hidden layers are added to learn these connections. We adopt the MLP structure described in Eq. 2.2.1, where the ReLU is employed as the activation function for its layers. More formally, the l-th hidden layer can be defined as:

$$H_l^{context}(x) = ReLU(W_l H_{l-1}^{context}(H_0) + b_l)\ , \tag{3.2.3}$$

where W_l and b_l are the l-th layer parameters. Unlike the standard MLP model, described in section 2.2, the final hidden layer's output, $H_L^{context}$ will not be directly used as the model's output, but will be combined with the second sub-network of our framework.

3.2.2 Textual modeling network layers

To improve the model's coverage and gain further insight towards the interests of users and what defines locations, a textual-based network is introduced. It simultaneously learns the same interaction as the contextual-based network, but with a natural language input. Two additional vectors d^u and d^p, representing user u's and location p's textual reviews, respectively, are applied as inputs for this network. Each vector is comprised of all n words written by the user or about the location merged together, kept in their original order. These words are then mapped to c-dimensional vectors defining their semantic meaning in the following embedding layers. The output of the user embedding layer is the representation of all words used by a user u in the form of a matrix, and can be denoted as:

$$V^u = [\phi(d_1^u), ..., \phi(d_l^u), ..., \phi(d_n^u)] \ , \tag{3.2.4}$$

where $\phi : D \to \mathbf{R}^{k_w}$ is a lookup function to a pre-trained textual embedding layer [46] that represents each word in vocabulary D as a vector in size k_w. Similarly, V^p denotes the word embedding matrix for location p.

Due to the large amount of parameters required to train the aforementioned contextual model, the textual network is implemented using a CNN-based architecture which is usually more computationally efficient than RNN. The semantic representations of users' and locations' reviews are fed to convolution layers, to detect parts of the text that best capture the review's meaning. These layers produce feature maps over the embedded word vector, using a window size of ws and filter $K \in \mathbf{R}^{k_w \times t}$. As suggested by [71], ReLU is used as an activation function for this layer:

$$\begin{aligned} z_{jl}^u &= ReLU(V_{l:l+ws}^u * K_j^u + b_j^u) \ , \\ z_j^u &= [z_{j1}^u, ..., z_{jl}^u, ..., z_{jn}^u] \ , \end{aligned} \tag{3.2.5}$$

where V_l^u is user u's l-th input word embedding and z_j^u the j-th feature, extracted from the complete text.

Based on the standard CNN structure presented in Eq. 2.2.7-2.2.8, feature maps produced by the convolution layers are reduced using a pooling layer:

$$\begin{aligned} o_j^u &= max(z_j^u) \ , \\ O^u &= [o_1^u, ..., o_j^u, ..., o_t^u] \ , \end{aligned} \tag{3.2.6}$$

where max-pooling is selected to identify the most relevant words. These are followed by hidden layers that jointly model the different feature maps to result in the latent representations of the user and location, denoted respectively as h_u and h_p :

$$h_u = ReLU(W_1^u \times O^u + b_1^u) \ . \tag{3.2.7}$$

Similarly to the contextual sub-network, presented in section 3.2.1, we aim to learn a vector describing the user-location interaction rather than the two components separately. We therefore utilize yet another $ReLU$ layer to combine the two representations, where the user and location vectors are concatenated:

$$h_{reviews} = ReLU(W_2 \times [h^u, h^p] + b_2) \ . \tag{3.2.8}$$

The outputs generated from the two neural networks are then finally merged to produce a prediction $\hat{y}_{up} \in [0, 1]$. The last layers of the two networks, each representing a different view of the same user-location interaction, are concatenated and fed to an hidden layer, responsible to blend the learning and transform it to an implicit score:

$$\hat{y}_{up} = \sigma(W_3 \times [h_{context}, h_{reviews}] + b_3) \ , \tag{3.2.9}$$

where W_3 and b_3 are the layer's parameters and determine the impact of each contextual and textual feature over the output. The sigmoid function was selected to transform the hidden layer output to the desired range of $[0, 1]$.

3.3 Training the Network

To train the recommendation models, we adopt the point-wise loss objective function, as done in [20, 65, 71], where the difference between the prediction $\hat{y}_{up}$ and the actual value y_{up} is minimized. To address the implicit feedback nature of LBSNs, for each positive case in the given training set, Y, we sample a set of negative instances, denoted as $Y^{\text{-}}$.

Due to the implicit feedback nature of the recommendation task, the algorithm's output can be considered as a binary classification problem. As the sigmoid activation function is being used over the last layer, the output probability can be defined as:

$$p(Y, Y^{\text{-}} \mid E_u, E_p, V^u, V^p, \Theta_f) = \prod_{(u,p) \in Y} \hat{y}_{up} \prod_{u,p' \in Y^{\text{-}}} (1 - \hat{y}_{up'}) \ , \tag{3.3.1}$$

where E_u and E_p are the embedding layers for all users and locations, respectively. Similarly, V^u and V^p are the textual reviews embedding layers and Θ_f represents the model parameters. Taking the negative log-likelihood of p results in the binary cross-entropy loss function for the prediction portion of the model:

$$L_{pred} = - \sum_{(u,p) \in Y \cup Y^{\text{-}}} y_{up} log(\hat{y}_{up}) + (1 - y_{up}) log(1 - \hat{y}_{up}) \ . \tag{3.3.2}$$

Minimizing Eq. 3.3.2 will optimize the model parameters to result in more accurate predictions. However, there are two more outputs in the model, the users' social network $\psi_c E_u$ and the locations' distance graph $\psi_c E_p$. We further wish to penalize the model when embeddings of connected users or locations are different. Two additional loss functions are therefore required to train the contextual sub-network and the whole model as a result. We follow derivations presented in [65] and employ the categorical cross-entropy loss to minimize the difference between the softmax and the user graph distributions:

$$L_{u\_context} = -\sum_{(u,u_c)} log(\psi_c E_u - log \sum_{u_{c'} \in C_u} exp(\psi_{c'} E_u)) \ , \tag{3.3.3}$$

where u_c is user u's context, C_u is the set of all possible contexts and $\psi_c E_u$ is the user embedding softmax, as defined in Eq. 3.2.1. Taking the binary class label into account prompts the following loss function, corresponding with minimizing the cross-entropy loss of user u and context c with respect to the y class label:

$$L_{u\_context} = -\mathbf{I}(y \in Y) log\sigma(\psi_c E_u) - \mathbf{I}(y \in Y^{-}) log\sigma(-\psi_c E_u) \ , \tag{3.3.4}$$

where $\mathbf{I}$ is a function that returns 1 if y is in the given set, and 0 otherwise. The same logic is used to formulate the loss function for the POI context:

$$L_{p\_context} = -\mathbf{I}(y \in Y) log\sigma(\psi_c E_p) - \mathbf{I}(y \in Y^{-}) log\sigma(-\psi_c E_p) \ , \tag{3.3.5}$$

We simultaneously minimize the three loss functions L_{pred}, $L_{u\_context}$ and $L_{p\_context}$. The joint optimization improves the recommendation accuracy while enforcing similar representations for locations in close proximity and users connected in the social network. The loss functions are combined using two hyper-parameters, λ_1 and λ_2 to weight the contextual contribution:

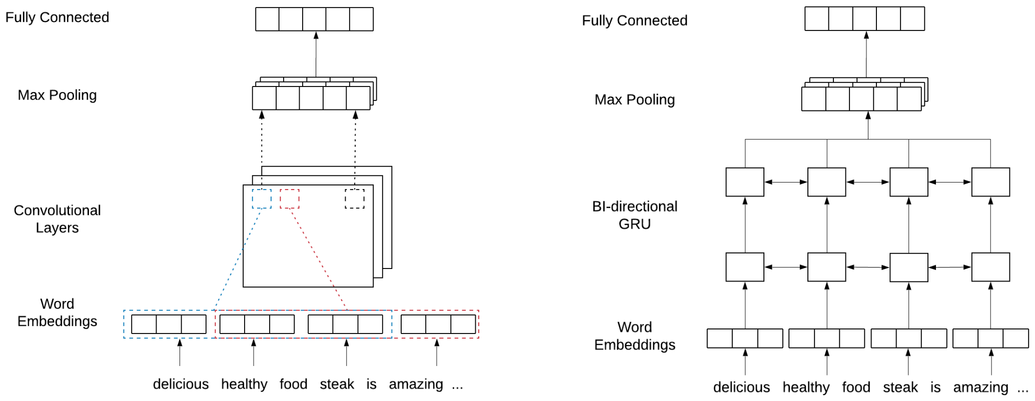
$$L = L_{pred} + \lambda_1 L_{u\_context} + \lambda_2 L_{p\_context} \ . \tag{3.3.6}$$

To optimize the combined loss function, a method of gradient descent can be adopted, and more specifically we utilize the Adaptive Moment Estimation (Adam) [27]. This optimizer automatically adjusts the learning rate and yields faster convergence than the standard stochastic gradient descent in addition to making the learning rate optimization process more efficient. In order to avoid additional over-fitting when training the model, an early stopping criteria is integrated. The model parameters are initialized with Gaussian distribution, while the output layer's parameters are set to follow uniform distribution.

3.4 Textual Modeling Using Word Sequences

To further investigate the gain achieved by integrating a textual modeling component over reviews in TCENR, we suggest an extension, denoted as TCENR$_{\text{seq}}$. Following its success in previous language modeling tasks [1, 63] and its ability to capture sentences' sequential nature, we employ an RNN component to learn latent features from reviews. An illustration of the proposed extension is presented in Figure 3.4.1b, while the CNN method used in the vanilla TCENR is shown in Figure 3.4.1a, to provide a convenient base for comparisons. More specifically we follow the findings of previous works [3, 5] and implement our recurrent network using GRU, an architecture that achieves competitive performance compared to LSTM, but with fewer parameters, making it more efficient.

Since the context of a word can be determined by other preceding and successive words or sentences, our proposed method employs a bi-directional GRU over the user embedding, V^u, and the location, V^p. Each word l's hidden state is learned by forward and backward GRU layers, denoted as $\overrightarrow{h_l^1}$ and $\overleftarrow{h_l^1}$, respectively. While the forward pass is identical to h_l described in Eq. 2.2.6, the backward pass of a word l



(a) Textual modeling component using CNN (b) Textual modeling component using RNN

Figure 3.4.1: Proposed alternatives to learn user and location representations from textual reviews. 3.4.1a is a CNN-based solution employed in TCENR, while 3.4.1b illustrates the suggested extension using RNN

requires the hidden state of successive words and can be fully defined as:

$$
\begin{aligned}
\overleftarrow{f}_l &= \sigma[W_f[\overleftarrow{h}_{l+1}, V_l^u] + b_f] \ , \\
\overleftarrow{s}_l &= \sigma[W_s[\overleftarrow{h}_{l+1}, V_l^u] + b_s] \ , \\
\overleftarrow{\tilde{c}}_l &= tanh(W_z V_l^u + f_l \cdot W_h \overleftarrow{h}_{l+1} + b_{\tilde{c}}) \ , \\
\overleftarrow{h}_l &= (1 - \overleftarrow{s}_l) \cdot \overleftarrow{h}_{t+l} + \overleftarrow{s}_l \cdot \overleftarrow{\tilde{c}}_l \ ,
\end{aligned}
\tag{3.4.1}
$$

where f_l is the forget gate, s_l is the output gate, $\tilde{c}_l$ is the new candidate state and h_l is current state for word l.

To learn a more concise and combined representation of a word while taking into account the context of all surrounding words, we feed the concatenation of $\overrightarrow{h_l^1}$ and $\overleftarrow{h_l^1}$ to an additional bi-directional GRU layer, such that its input for every word l is $e_l^2 = [\overrightarrow{h_l^1}, \overleftarrow{h_l^1}]$. The second recurrent unit will output n latent vectors, each is a sequentially infused representation of a word written by the target user or about the candidate item.

We further feed all modified word vectors to the pooling and fully connected layers, presented in Equations 3.2.6 and 3.2.7, respectively. By doing so, we allow the method of textual modeling to be the only variant between TCENR and TCENR$_{\text{seq}}$, and further reduce the number of learned parameters. Replacing only the convolutional layer with that of the GRU, enables us to directly determine the effect RNN has on textual modeling for POI recommender systems compared to CNN, as well as enabling the model to learn a more concise user and location representations. As in TCENR, the resulting vectors will be merged in order to learn the user-location interaction.

3.5 Model Evaluation

In this section we perform empirical experiments to evaluate TCENR and its extension, TCENR$_{\text{seq}}$. We further present multiple design selections to the proposed frameworks along with their impact.

3.5.1 Experimental Setup

To evaluate our suggested model, we use Yelp's real-world dataset[1]. It includes a subset of textual reviews along with the users' friends, and the businesses' geographical locations. Due to the limited resources used in the model evaluation, we chose to filter the dataset and keep only a concise subset, where all users and locations with less than 100 written reviews or less than 10 friends are removed. The filtered dataset includes 141,028 reviews, and 98.08% sparsity for the rating matrix. The social and geographical graphs were constructed by random walks. 10% of the original vertices were sampled as base nodes, while 20 and 30 vertices were connected to each base node for users and locations, respectively, with a window size of 3. To build the POI graph, two locations are considered directly connected if they are up to 1 km apart.

We test our framework's performance by splitting the original data to training-

[1] https://www.yelp.com/dataset/challenge

validation-test sets using random sampling, with the respective ratios of 56%-24%-20%, resulting 78,899 training instances. In addition, the input data was negatively sampled with 4 negative locations for every positive one.

To effectively compare our proposal with other alternatives, we adopt the same settings as applied in [20, 65]. The MLP input vectors are represented with an embedding size of 10, while two layers are added on top of the merged result. Following the tower architecture, where the size of each layer is half the size of its predecessor, the number of hidden units are 32 and 16 for the first and second layers, respectively.

In the CNN component each word is represented by a pre-trained embedding layer with 50 units, while the convolutional layer is constructed with a window size of 10 and a stride of 3. It results 3 feature maps that are flattened after performing the max-pooling operation with a pool size of 2. The results are further modeled by a hidden layer with 32 units. Following the merge of the two hidden units, their interaction is learned using another layer with 8 units. To combine the three loss functions as described in Eq. 3.3.6, we follow the results of [65] and set the hyper-parameters $\lambda_1 = \lambda_2 = 0.1$. For the training phase of the model, a learning rate of 0.005 was used over 50 maximum epochs and a batch size of 512 samples.

3.5.2 Baselines

To evaluate our algorithm, we chose to compare it to these seven, empirically proven, frameworks:

- HPF [19]. Hierarchical Poisson matrix Factorization. A Bayesian framework for modeling implicit data using Poisson Factorization.

- NMF [36]: Non-negative Matrix Factorization, a CF method that takes only the rating matrix as input.

- Geo-SAGE [61]: A generative method that predicts user check-ins in LBSNs using geographical data and crowd behaviors.

- LCARS [66]: Location Content Aware Recommender System. A probabilistic model that exploits local preferences in LBSN and content information about POIs.

- NeuMF [20]. Neural Matrix Factorization. A state-of-the-art model combining MF with MLP on implicit ratings.

- PACE [65]. Preference And Context Embedding. A MLP-based framework with the addition of contextual graphs' smoothing for POI recommendation.

- DeepCoNN [71]. Deep Cooperative Neural Networks. A CNN-based method that jointly learns an explicit prediction by exploiting users' and locations' natural language reviews.

For the task of evaluating our model and the baselines, we chose to apply the following popular metrics:

- **Accuracy** - Presents the ratio of correct predictions: $\frac{1}{T}\sum \mathbf{I}(\hat{y}_{up} = y_{up})$. T is the test set size and **I** returns 1 if the prediction and ground truth are equal.

- **Mean Square Error (MSE)** - Accumulates the difference between the real-valued prediction and the ground truth: $\frac{1}{T}\sum(\hat{y}_{up} - y_{up})^2$.

- **Pre@10** - Precision for a list of top-10 recommendations. Can be described as the number of locations the target user is interested in, out of her 10 highest predicted test instances.

- **Rec@10** - Recall for a list of top-10 recommendations. The ratio of relevant instances that are included in the target user's 10 highest predicted locations.

The proposed models were implemented using Keras[2].

[2] https://keras.io

3.5.3 Performance Evaluation

The performance of our proposed algorithms and the seven baselines is reported in Table 3.5.1, along with the improvement ratio of TCENR over each method in brackets. The presented results are based on the average of three individual executions.

Table 3.5.1: Performance comparison over the Yelp dataset. Improvement of TCENR compared to each method is shown in brackets

Model	**Accuracy**	**MSE**	**Pre@10**	**Rec@10**
HPF	0.8141	0.1800	0.5526	0.3699
	(1.69%)	(34.94%)	(18.51%)	(40.98%)
NMF	0.8222	0.1189	**0.7851**	0.3517
	(0.69%)	(1.51%)	**(-16.58%)**	(48.28%)
Geo-SAGE	0.7995	0.1807	0.2912	0.4145
	(3.55%)	(35.19%)	(124.89%)	(25.81%)
LCARS	0.8142	0.1612	0.6408	0.5127
	(1.68%)	(27.36%)	(2.2%)	(1.72%)
NeuMF	0.8273	0.1421	0.6488	**0.5586**
	(0.07%)	(17.59%)	(0.94%)	**(-6.64%)**
Pace	0.8239	0.1186	0.6406	0.5049
	(0.49%)	(1.26%)	(2.23%)	(3.29%)
DeepCoNN	0.8037	0.1454	0.5385	0.323
	(3.01%)	(19.46%)	(21.62%)	(64.46%)
TCENR	**0.8279**	0.1171	0.6549	**0.5215**
TCENR$_{seq}$	0.8273	**0.1161**	**0.6655**	0.4738
	(0.07%)	**(-0.86%)**	**(-1.59%)**	(10.07%)

As can be witnessed from the results, the proposed model, TCENR, achieves the best results overall compared to all baselines. Furthermore, it was found to significantly outperform HPF, NMF, Geo-SAGE, LCARS, Pace and DeepCoNN for $p < 0.05$ based on a one-sided unpaired t-test in terms of accuracy and MSE. The contrasting results in terms of precision and recall compared to NeuMF suggests that TCENR offers less, but more relevant recommendations to the user. While NMF provides the best precision score compared to all methods, it under-performs in all other measures, making it a less desirable model. Taking a closer look shows that, surprisingly, NeuMF outperforms PACE in accuracy, precision and recall. This may be due to the less sparse dataset tested, which does not allow the contextual regularization to be fully harvested. In addition, the use of only the first 500 words to represent the textual input for each user and location may explain the relatively low scores of the DeepCoNN model on the dataset, while the performance of Geo-SAGE and LCARS demonstrates that relying solely on geographical data does not allow such models to fully capture users' preferences in LBSNs.

Comparing TCENR and its proposed extension $\text{TCENR}_{\text{seq}}$ provides contrasting results. By employing RNN instead of CNN to extract user and location features from textual reviews, $\text{TCENR}_{\text{seq}}$ achieves lower error rate and improved precision score, while accuracy and recall are worsened. It may be considered that by accurately capturing different aspects from user reviews, the model is able to reinforce its hypotheses and therefore reduce the uncertainty in some cases. However, when faced with a contrast between textual aspects and the ground truth, it might choose the wrong class label. Nonetheless, the results demonstrate the importance of adopting the most suitable techniques and measures to learn different data types, rather than employing a single method over all inputs. Moreover, it shows the positive impact of using textual data in conjunction to historical activities. The reported performance further suggests additional insight towards the selection of CNN and RNN for the task of language modeling in future recommendation tasks.

To further evaluate our suggested frameworks and the seven baselines in terms of runtime, the average time required to fully train each method is presented in Figure 3.5.1. As demonstrated by the results, TCENR is competitive with most baselines, and found to be more efficient than DeepCoNN and LCARS. The reported runtime of $\text{TCENR}_{\text{seq}}$ further demonstrates the relative efficiency of CNN-based solutions for textual modeling tasks. As the number of trainable parameters is increased due to the

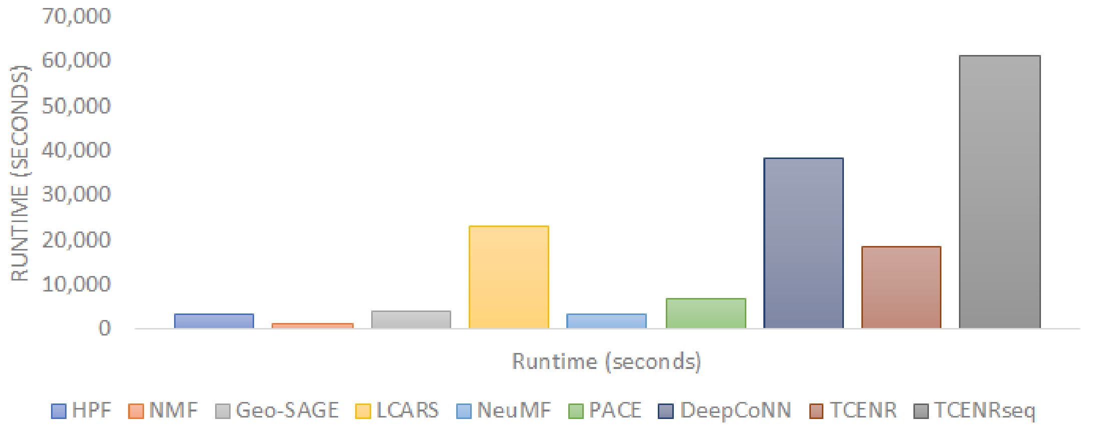


Figure 3.5.1: Runtime (seconds) of all models on the Yelp dataset

use of recurrent layers, our RNN-based extension takes 329% longer to train compared to TCENR, while achieving comparative results.

3.5.4 Model Design Analysis

In this section we discuss the effect of several design selections over the suggested model's performance.

Merge Layer

The importance of the model's final layers, responsible for combining the dense output of both the MLP and convolutional networks, requires a close attention, as it effects the networks' ability to jointly learn and the prediction itself. To properly select the fusion operator the following methods had been considered:

- Combining the last hidden layers of the two models using concatenation. A model using this method will be denoted as $TCENR_{con}$ and described in Eq. 3.2.9.

- Merging the last hidden layers using dot product, resulting a model named

$TCENR_{dot}$ that can be defined as:

$$\hat{y}_{up} = h_{context} \cdot h_{reviews} \ . \tag{3.5.1}$$

- Combining the two previously described methods, where the two representations will be jointly learned by concatenation and dot product. The resulted model will be denoted as $TCENR_{dot\_con}$ and can be developed by combining Eq. 3.2.9 and Eq. 3.5.1 using addition and translating the result to a range of $[0, 1]$ with the sigmoid function:

$$\hat{y}_{up} = \sigma(\sigma(W_4 \times [h_{context}, h_{reviews}] + b_4) + h_{context} \cdot h_{reviews}) \ . \tag{3.5.2}$$

- Adopting a weighted average for the prediction result of the two networks. Denoted as $TCENR_{weight}$, this model can be defined as:

$$\hat{y}_{up} = \lambda_3 \sigma(W_5 \times h_{context} + b_5) + \lambda_4 \sigma(W_6 \times h_{reviews} + b_6) \ . \tag{3.5.3}$$

As shown in Figure 3.5.2, adopting the more simple methods of weighted average and dot product leads to an inferior performance to TCENR, demonstrating the added value of utilizing the latent features learned by each sub-network jointly. When combined with the under-performing method of dot product in $TCENR_{dot\_con}$, the use of concatenation improves over dot product alone. However, since the two methods are integrated using a simple average, employing only concatenation as done in $TCENR_{con}$ produces the best results, and therefore integrated into the final model.

MLP Layer Design

Although found by [20] that adding more layers and units to the MLP-based recommender has a positive effect, the use of CNN and the additional hidden layer suggests

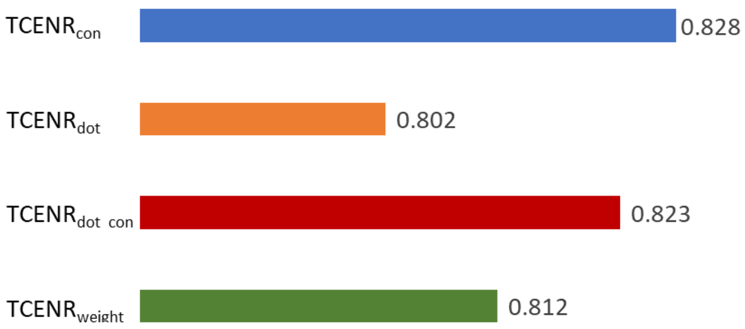


Figure 3.5.2: Comparison of merging methods in terms of accuracy

it is a subject worth investigating. To this end, we test the proposed algorithm with 1-4 hidden layers used to learn the user-item interaction with contextual regularization in varying sizes from 8 to 128 hidden units. The results in terms of test set's accuracy are presented in Table 3.5.2, where the number of hidden layers is defined as columns and the size of the first unit is presented as rows. Unlike previous results, we find that two hidden layers with 32 and 16 units result in the best performance for our dataset.

Table 3.5.2: Model's accuracy with different layers

1st layer	**H=1**	**H=2**	**H=3**	**H=4**
16	0.824	0.827	-	-
32	0.823	**0.837**	0.825	-
64	0.822	0.829	0.83	0.827
128	0.823	0.828	0.829	0.827

Number of Words

The use of written reviews in their original order allows the strengths of CNN and RNN to be exploited by finding the best representation for every few words, and eventually for the whole text. Our final dataset, however, is composed of very long reviews, where to fully learn a single user or location, more than 20,000 words are required, making it computationally expensive to extract relevant representations. To benefit from the sequential nature of the written reviews while keeping the solution feasible, the number of words was limited to a range of 500-6,000. As can be witnessed from Figure 3.5.3, there is a slight improvement in accuracy as the number of words increase up to 3,000, while additional words result in an increased bias towards users and locations with longer reviews, and in turn reduces the model learning capabilities.

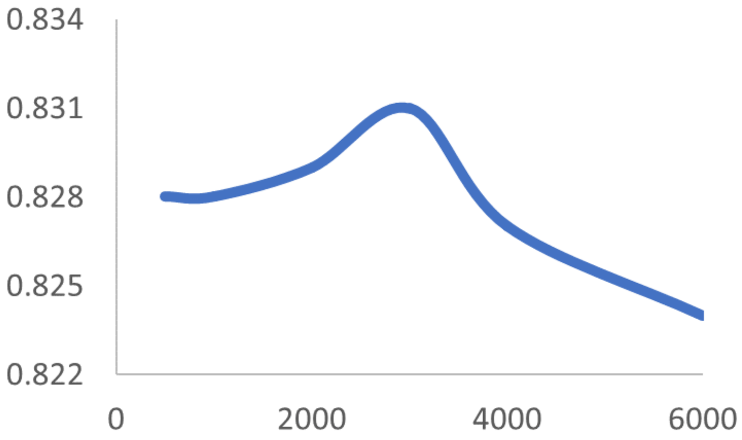


Figure 3.5.3: Number of words comparison in terms of accuracy

Chapter 4

Explainable Neural Attention Frameworks

In this chapter we advance our focus and suggest novel deep learning frameworks that not only generate accurate recommendations and mitigate the data sparsity problem, but also yield meaningful explanations. Our proposed methods provide reasoning in the form of importance weights, where items are described by their most defining attributes and users by past interactions.

Two different frameworks are proposed in this chapter, both based on the premise of neural attention. First, we will present *Self-Attention Recommender based on Attributes and History* (SARAH), a recommendation framework utilizing the rising technique of self-attention to simultaneously represent users and items. In contrast to other attention-based recommender systems, SARAH includes two self-attention components, used to identify the most important input features for each user, independently of the candidate item, and vice versa.

Then, a variation to SARAH will be developed, denoted as *Dual Attention Recommender with Items and Attributes* (DARIA). DARIA is a recommender system that employs two consecutive layers of neural attention. As one layer focuses on the most relevant items a user previously rated with regard to the given candidate item, a second layer will emphasis the item features that most contributed to this similarity. Although DARIA will attempt to solve the same challenge as SARAH, it relies solely on the standard neural attention technique, while SARAH only applies the rising self-attention paradigm.

Section 4.1 will discuss the motivation and background for proposing our attention-based frameworks, that will be respectively developed in Sections 4.2-4.3. An empirical evaluation of the two methods will be further presented in Sections 4.4-4.5.

4.1 Motivation

While recommender systems based on latent factors, such as TCENR, are able to provide increasingly accurate recommendations, generating lists of items alone is often not sufficient. Users may disagree with the given suggestions and over time lose their trust in the system [21]. Explainable recommenders therefore attempt to confront this issue by producing interpretable reasoning to justify its selections. These systems are then required to not only learn a model that is accurate, but to also produce explainable outputs. This requirement, however, is in conflict with the concept of latent factors, where machine-generated scalars represent each component.

The rising popularity of neural attention had provides a promising ground to integrate the seamlessly contradicting concepts of explain-ability and latent features, as demonstrated in previous work [9, 17, 62]. By applying attention techniques, recommender systems can allow different inputs to impact predictions according to their level of significance, and in addition to report these importance distributions to the end user as an explanation.

Our proposed methods, SARAH and DARIA, extend the use of neural attention in recommender systems and present two different and novel approaches. However, they both exploit the availability of user past activities and item features to allow flexible frameworks. Due to the extensive use of recommender systems in e-commerce websites and the privacy requirements towards users' data, practitioners usually have access to only limited information about the user, but often have a vast amount of items' attributes. Therefore, in constructing our models we choose to focus on historical data available in the system as the users' input, while the widely available and task-specific attributes will be used to describe items. Moreover, the use of attributes allows both SARAH and DARIA to alleviate the item cold-start problem.

Although SARAH and DARIA attempt to solve a similar task using the same inputs and generating equivalent explanations, they differ in their implementation. Comparing these two methods allows us to better analyze neural attention and self-attention for the task of item recommendation, in a way that has not been done before, to the best of our knowledge. The proposed frameworks are further evaluated over four datasets in varying scenarios, by utilizing the Yelp dataset for reviews over locations and the Amazon Electronics, Movies and Home data, that consist of product reviews in an e-commerce setting.

The main contributions of this chapter are summarized as follows:

1. We propose a novel idea of stacking two attention layers over items and their features, in order to best represent the user-item interaction.

2. To the best of our knowledge, we are the first to introduce a RS using self-attention to model user history and item attributes. This presents a broad range of future possibilities to improve predictions and explain-ability.

3. We conduct solid experiments on datasets in different fields, showing that our framework is able to provide superior results compared to six diverse baselines. We further illustrate the ability of SARAH to provide explanations in the form of case studies, as well as the contribution of self-attention to its output.

A list of notations used throughout this chapter is provided in Table 4.1.1.

Parameter	**Description**
f	Number of item features
k	Number of users' past items
Z_i	Latent embeddings of item i's features
e_i, E_u	Latent embedding of item i and user u's past items
V_i	Modified representation of item i's features
H_u	Contextual representation of user u's past items
α_i	Self attention scores over item i's features
α_u	Self attention scores over user u's past items
q_i, p_u	Final representations for item i and user u
α_{ui}	Attention scores over user u's past items with regard to i
r	Number of most relevant past items of u
$V_{j'}^{ui}$	Modified representation of features for a relevant past item j'
$G_{j'}^{ui}$	Attention matrix for target item i and past item j' features
$\gamma_{j'}^{ui}$	Weighted representation of relevant past item j'
Q^{ui}	Weighted representation of user u's relevant past items
$\hat{y}$	Final rating prediction

Table 4.1.1: Parameter notations used in Chapter 4

4.2 Self-Attention With Attributes And History

In this section we present our proposed framework, Self-Attention Recommender based on Attributes And History (SARAH).

Implemented using paradigms of deep neural networks, SARAH utilizes each item's input features to generate an accurate representation, where a novel self-attention layer determines what features should have higher impact. Likewise, a second self-attention layer specifies the most characterizing items each user has previously interacted with, used to construct the user latent vector. Our proposed method combines the two representations to result in an implicit prediction to whether a given user will be interested in the candidate item or not, while sharing the self-attention importance weights as explanatory factors.

To develop our framework, we will first describe the item self-attention network, resulting in a latent vector using an item and its features as input. Then, we will introduce the user self-attention component that transforms the user's last interacted items to a latent representation, along with her contextual data. The two resulting representations describe the candidate item and target user, respectively, while giving relative weights to their components based on their importance. Finally, we define the fusion of the two vectors as a scoring function along with the optimization process of the combined network. The complete architecture of our framework is shown in Figure 4.2.1, where each interaction is fed as a triplet of item i, the set of its f features, s_i, and a set of the last k items a user u interacted with, c_u.

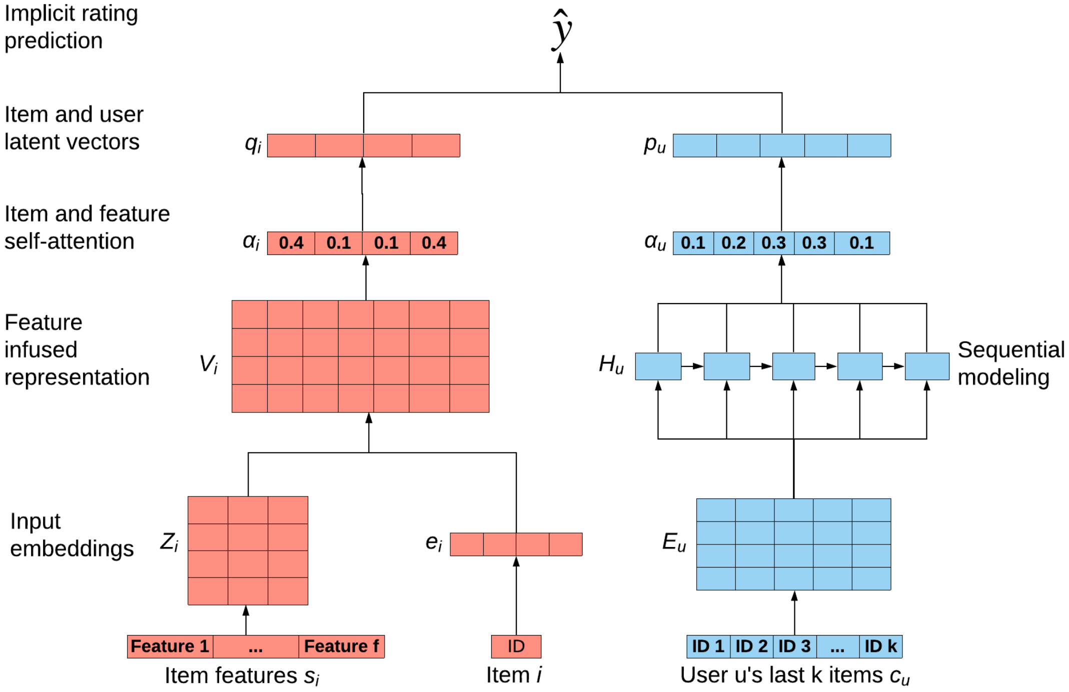


Figure 4.2.1: SARAH Framework

A sample recommendation scenario using SARAH is provided in Figure 4.2.2, where a user is represented with four past electronics products and items by different related features. Demonstrated in the user attention weights, α_u, SARAH identifies the laptop our user recently bought as the item that best capture her preferences. This is due to the overall similarity of the laptop to other items such as a tablet and a mouse. In addition, we note that the mouse representation is highly impacted by the related purchase of the laptop, as determined by the RNN component. To represent our candidate item, a smart-phone, SARAH identifies the screen and brand as the more important features, and as a result a high recommendation probability of 0.8 is produced.

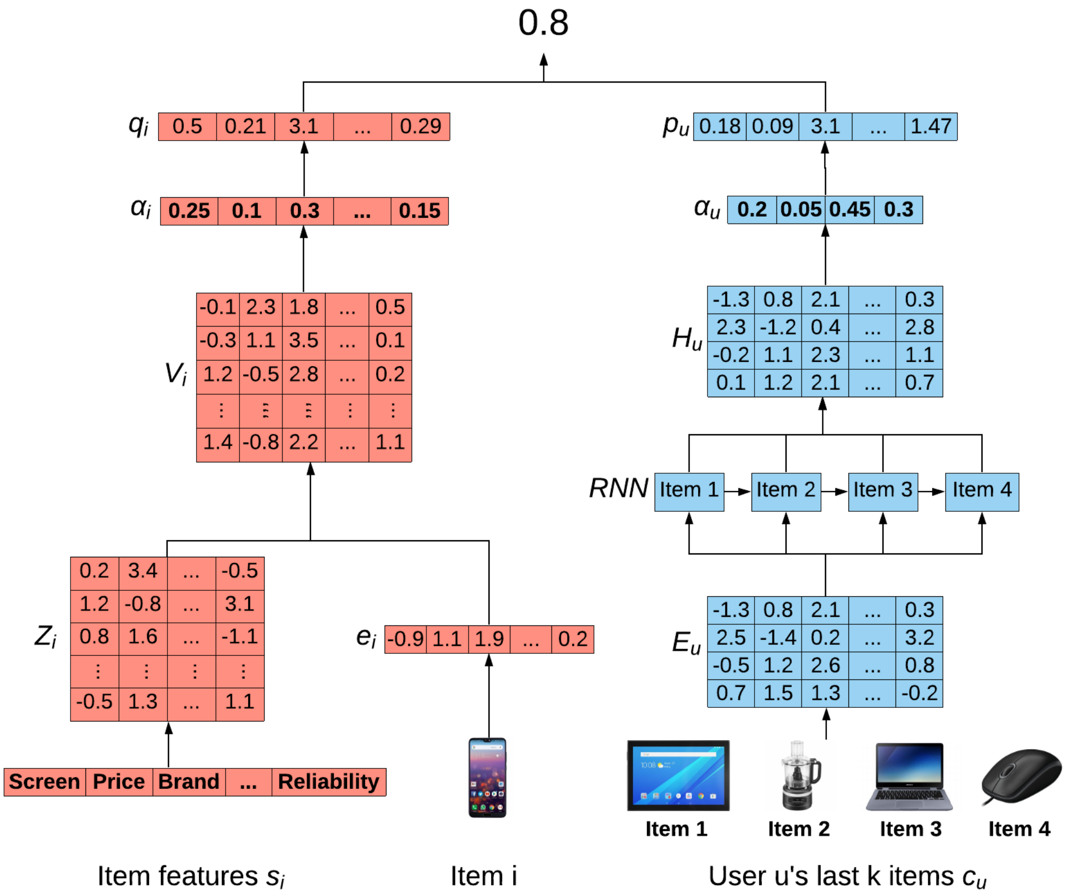


Figure 4.2.2: SARAH Sample Scenario

4.2.1 Item Self-Attention

The item self-attention component's goal is to produce an interpretable representation for a candidate item, where each is composed of a given set of attributes. These can be derived from various inputs, such as textual reviews, description, categories, geo-spatial location, etc. We claim that while all items are comprised of the same features, the importance of each feature differs. For example, being family friendly is an important attribute to consider when watching an animated show, but less so when choosing a horror film. Using self-attention we represent each item as the weighted distribution of its features, where stronger weights indicate relative importance. This allows SARAH to learn an item representation regardless of the target user, where the weights are interpretable and point towards the most relevant features.

While each item is composed of features, some can be shared between very different items. For example, two movies can be of the same genre and share participating actors, but one is more popular than the other due to reasons that are not captured by the set of given attributes. Therefore, in addition to features we also use the item's historical information, described as its embedding, to construct the final representing vector. As shown on the left side of Figure 4.2.1, the two sets of inputs are fed to the network as plain item and feature ids, implemented as one-hot encoding vectors, where 1 indicates the current item or feature and 0 the rest. Two embedding functions are used to transform the inputs into representing vectors, $e_i \in \mathbf{R}^d$ for item i and $Z_i = \{z_{i0}, z_{i1}, z_{i2}, ..., z_{if}\}$ for its features, where $z_{il} \in \mathbf{R}^{\tilde{d}}$ denotes the embedding vector for item i's l-th attribute. These embeddings result in two descriptions for the same item, one using a standard latent vector and the other using its features which are interpretable.

To combine the two representations in a way that allows the same attribute to have different effects on various items but still enable to isolate the different features, we let the model learn their combination by employing a fully connected nonlinear layer over the embeddings' concatenation:

$$v_{il} = ReLU(w_v[e_i, z_{il}] + b_v) \ , \tag{4.2.1}$$

where $ReLU$ is a nonlinear activation function, w_v is the weight vector and b_v the bias. $V_i = [v_{i0}, v_{i1}, ..., v_{if}]$ is the layer's output denoting the feature-infused representation of item i, where f is the number of features. By combining the item attributes with its past interactions, we allow the model to potentially represent items with little or no history, making SARAH relatively resilient to the item cold-start problem, as will be demonstrated in Section 4.5.

We then employ a self-attention network to transform V_i into an interpretable vector, comprised of each feature's importance. By feeding it to a two-layer perceptron, V_i is reduced to a concise representation of a single attention score for each input feature:

$$a_i = w_2(tanh(W_1 V_i) + b_1) + b_2 \ , \tag{4.2.2}$$

where $W_1 \in \mathbf{R}^{f \times d}$ and $w_2 \in \mathbf{R}^f$ are the respective weight matrix and vector, $b_1 \in \mathbf{R}^d$, $b_2 \in \mathbf{R}$ are the layer's bias terms and $tanh$ is the hyperbolic tangent function. To increase the self-attention model's ability to generalize and reduce exaggerated importance towards any single feature, we introduce a dropout function over V_i.

The resulting vector a_i can be seen as the importance weights of each feature over the item i, however for it to be utilized as a weight distribution we feed it to a softmax function:

$$\alpha_i = softmax(a_i) \ . \tag{4.2.3}$$

α_i will then be employed as a weighting vector, responsible for constructing a one-dimensional representation for the given item i. This is achieved by performing a weighted average over the item features' latent factors according to their relative importance:

$$q_i = \sum_{l=0}^{f} \alpha_{il} v_{il} \ , \tag{4.2.4}$$

where q_i is the item i's latent vector. In addition, α_i will act as an explaining factor to the end user, where the features of a recommended item i are ranked based on their attention scores. Examples of the vector's values are presented in the left part of Figure 4.2.1.

4.2.2 User Self-Attention

Similar to the item component, the user self-attention network's objective is to provide an accurate, yet interpretable, representation for every user u. We define hyperparameter k as the number of last items used to represent each user, and provide the flexibility to explore different time spans. The list of k input items is denoted as c_u and demonstrated in the right side of Figure 4.2.1. Unlike the previously described item self-attention sub-network, we do not add an additional user embedding based on the individual user id, since it will be equivalently comprised of the past interactions, but in a non-interpretable method. Therefore, to describe a user, we use the embedding $E_u = \{e_{u1}, e_{u2}, e_{u3}, ..., e_{uk}\}$, where e_{uj} is the same item embedding presented in subsection 4.2.1, e_i, but for u's j-th item.

Since users' actions are frequently effected by previous activities, we attempt to enrich her items' embeddings with the contextual information found within the sequence of interactions. More specifically, we employ a RNN over the k embeddings, so the representation of each item will include that of its predecessors to some degree. Similarly to TCENR, we follow the findings of previous works [3], and adopt GRU as our recurrent technique. Based on Eq. 2.2.6, each user past item uj can be described by a new vector combining relevant contextual data $h_uj \in \mathbf{R}^d$.

Although a user can be defined as the set of items she previously interacted with, some items are more important in this sequence than others. For instance, a user who frequently watch action movies should be mostly represented by the latent features of this genre and not by those of a drama movie she watched once. To this end, we introduce a self-attention network over the user's k items, similarly to the layers presented in subsection 4.2.1:

$$\begin{aligned} a_u &= w_4(tanh(W_3 H_u) + b_3) + b_4 \ , \qquad (4.2.5) \\ \alpha_u &= softmax(a_u) \ , \end{aligned}$$

where the input $H_u \in \mathbf{R}^{k \times d}$ is the set of k items' sequential embeddings $\{h_{u0}, ..., h_{uk}\}$. The user self-attention network is learned using the weights W_3, w_4 and the bias terms b_3, b_4. The resulting vector $\alpha_u \in \mathbf{R}^k$ is the importance distribution of the k items for the user u, used to provide recommendation reasoning to the end user, by presenting her with the information towards how she is perceived by the system. Potential values of α_u are given as example in the right side of Figure 4.2.1.

To retrieve a meaningful representation to be modeled along with the candidate item i, we weight each of user u's k sequential item embeddings using the attention importance vector, p_u:

$$p_u = \sum_{j=0}^{k} \alpha_{uj} H_{uj} \ , \qquad (4.2.6)$$

generating in a vector that emphasizes the latent features of the most relevant historical interaction of the user.

4.2.3 Rating Prediction

In order to combine the user and item weighted representations and to result in a prediction, we use the classic dot product operation. By doing so, we avoid any excessive alteration to the user and item interpretable vectors and keep the weighted distributions relevant to explain the given prediction. To allow the framework to be applied to a wide variety of recommendation scenarios, we adapt every sample to the implicit rating problem, where 1 denotes an interacted item and 0 items the user have no interest in. The resulting output is therefore transformed to be in the range of [0-1] using the sigmoid function:

$$\hat{y} = \sigma(q_i^T \cdot p_u) \ . \tag{4.2.7}$$

We use negative sampling to generate instances of 0-class ratings from the dataset, defined as $Y^{\text{-}}$ to differ from the training positive set Y.

Due to the use of implicit ratings and the sigmoid function in prediction, the learning problem can be now viewed as a task of binary classification. The model's output probability can be then defined as:

$$p(Y, Y^{\text{-}} \mid E^i, Z^s, \Theta_f) = \prod_{(u,i)\in Y} \hat{y}_{ui} \prod_{(u,j)\in Y^{\text{-}}} (1 - \hat{y}_{uj}) \ , \tag{4.2.8}$$

where (u, i) is a positive interaction, (u, j) a negative sampled instance, $E^i \in \mathbf{R}^{m\times d}$ and $Z^s \in \mathbf{R}^{f\times\tilde{d}}$ are the embeddings of all m items and f features,respectively, and Θ_f denotes the model's parameters.

Similar to Eq. 3.3.2, we take the negative log-likelihood of Eq. 4.2.8 and employ binary cross-entropy as the model's loss function.

4.3 Dual Attention Recommender

By applying independent self-attention layers to represent users and items, SARAH assumes that the importance of previous user activities is not effected by the recommended item in question. However, this is not always the case. For example, when a user who rarely watches horror movies considers whether to watch a new horror movie, the decision might be impacted by the similarity to the few horror movies she previously watched and liked, rather than to action movies she most frequently watches. We wish to further investigate the recommendation capability of SARAH by proposing a second model that only employs the standard neural attention and considers the current interaction when learning user and item representations.

DARIA is therefore a proposed alternative to SARAH that attempts to describe a given user-item instance by focusing on the most correlating components of its two inputs. This method assumes that although a user has previously interacted with many items, not all past feedbacks are relevant in predicting her interest towards a given item. Extending the previous example, DARIA will compare the importance of each past movie to the recommended horror movie in order to determine its impact on the user's representation within the system, making it specific to the predicted interaction. We therefore follow previous work [11, 60] and employ neural attention to model this behavior, while disregarding irrelevant past items when determining whether to recommend a given item.

However, we claim that for the task of explain-ability it is insufficient to only provide what past actions contributed to the recommendation. It is more intuitive and general to further explain what features in these relevant past interactions have had the highest impact on this reported contribution. We will therefore extract only the most relevant past items, as determined by the previously described attention network, and apply a second attention layer over their attributes, compared to those of the recommend item. This layer's output will then be the importance distribution of each relevant past item's features compared to these of the recommended item. It will be further used to construct the concise representation of the user-item interaction and to generate implicit predictions in turn. Similar to SARAH, the model yields the importance distributions over items and features, learned by the attention layers, as explanatory factors to the end user.

By implementing DARIA we hope to shed light on the impact of self-attention and neural attention networks on deep recommender systems. The proposed method is illustrated in Figure 4.3.1. Section 4.3.1 describes the steps taken to describe users in DARIA with relevant past items, while Section 4.3.2 outlines the use of features to generate combined representations and predictions. Finally Section 4.3.3 summarizes DARIA along with a comparison to SARAH.

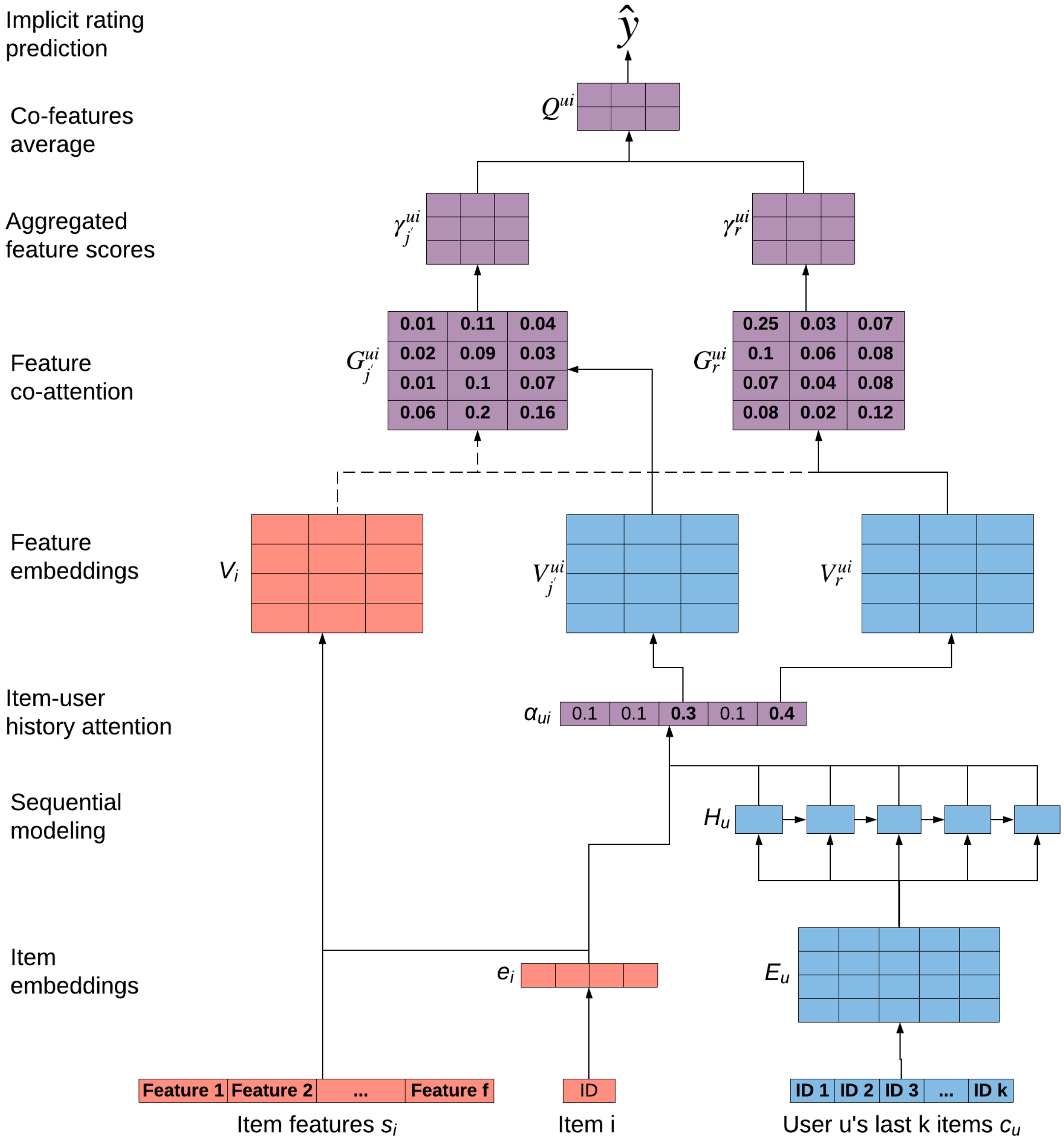


Figure 4.3.1: DARIA Framework

4.3.1 Past Items Importance

DARIA's first goal is to identify the most relevant past actions of user u with regard to the candidate item i and ignore all other past feedbacks. To do so, we adopt the same embeddings as in SARAH, e_i and E_u to represent the recommended item and target user, respectively. Moreover, we apply the same GRU layer as in Eq. 2.2.6 to add sequential context to E_u, resulting $h_{uj} \in H_u$ as the contextual representation of user u's past feedback for item j.

A neural attention layer is then applied to identify the most relevant $r < k$ items in user u's history with regard to i. We first generate a score for each of the k items using dot-product between its embedding's latent features and those of e_i:

$$\begin{aligned} a_{uij} &= h_{uj} \cdot e_i \ , \\ a_{ui} &= [a_{ui0}, a_{ui1}, ..., a_{uik}] \ , \\ \alpha_{ui} &= softmax(a_{ui}) \ , \end{aligned} \tag{4.3.1}$$

where j is one of the k items in user u history, a_{ui} is the set of u's attention scores between i and each past item and α_{ui} is an importance distribution of the k items. Unlike the relatively static user attention scores generated by SARAH, α_{ui} is greatly effected by the current recommended item i, and will likely be different for another recommended item i'.

While α_{ui} will be provided to the user as an explanatory factor, reporting how related each of the items she previously liked to the recommended item i, we wish to further investigate what features impact the items' significance levels. However, there is no need to further explore the effect different features have for irrelevant past items. To this end, we follow Eq. 2.2.8 and use max-pooling to keep only the r items with the highest attention weights in α_{ui}, denoted as $\beta^{ui} = [ui_0, ui_1, ..., ui_r]$, where the remaining $k-r$ past items of user u are discarded. Each item $\beta^{ui}_{j'}$ is then transformed into its features embedding, denoted as: $E^{ui}_{j'} \in \mathbf{R}^{f \times d'}$, where f is the number of item features, and d' is the feature embedding's dimensionality.

To determine what features made each of the r items more related to i than the other $k-r$ items, we are required to employ the attributes of i. Illustrated in the left side of Figure 4.3.1, the recommended item's f features are given as input, denoted as s_i, and further transformed to their respective embedding $Z_i = \{z_{i0}, ..., z_{if}\}$. Since we still wish to have an item-specific feature representation, we follow Eq. 4.2.1 to transform Z_s and $E^{ui}_{j'}$ to V_i and $V^{ui}_{j'}$, respectively:

$$\begin{aligned} v^{ui}_{j'l} &= ReLU(w_v[e^{ui}_{j'}, z_{j'l}] + b_v) \ , \qquad (4.3.2) \\ V^{ui}_{j'} &= [v^{ui}_{j'0}, v^{ui}_{j'1}, ..., v^{ui}_{j'f}] \ . \end{aligned}$$

4.3.2 Item Features Attention

A second attention layer will compare the features of each past item j' with those of item i. However, unlike the attention technique previously used in Eq. 4.3.1, the design of our model allows us to compare all attribute combinations between the two items. We can therefore evaluate the relevance of each feature for j' with all the other features of i:

$$G^{ui}_{j'} = sofmtax(V_i \times V^{ui}_{j'}) \ , \qquad (4.3.3)$$

where × denotes matrix multiplication and $G^{ui}_{j'} \in \mathbf{R}^{f \times f}$ is a two-dimensional matrix where the f features of past item j' can be depicted as columns while those of the recommended item i as rows.

To describe the overall user-item interaction by past items' feature importance compared to i, DARIA then transforms the scores of an item j' l-th feature, j'_l, by employing a simple summation over each of the matrix $G^{ui}_{j'}$ columns:

$$\gamma^{ui}_{j'l} = \sum_{t=0}^{f} G^{ui}_{j'lt} \ , \tag{4.3.4}$$

where t is a feature of candidate item i as well as a row in $G^{ui}_{j'}$, while l is a feature of u's past item j' and a column in the attention matrix. By summing all rows of a feature l, we are able to retrieve its aggregated attention score with regard to all features of item i.

In addition, $\gamma^{ui}_{j'l}$ will act as an explanatory factor given to the user. It provides an insight towards what features make a past item j' relevant in estimating the preference towards item i. Similar to 4.2.4, $\gamma^{ui}_{j'l}$ will be further utilized as a weighting vector to generate a representation of user u's past item j' according to its attention scores with the candidate item i:

$$q^{ui}_{j'} = \sum_{l=0}^{f} \gamma^{ui}_{j'l} \cdot v^{ui}_{j'l} \ , \tag{4.3.5}$$

where $v^{ui} \in \mathbf{R}^{r \times f \times d'}$ is user u's embedding of the r most relevant past items with regard to i, $q^{ui} \in \mathbf{R}^{r \times d'}$ is the past items' feature-based latent vector, f the number of item features and d' is each feature latent dimensionality.

We finally produce a prediction by combining the factors of most relevant r items of user u. DARIA creates a concise representation of the user-item interaction by concatenating the different q^{ui}_{*} vectors, such that: $Q^{ui} = [q^{ui}_1, ..., q^{ui}_{j'}, ..., q^{ui}_r]$. Since there is a strong connection between the latent factors of some past items, we feed Q^{ui} to a single-layered neural network:

$$\hat{y} = \sigma(w_y \times Q^{ui} + b_y) \ , \tag{4.3.6}$$

where $w_y \in \mathbf{R}^r$ and b_y are the weight vector and bias term, respectively. We apply the sigmoid function as the network's non-linearity to produce an implicit prediction, used as DARIA's output. The model's training follows the same steps as SARAH, described in detail in Section 4.2.3.

4.3.3 Comparison to SARAH

Although there are numerous similarities between SARAH and DARIA, including their expected inputs, structure of generated outputs and the use of recurrent neural network to model user context, the two methods are fundamentally different in their implementations. The main distinction lies in the concept behind each method. While SARAH assumes that users and items should have similar latent representations regardless of the current recommendation, DARIA attempts to learn the combined interaction of each user and item tuple. The user representation in DARIA is therefore heavily effected by the items that are closely related to the recommended item, as determined by the neural attention layer, and will be significantly different for multiple recommended items. While in SARAH a user is represented by all her k past items in a different degree of importance, in DARIA we only include the $r < k$ past items that are most similar to the candidate item. For example, a certain movie that is deemed as the most representative of the user by SARAH, might be filtered out and not have any impact on the same user representation in DARIA.

In addition, even though the two methods generate an importance distribution over item features as an explanatory factor, the same data has different meaning across our frameworks. While in SARAH features are applied to define each item in an interpretable way, DARIA adopts features to explain why a past item is considered similar to the recommended item. This is then aggregated to describe the features' importance of the user past items with regard to the recommended item. Finally, while the recommendation generated by SARAH is achieved by measuring the similarity between user and item representations using dot product, DARIA applies a neural network over the aggregated feature importance.

By Comparing the two methods, as done in Section 4.5, we attempt to examine two opposing techniques based on neural attention and determine whether we should describe users and items independently to produce accurate predictions, or alternately generate recommendations by learning the user and item interactions.

4.4 EXPERIMENTAL SETTINGS

4.4.1 Datasets

To evaluate our proposed frameworks in diverse recommendation settings, we adopt datasets from different fields.

Amazon. We chose to employ the Electronics, Movies and Home datasets provided by [40]. The three datasets feature over 16 million reviews combined, written about products purchased in the popular e-commerce website. Each of the given datasets is pre-filtered to include only users and items with more than 5 reviews.

Yelp[1]. The public dataset provided by Yelp presents more than 5.9 million reviews by users of locations in the successful location-based social network. While Amazon's datasets are dedicated each to a single area, the one by Yelp ranges over various location types, such as hotels, restaurants, grocery stores, gas stations and more. To allow items to differ by relevant features (i.e. preventing the existence of many disjoint attributes), we limit the data to focus on food-related locations (restaurants, bars, fast food, e.g.), using dataset-provided categories.

In each scenario, we divide ratings into training, validation and test sets using a time-based split, where each user's last interaction constructs the test set and the one before is used for validation. We further filter out users with less than 15 reviews, to allow sufficient historical records to represent each user. For example, if a user input consists of her last 10 interactions, no less than 3 samples are left to construct

[1] https://www.yelp.com/dataset/challenge

the training set while the remaining 2 define the validation and test sets. To provide negative instances we sample 3 zero-rated samples for each positive rating over the training and validation sets. Statistics regarding the final datasets excluding negative samples are presented in Table 4.4.1.

Dataset	**Yelp**	**Electronics**	**Movies**	**Home**
Users	43,702	14,346	19,566	3,725
Items	51,068	34,389	30,761	14,008
Positive Ratings	1,681,773	364,899	852,091	90,848
Rating Sparsity	99.925%	99.926%	99.858%	99.826%
Avg Samples/User	38.48	25.43	43.55	24.39
Avg Samples/Item	32.96	10.74	27.72	6.72

Table 4.4.1: Datasets' Statistics

Due to the availability of textual inputs in many recommendation scenarios and to allow a similar ground for comparison to other methods, we chose topics derived from user reviews as the item features in the model's and baselines' experiments. To learn the topics, we first feed the reviews as bag-of-words to a latent dirichlet allocation (LDA) model [6]. Then we learn word embeddings using a simple CNN network over our training data. Each topic is finally represented by the product of its distribution over the semantic word embeddings. Although we chose to focus on topics, the proposed framework can easily be fed with various other features, such as categories, location and more, given their latent vectors. Furthermore, the model is not limited to topics derived from LDA, but can equally apply other textual embedding methods.

4.4.2 Evaluation Metrics

To evaluate the recommendation performance of our proposed models and baselines, we focus on its ability to provide relevant top-10 suggestion to the user. For every item the user had reviewed in the test set, we sample 99 negative items she has not encountered with and provide these 100 interactions to each model. Since the test set is comprised of one positive instance for each user, the final set will hold N positive items and $99 * N$ negative samples. Towards the task of model evaluation, we follow previous work [11, 16, 20, 62] and adopt the popular HR@10 and NDCG@10 metrics to measure the performance in predicting top-10 items:

- **Hit-Ratio (HR)**: Returns the percentage of users that had a relevant item in their top-10 suggestions. Defined as:

  $$HR@10 = \frac{1}{N} \sum_{u} \mathbf{I}(|R^u \cap T^u|) \ , \tag{4.4.1}$$

  where R^u and T^u are user u's top 10 predicted and ground truth items, respectively, N is the number of users and $\mathbf{I}$ is a function that returns 1 when its input is positive, and 0 otherwise.

- **Normalized Discounted Cumulative Gain (NDCG)**: A measure that assigns weights based on an item's position within the top 10 suggestions:

  $$NDCG@10 = \frac{1}{Z} \sum_{j=1}^{10} \frac{2^{rel_j} - 1}{log_2(j+1)} \ , \tag{4.4.2}$$

  where rel_j denotes the relevancy of an item in position j and Z is a normalizing factor.

We report the average scores of the aforementioned metrics over all users in the test set. Parameter tuning in the validation set is done by measuring the ROC score.

4.4.3 Baselines

To evaluate the performance of SARAH and DARIA over the four datasets, we compare its results to six key baselines, ranging from classic MF and probabilistic methods to deep learning-based models:

- **CTR** [58]: Collaborative Topic Regression. A probabilistic model that estimates an item latent vector using its topics, derived from LDA.
- **NMF** [36]: Non-negative Matrix Factorization, a CF method that takes only the rating matrix as input.
- **ConvMF** [25]: Convolutional Matrix Factorization. A strong baseline for recommendations using textual input. This probabilistic model utilizes CNN to learn an item latent vector.
- **NeuMF** [20]: Neural Matrix Factorization. A state-of-the-art framework for recommendation using only past feedbacks. NeuMF combines a generalization of MF with MLP over tuples of users and items.
- **RUM** [11]: Recommender system with external User Memory networks. A trending deep model that employs attention to identify the most relevant items and features in the target user history compared to the candidate item.
- **CMN** [16]: Collaborative Memory Networks. A novel method that adopts attention network and memory matrices to provide a neighborhood-based component in an hybrid RS.

By comparing our proposed frameworks to each of the six baselines, we measure their performance along with models that best represent different recommendation approaches. CTR [58] and NMF [36] are two well-known methods for classic collaborative filtering, while ConvMF [25] and NeuMF [20] are found to be highly effective in different scenarios. Furthermore, both CTR and ConvMF employ textual data to represent items, similar to SARAH and DARIA in our selected setting. By evaluating RUM [11] and CMN [16], two emerging approaches that utilize neural attention, we are able to analyze the impact of self-attention over the standard attention paradigm.

4.4.4 Parameters Settings

Our proposed models are implemented using Tensorflow[2], and released as open source[3]. As part of our application, all parameters are initialized to follow uniform distribution, and in order to avoid over-fitting when training the model an early stopping criteria is integrated. Items latent dimensionality and number of features are determined by grid search in the ranges of {4,8,16,32,64,128} and {10,30,50,70,90}, respectively. We further set the size of each user's input, k, to be in {4,6,8,10,12}. To evaluate ConvMF and pre-train the topics for our two frameworks, we utilize the word embeddings of GloVe [46], with dimensionality of 50. To optimize our framework, we apply a learning rate in the range of {0.001,0.005,0.01,0.05,0.1} over no more than 100 epochs with a batch size of 8,192 instances.

4.5 EXPERIMENTS AND EVALUATION

In this section we evaluate SARAH and DARIA by conducting a series of analyses to answer the following research questions:

1. **RQ1**: Are the proposed frameworks able to achieve competitive recommendation performance compared with state-of-the-art baselines?
2. **RQ2**: How different hyperparameter values effect the models' ability to learn and present explanations?
3. **RQ3**: Does our models produce relevant and understandable reasoning along with their provided suggestions?
4. **RQ4**: How does the suggested methods compare with the dedicated POI recommendation models presented in Chapter 3?

[2] https://www.tensorflow.org
[3] https://github.com/omer-tal/SARAH

Dataset	**Yelp**		**Electronics**		**Movies**		**Home**	
Metric	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10
CTR	0.362	0.202	0.462	0.236	0.43	0.242	0.348	0.159
	(129.74%)	(215.89%)	(49.60%)	(93.01%)	(73.70%)	(111.33%)	(62.29%)	(115.40%)
NMF	0.644	0.399	0.436	0.267	0.444	0.254	0.333	0.175
	(39.49%)	(62.85%)	(57.24%)	(69.19%)	(68.14%)	(100.98%)	(69.76%)	(95.47%)
ConvMF	0.468	0.269	0.341	0.201	0.329	0.185	0.261	0.141
	(96.57%)	(148.93%)	(102.27%)	(126.23%)	(126.93%)	(175.81%)	(116.25%)	(142.80%)
NeuMF	0.808	0.489	0.338	0.169	0.501	0.281	0.295	0.147
	(14.43%)	(37.70%)	(106.66%)	(173.71%)	(48.92%)	(81.68%)	(91.70%)	(131.92%)
RUM	0.750	0.459	0.438	0.270	0.518	0.319	0.323	0.168
	(21.51%)	(44.71%)	(57.35%)	(75.98%)	(44.06%)	(60.06%)	(75.21%)	(103.59%)
CMN	0.875	0.592	0.428	0.267	0.534	0.32	0.098	0.048
	(6.47%)	(14.56%)	(83.45%)	(93.08%)	(39.71%)	(59.61%)	(479.23%)	(606.96%)
DARIA	$0.892^{\star}$	$0.614^{\star}$	$0.637^{\star}$	$0.401^{\star}$	$0.678^{\star}$	$0.432^{\star}$	$0.548^{\star}$	$0.318^{\star}$
	(3.7%)	(9.9%)	(7.59%)	(12.34%)	(10.12%)	(18.08%)	(3.23%)	(7.62%)
SARAH	$\mathbf{0.925}^{\star}_{\diamond}$	$\mathbf{0.675}^{\star}_{\diamond}$	$\mathbf{0.686}^{\star}_{\diamond}$	$\mathbf{0.451}^{\star}_{\diamond}$	$\mathbf{0.746}^{\star}_{\diamond}$	$\mathbf{0.51}^{\star}_{\diamond}$	$\mathbf{0.565}^{\star}_{\diamond}$	$\mathbf{0.342}^{\star}_{\diamond}$

Table 4.5.1: Performance comparison between SARAH, DARIA and the six baselines. $\star$ and $\diamond$ indicate significant improvement over the baselines and DARIA, respectively, based on a 5-sample paired t-test at the 0.01 level. Relative improvement of SARAH compared to each model is given in brackets.

4.5.1 Overall Performance (RQ1)

First, we compare the prediction performance of the six baselines and our suggested models. Table 4.5.1 displays the comparisons *w.r.t.* HR@10 and NDCG@10 over the different datasets. Several observations can be derived:

- Analyzing the results over the more balanced datasets, Yelp and Movies, demonstrates the strengths of deep learning based methods, as NeuMF, RUM, CMN, DARIA and SARAH are able to provide significantly better recommendations compared to CTR and NMF.

- The two methods based on textual input alone are found to be insufficient in modeling the user-item interaction. Both the probabilistic method CTR and CNN-based ConvMF achieve the lowest scores over the Yelp and Movies data. It may be due to the use of only 50 units to represent words' embeddings, but the results hints that while textual data is a valuable input, it shouldn't be solely relied upon to learn users' and items' representations.

- There is a large discrepancy when evaluating the models across the different datasets. First, the results obtained over the Yelp data are significantly higher compared to the Electronics and Home datasets for all methods. In addition, the trends from the Yelp and Movies results are not kept. In terms of HR@10 and NDCG@10, CTR and NMF achieve similar results to RUM and CMN over the Electronics data, while all four baselines outperform ConvMF and NeuMF. Furthermore, the best performing baseline in the Yelp dataset, CMN, results in the lowest scores over the Home data. This change can be explained by the analysis of the datasets' statistics, shown in Table 4.4.1. While the Home dataset is the most dense, and the rating matrix sparsity rate is similar between the Yelp and Electronics data, the datasets mainly differ by the average samples per user and especially per item. In the Electronics, and most notably the Home data, the insufficient number of samples for each item prevents from deep models, such as ConvMF and NeuMF, to learn relevant representations. However, methods based on attention networks (e.g. RUM) are seem to be more resilient to this problem. This, and the fact that each item is represented by its features as well as its embedding, allows both DARIA and SARAH to surpass all other methods in this scenario.

- Comparing our two proposed frameworks, it is clear that SARAH consecutively achieves more accurate performance over DARIA. Since in both cases users and items are identified by identical inputs, it is clear the difference lies in the model itself. While it is more simple, the use of self-attention allows SARAH to learn separate representations with no regard to the given instance and still yield better item recommendations.

- Overall, SARAH outperforms the evaluated models across all datasets, as the use of features and self-attention allows it to be less prone to the item cold start problem, while utilizing RNN and pre-learned embeddings results in competitive performance in a more dense scenario. Table 4.5.1 further demonstrates the improvement is statistically significant using a paired t-test over five independent evaluations where $p < 0.01$.

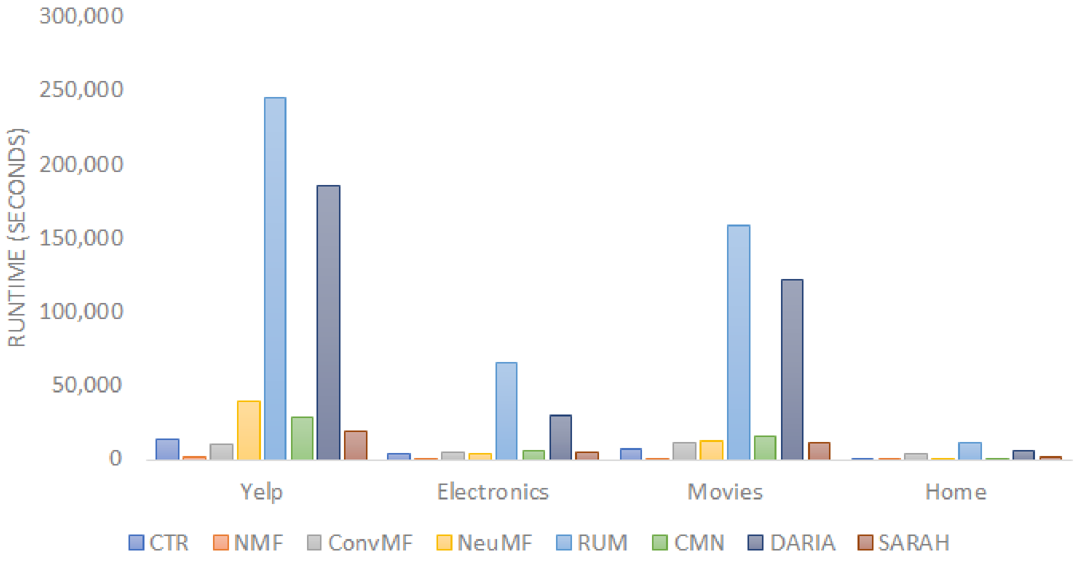


Figure 4.5.1: Runtime (seconds) of all models on the four datasets

In addition, we report the training efficiency of SARAH and DARIA, defined as the time required to train each model and reported in Figure 4.5.1. As demonstrated by the provided analysis, SARAH is relatively efficient compared to most baselines, including NeuMF, ConvMF and CMN. Moreover, unlike these methods SARAH allows the utilization of different input types with no additional cost to its efficiency. DARIA, in comparison, is found to be more complex in all scenarios due to its large number of parameters. However, it is more efficient than RUM while achieving better recommendation accuracy.

4.5.2 Hyperparameter Analysis (RQ2)

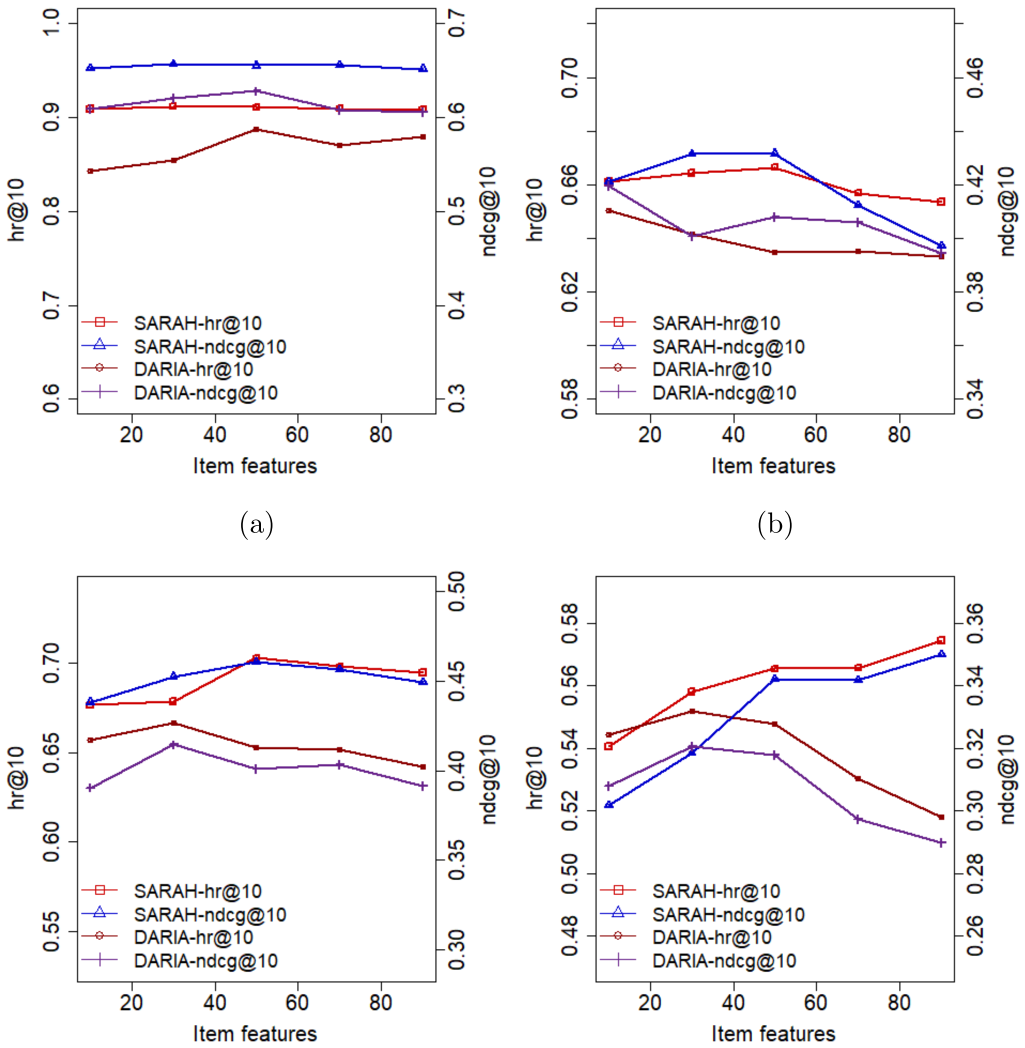
In this section we study the effects of various factors over the performance of SARAH and DARIA.

Number of item features

As the main component for describing items within the systems, the number of features is used to construct both our models' input and one of the two explainable factors derived from them. Figure 4.5.2 illustrates the influence of different sizes of input, ranging from 10 to 90 features, for SARAH and DARIA over all four datasets. To provide a fair comparison all features are from the same type, topics derived by LDA. Each item is therefore described by its positive or negative correspondence to each dataset-specific topic (e.g. price, genre, location or service).

Unlike the Yelp dataset which is relatively unaffected by this parameter, the three Amazon datasets provide contrasting trends. This phenomena could be explained by the relative heterogeneity of items sold on Amazon compared to the diversity in restaurants and bars featured on Yelp, which results in noisy features when representing items by too many attributes, but requires enough features to successfully distinguish the different items.

Analyzing the Electronics and Movies dataset results, demonstrated in 4.5.2b and 4.5.2c, shows that SARAH requires at least 30 topics to perform well, while a decline in performance is derived from additional features. DARIA displays a similar trend that is slightly skewed, achieving its best performance with only 10 topics over the Electronics data and 30 when applied in the Movies scenario. Comparing SARAH and DARIA over the Home dataset, however, exhibits opposite results. SARAH performs best as more features are added, while applying too many features in DARIA leads to a sharp decline. This contradiction may be due to the relatively small number of samples per item in the Home dataset. Given the lack of sufficient historical items' data, SARAH requires additional features to better represent an item. In contrast, features in DARIA are combined with the user input, allowing more data to be used when only few features are available.



(a)

(b)

(c)

(d)

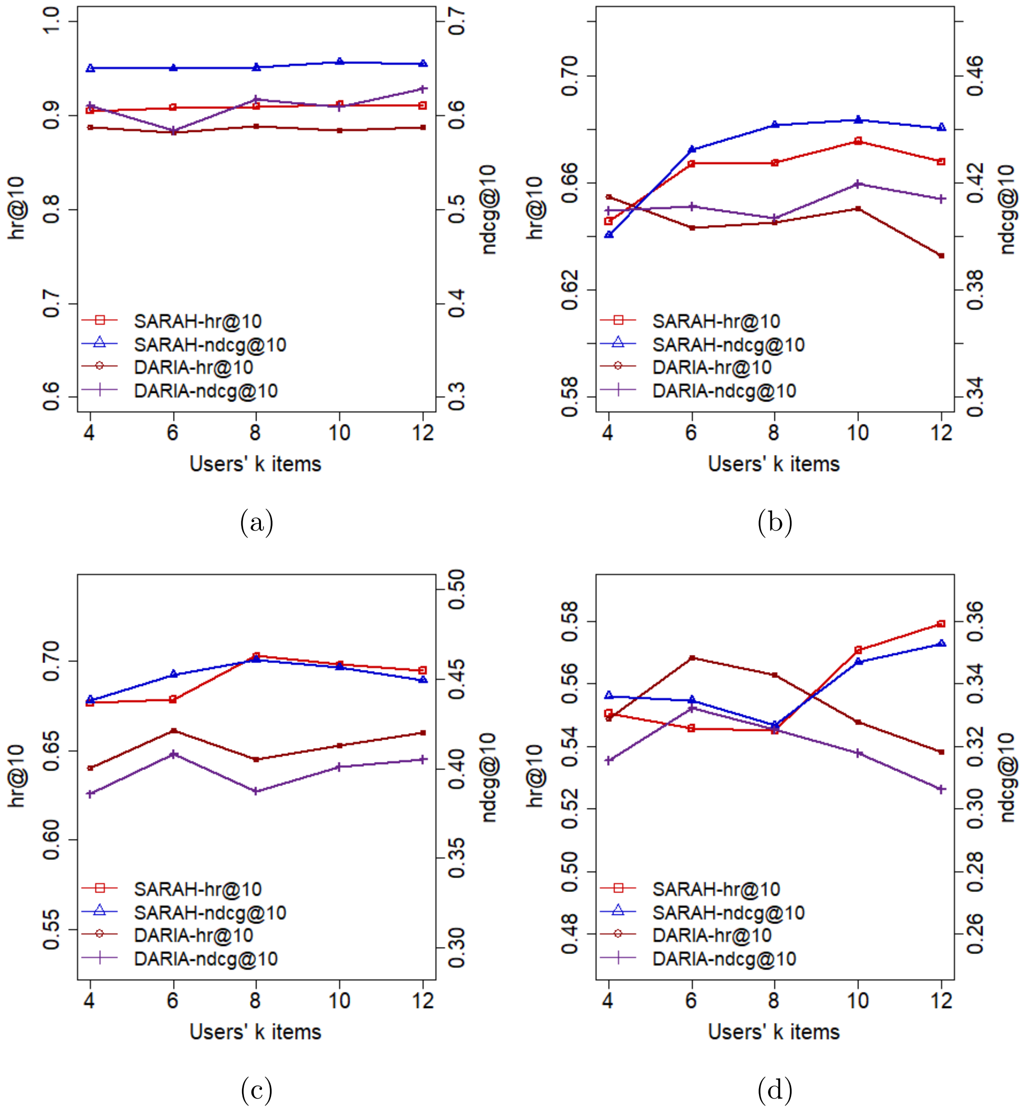
Figure 4.5.2: HR@10 and NDCG@10 over number of features for (a) the Yelp, (b) Electronics, (c) Movies and (d) Home datasets

User k items

Similar to the number of item features, different values of k effect each user's input, as this hyperparameter represents the number of items used to describe her in every interaction. Since the items defining a user cannot be used to optimize the model, this parameter further impact the training set size. As the minimal number of training features for every user is 13 items overall, we tested k values up to 12 items, resulting in users with one sample in the training, validation and test sets. The potential importance of this parameter can be seen in Figure 4.5.3, as more items improve the performance over the Electronics and Movies Amazon datasets, until reaching a specific number of items where the training size turns to be insufficient. For the Home dataset however, the reported effect of this hyperparameter is similar to the impact of the number of features hyperparameter. As higher k values improve SARAH's predictions over the Home data, having more than 6 items to describe each user results in sub-optimal performance for DARIA, possibly due to the higher volume of training instances required to optimize the model. On the Yelp dataset, where more samples per user are available on average, the two frameworks are more resilient to changes in this value.

Item embedding size

Employed both to construct the input for the item self-attention in SARAH and to represent items a user interacted with, the number of weights used to describe an item has the potential to impact SARAH's two self-attention components. This hyperparameter is equally significant for DARIA, as it determines which past items should be ignored. Moreover, the item embedding size also determines DARIA's RNN dimensionality, effecting the user representation as well as the item. We have tested this hyperparameter with values ranging between 4 and 128 units. As illustrated in Figure 4.5.4, increasing the number of variables representing items improves the two models' performance significantly for most datasets, until reaching an optimal value found in the range of 16-32 units, beyond which the prediction capability decreases. In fact, optimizing this hyperparameter has had the most apparent outcome over our frameworks' performance. While in SARAH its effect is relatively moderated due to the use of separate RNN dimensionality to represent users, describing items with too many latent weights significantly deteriorate DARIA's recommendation capabilities.



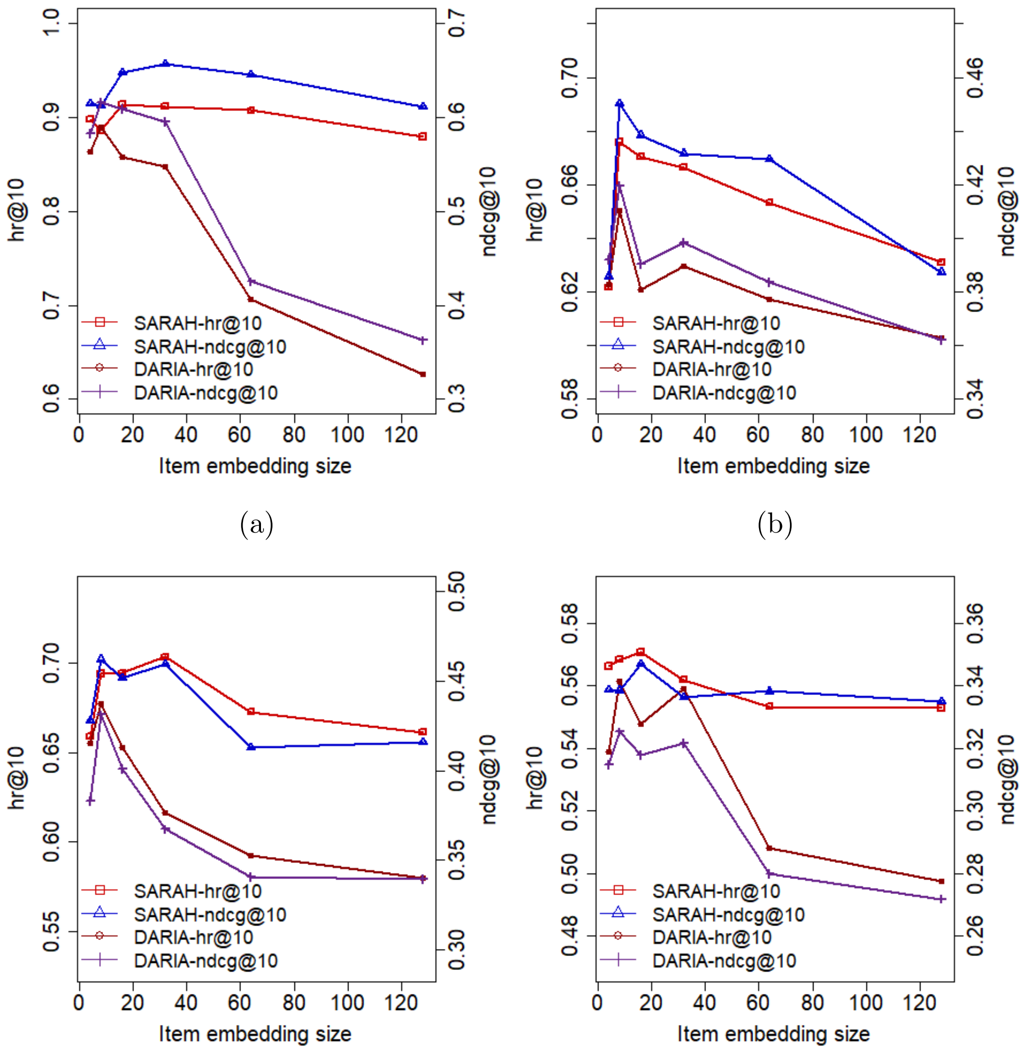
(a) (b)

(c) (d)

Figure 4.5.3: HR@10 and NDCG@10 over k for (a) the Yelp, (b) Electronics, (c) Movies and (d) Home datasets

RNN output weights

The number of units used to construct SARAH's RNN output layer influences each of the k user items' embedding size after being infused by sequential data. Along with impacting the model's ability to represent each item and its context for the target user, this hyperparameter also determines the final user embedding size used to estimate the output rating. We tested values in the range of 10 to 90 units, as



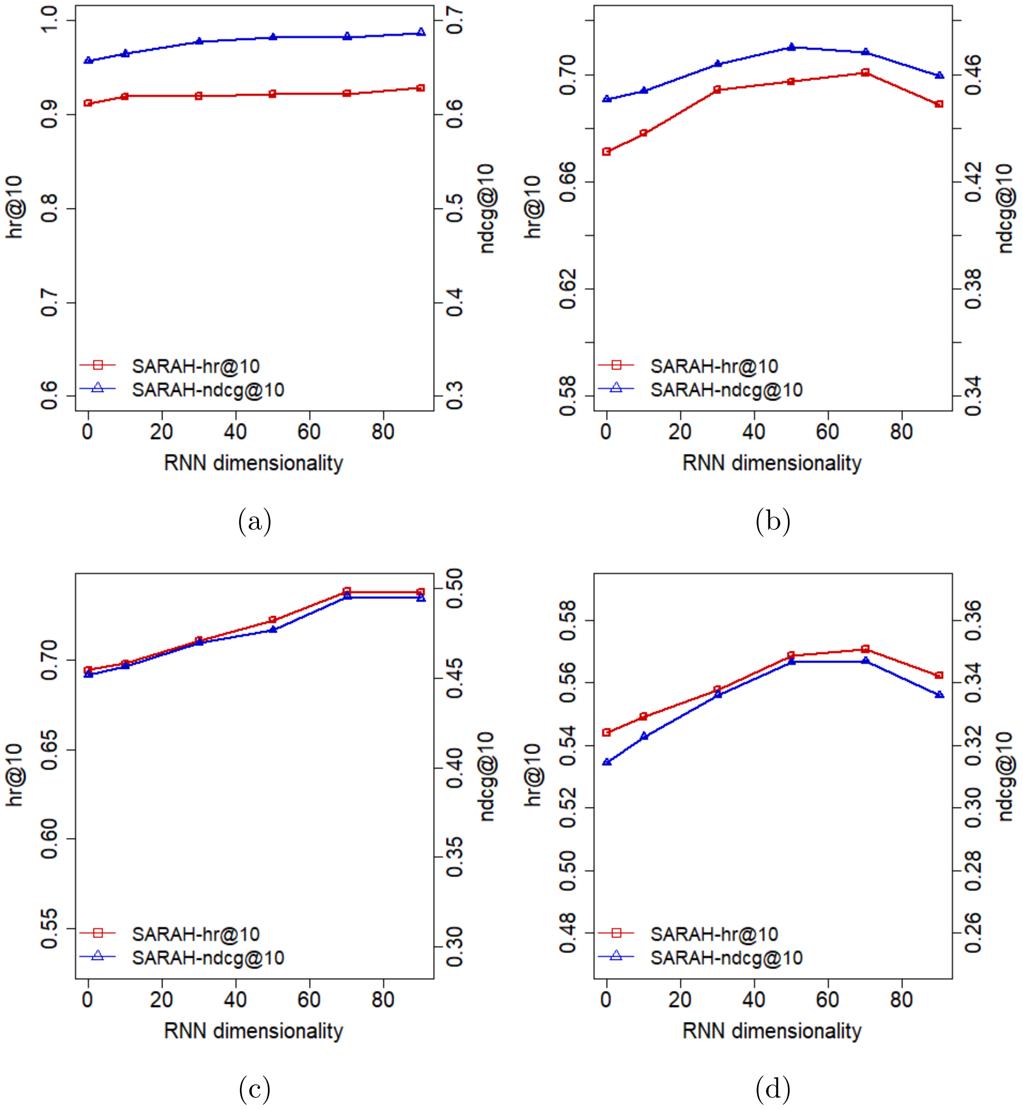
(a) (b)

(c) (d)

Figure 4.5.4: HR@10 and NDCG@10 over item embedding size for (a) the Yelp, (b) Electronics, (c) Movies and (d) Home datasets

well as when not using RNN (number of weights is 0), to determine the potential importance of sequentiality in the model. As illustrated in Figure 4.5.5, utilizing contextual data brings great improvement to the model and allows it to reach its optimal scores. While values above 80 units result in decreased performance in most reported scenarios, the positive impact of the RNN layer is clearly visible even when items are represented using only 10 hidden units. For DARIA, however, the RNN dimensionality cannot be modified as it has to be equal to the item embedding size.

Since this analysis was previously discussed, we did not include DARIA's performance in this evaluation.



(a)

(b)

(c)

(d)

Figure 4.5.5: HR@10 and NDCG@10 over RNN output layer size for (a) the Yelp, (b) Electronics, (c) Movies and (d) Home datasets

4.5.3 Case Studies (RQ3)

In this section we evaluate SARAH's capability in providing explainable representations for items and users. These can accompany the model's output either as weight distributions or as messages of the sort: *"We suggest you an item best described by* *the following features..."* and *"This item is recommended due to these items you previously liked..."*. The given explanations' relevance is visualized using heat maps over randomly sampled items and users from the Amazon Movies dataset.

Item Features Explanations

In adopting items' content for explanation, our proposed framework is able not only to report which items in the user history contributed to the recommendation, but also what features of the item define it best, allowing the user to make an informed decision given a list of top-10 suggestions. By utilizing attention scores, SARAH can weight features from different inputs and types seamlessly, and report these finding to the end user. Furthermore, by not having additional nonlinearities on top of the attention weights, these features have direct and full impact on the item representation and predicted output. To demonstrate the model's capability in providing meaningful feature distributions, heat maps over the attention scores generated in Eq. 4.2.3 for two randomly sampled items are presented in Figure 4.5.6.

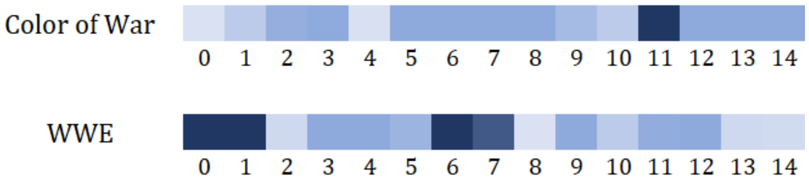


Figure 4.5.6: Visualization of two items' attention scores based on 15 features. Darker colors denote higher weights.

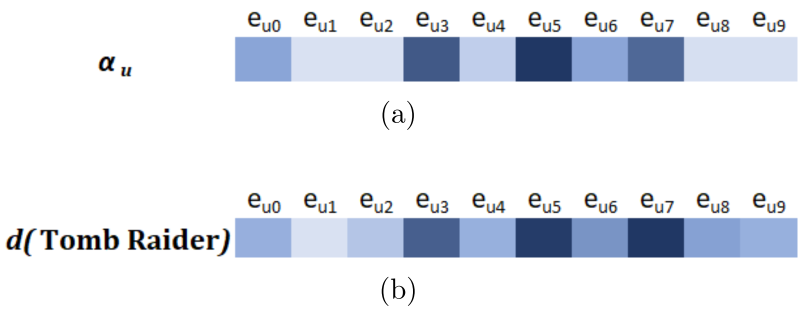
Employing attention weights can provide relevant information to the end user, allowing her to choose an item based on temporal preferences and not only by following historical patterns. For example, "Color of War" was recommended mainly due to the image quality, outlined by the 11-th feature. WWE, on the other hand, is strongly represented by being appropriate to the whole family, being humoristic and its characters, denoted by the features in positions 0,1 and 6, respectively. Accompanying this type of data with the given suggestions allows users to select the items that answer their needs best, without further exploring the given recommendations.

User History Explanations

We further analyze the effect different past items have on a user representation, as illustrated in Figure 4.5.7 using heat maps. First, we provide the importance of 10 past items for a randomly sampled user, u, in Figure 4.5.7a. The presented map shows the attention values generated in Section 4.2.2, where darker colors denote higher impact of an item over the users' behavior. However, we wish to further delve into the effect these weights have in the process of item recommendation. To this end, we sample a positive item, i, and measure the similarity between each of the user k items' embeddings and that of the candidate item:

$$d(i) = [d(e_i, e_{u1}), d(e_i, e_{u2}), ..., d(e_i, e_{uk})] \ , \qquad (4.5.1)$$

where d is the euclidean distance and e_* a past item embedding as presented in section 4.2.1. The sample outcome of Eq. 4.5.1 for the movie "Tomb Raider" liked by user u is shown in Figure 4.5.7b, where darker colors represent higher similarity. As can be witnessed by the two heat maps, there is an apparent correlation between corresponding columns. The items that best capture the user's behavior, denoted as e_{u3},e_{u5} and e_{u7}, are also the most similar to the liked movie.

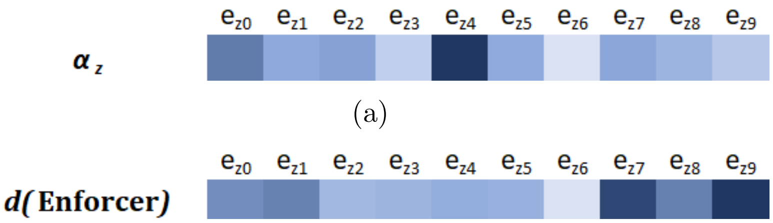


(a)

(b)

Figure 4.5.7: Visualization of user attention scores and euclidean distance between past items and a positive item

We conduct the same experiment over a negative example, provided in Figure 4.5.8. The corresponding heat maps now represent the past items' importance for user z and the similarity of each past item to a movie the user did not like, "Enforcer". In the negative case, however, there is a clear discordance between the two heat maps. "Beauty and the Beast", the movie that best represent the user, denoted as e_{z4}, is relatively different from the candidate movie, which is a thriller. To conclude, the provided visualizations demonstrate that the past items best describing the user are closely related to items she likes, while there is no such relation to other items.



(a)

(b)

Figure 4.5.8: Visualization of user attention scores and euclidean distance between past items and a negative item

4.5.4 Comparison To TCENR (RQ4)

In this section, we examine the two proposed frameworks' ability to be applied successfully in a dedicated point-of-interest recommendation scenario. We evaluate SARAH and DARIA over a concise subset of the Yelp dataset, as described in Section 3.5, and compare the two frameworks to TCENR and its extension,TCENR$_{\text{seq}}$. Table 4.5.2 presents the comparison *w.r.t.* HR@10, NDCG@10, accuracy and mean squared error.

As can be witnessed from the results depicted in table 4.5.2, although our explainable recommenders are general frameworks, they can be successfully applied in challenging settings, such as POI recommendation. Moreover, by significantly outperforming both TCENR and TCENR$_{\text{seq}}$, SARAH and DARIA achieve promising results even compared to empirically proven specialized models. Evaluating the two

Metric	**HR@10**	**NDCG**@10	**Accuracy**	**MSE**
TCENR	0.4927 (59.27%)	0.2608 (95.68%)	0.8279 (2.1%)	0.1171 (3.5%)
TCENR$_{\text{seq}}$	0.49 (60.14%)	0.263 (94.22%)	0.8273 (2.18%)	0.1161 (2.67%)
DARIA	$0.742^{\star}$ (5.75%)	$0.4611^{\star}$ (10.78%)	$0.8354^{\star}$ (1.19%)	$\mathbf{0.1104}^{\star}$ (-2.36%)
SARAH	$\mathbf{0.7847}^{\star}_{\diamond}$	$\mathbf{0.5108}^{\star}_{\diamond}$	$\mathbf{0.8453}^{\star}_{\diamond}$	$0.113^{\star}$

Table 4.5.2: Performance comparison between SARAH, DARIA, TCENR and TCENR$_{\text{seq}}$, using the concise subset of the Yelp dataset, presented in Section 3.5. $\star$ and $\diamond$ indicate significant improvement over the two models presented in Chapter 3 and DARIA, respectively, at the 0.01 level. Relative improvement of SARAH compared to each model is given in brackets.

alternatives demonstrates a small advantage to SARAH over DARIA in all metrics but mean squared error. This phenomena may imply that while DARIA is making relatively smaller mistakes, they are more critical and result misclassification.

Chapter 5

Conclusion and Future Work

In this thesis we presented three novel approaches to tackle some of the more pressing problems in the area of recommender systems. We first proposed TCENR, a framework that utilizes two types of neural networks, MLP and CNN, to extract data from past activities and written reviews. This method, dedicated to the task of POI recommendation, was able to outperform its baselines in terms of accuracy and MSE. We further introduced an extension, denoted as TCENR$_{\text{seq}}$, to examine the impact of employing RNN as the main component for textual modeling. Comparing the model variations demonstrated the importance of fitting the right architecture with the desired goal and inputs.

We then changed our focus to the problem of interpretability in recommender systems based on deep learning. The use of neural attention allowed us to propose two explainable frameworks, while not settling on inferior model accuracy. We presented SARAH, a method consisting on self-attention to learn item and user representations from features and past feedbacks, respectively. We then proposed DARIA, a model that utilizes two layers of standard neural attention over the same inputs. Evaluating the two frameworks provided us with insight towards the effect of different neural attention approaches in the same recommendation setting, where SARAH proved to outperform its variation. Nonetheless, the ability to seamlessly focus on the most relevant features and past items allowed both SARAH and DARIA to significantly outperform six varied baselines over four different datasets.

Comparing the distinct approaches presented in this work shows some resemblance. All methods not only rely on past user-item interactions, but on the availability of additional attributes. While in TCENR we explicitly focused on reviews, geographical locations and social networks, in SARAH we allowed items to be represented by any type of feature. However, relying on external data in conjunction with user feedbacks is a key component, contributing to the consistently high performance in varied scenarios.

For future work, we could further improve SARAH to employ user features, allowing it to be more resistant to the user cold start problem. A further improvement is the development of an interactive recommender system based on SARAH. While an end user is currently aware of how is she perceived by the system, we could allow her to adjust the attention weights, making the model more compatible with her interests.

Bibliography

[1] Amjad Almahairi, Kyle Kastner, Kyunghyun Cho, and Aaron Courville. Learning distributed representations from reviews for collaborative filtering. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 147–154. ACM, 2015.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Trapit Bansal, David Belanger, and Andrew McCallum. Ask the gru: Multi-task learning for deep text recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 107–114. ACM, 2016.

[4] Vito Bellini, Angelo Schiavone, Tommaso Di Noia, Azzurra Ragone, and Eugenio Di Sciascio. Knowledge-aware autoencoders for explainable recommender sytems. *arXiv preprint arXiv:1807.06300*, 2018.

[5] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 46–54. ACM, 2018.

[6] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[7] Rose Catherine and William Cohen. Transnets: Learning to transform for recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 288–296. ACM, 2017.

[8] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. Neural attentional rating regression with review-level explanations. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1583–1592. International World Wide Web Conferences Steering Committee, 2018.

[9] Jingwu Chen, Fuzhen Zhuang, Xin Hong, Xiang Ao, Xing Xie, and Qing He. Attention-driven factor model for explainable personalized recommendation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 909–912. ACM, 2018.

[10] Jinpeng Chen, Wen Zhang, Pei Zhang, Pinguang Ying, Kun Niu, and Ming Zou. Exploiting spatial and temporal for point of interest recommendation. *Complexity*, 2018, 2018.

[11] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. Sequential recommendation with user memory networks. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 108–116. ACM, 2018.

[12] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10. ACM, 2016.

[13] Zhiyong Cheng, Ying Ding, Lei Zhu, and Mohan Kankanhalli. Aspect-aware latent factor model: Rating prediction with ratings and reviews. *arXiv preprint arXiv:1802.07938*, 2018.

[14] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[15] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 191–198. ACM, 2016.

[16] Travis Ebesu, Bin Shen, and Yi Fang. Collaborative memory network for recommendation systems. *arXiv preprint arXiv:1804.10862*, 2018.

[17] Mingsheng Fu, Hong Qu, Dagmawi Moges, and Li Lu. Attention based collaborative filtering. *Neurocomputing*, 2018.

[18] Yuyun Gong and Qi Zhang. Hashtag recommendation using attention-based convolutional neural network. In *IJCAI*, pages 2782–2788, 2016.

[19] Prem Gopalan, Jake M Hofman, and David M Blei. Scalable recommendation with hierarchical poisson factorization. In *UAI*, pages 326–335, 2015.

[20] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

[21] Jonathan L Herlocker, Joseph A Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 241–250. ACM, 2000.

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

[23] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 20(11):1254–1259, 1998.

[24] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. *arXiv preprint arXiv:1808.09781*, 2018.

[25] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. Convolutional matrix factorization for document context-aware recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 233–240. ACM, 2016.

[26] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender systems handbook*, pages 77–118. Springer, 2015.

[29] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[32] Huayu Li, Yong Ge, Richang Hong, and Hengshu Zhu. Point-of-interest recommendations: Learning potential check-ins from friends. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 975–984. ACM, 2016.

[33] Sheng Li, Jaya Kawale, and Yun Fu. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 811–820. ACM, 2015.

[34] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[35] Qiang Liu, Shu Wu, Diyi Wang, Zhaokang Li, and Liang Wang. Context-aware sequential recommendation. *arXiv preprint arXiv:1609.05787*, 2016.

[36] Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics*, 10(2):1273–1284, 2014.

[37] Hao Ma, Dengyong Zhou, Chao Liu, Michael R Lyu, and Irwin King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296. ACM, 2011.

[38] Jarana Manotumruksa, Craig Macdonald, and Iadh Ounis. A deep recurrent collaborative filtering framework for venue recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1429–1438. ACM, 2017.

[39] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172. ACM, 2013.

[40] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–52. ACM, 2015.

[41] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[42] Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[43] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[44] Cataldo Musto, Fedelucio Narducci, Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Linked open data-based explanations for transparent recommender systems. *International Journal of Human-Computer Studies*, 2018.

[45] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[46] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[47] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[48] Francesco Ricci, Lior Rokach, and Bracha Shapira. Recommender systems: introduction and challenges. In *Recommender systems handbook*, pages 1–34. Springer, 2015.

[49] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[50] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. Disan: Directional self-attention network for rnn/cnn-free language understanding. *arXiv preprint arXiv:1709.04696*, 2017.

[51] Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. Moviexplain: a recommender system with explanations. In *Proceedings of the third ACM conference on Recommender systems*, pages 317–320. ACM, 2009.

[52] Duyu Tang, Bing Qin, Ting Liu, and Yuekui Yang. User modeling with neural network for review rating prediction. In *IJCAI*, pages 1340–1346, 2015.

[53] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Latent relational metric learning via memory-based attention for collaborative ranking. In *Proceedings of the 2018 World Wide Web Confer-*

ence on World Wide Web, pages 729–739. International World Wide Web Conferences Steering Committee, 2018.

[54] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Multi-pointer co-attention networks for recommendation. *arXiv preprint arXiv:1801.09251*, 2018.

[55] Nava Tintarev and Judith Masthoff. Designing and evaluating explanations for recommender systems. In *Recommender systems handbook*, pages 479–510. Springer, 2011.

[56] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Advances in neural information processing systems*, pages 2643–2651, 2013.

[57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[58] Chong Wang and David M Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 448–456. ACM, 2011.

[59] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1244. ACM, 2015.

[60] Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Dkn: Deep knowledge-aware network for news recommendation. *arXiv preprint arXiv:1801.08284*, 2018.

[61] Weiqing Wang, Hongzhi Yin, Ling Chen, Yizhou Sun, Shazia Sadiq, and Xiaofang Zhou. Geo-sage: A geographical sparse additive generative model for spatial item recommendation. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1255–1264. ACM, 2015.

[62] Xiang Wang, Xiangnan He, Fuli Feng, Liqiang Nie, and Tat-Seng Chua. Tem: Tree-enhanced embedding model for explainable recommendation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1543–1552. International World Wide Web Conferences Steering Committee, 2018.

[63] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*, pages 495–503. ACM, 2017.

[64] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. Collaborative denoising autoencoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 153–162. ACM, 2016.

[65] Carl Yang, Lanxiao Bai, Chao Zhang, Quan Yuan, and Jiawei Han. Bridging collaborative filtering and semi-supervised learning: a neural approach for poi recommendation. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1245–1254. ACM, 2017.

[66] Hongzhi Yin, Yizhou Sun, Bin Cui, Zhiting Hu, and Ling Chen. Lcars: a location-content-aware recommender system. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 221–229. ACM, 2013.

[67] Hongzhi Yin, Weiqing Wang, Hao Wang, Ling Chen, and Xiaofang Zhou. Spatial-aware hierarchical collaborative deep learning for poi recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2537–2551, 2017.

[68] Hongzhi Yin, Xiaofang Zhou, Yingxia Shao, Hao Wang, and Shazia Sadiq. Joint modeling of user check-in behaviors for point-of-interest recommendation. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1631–1640. ACM, 2015.

[69] Shuai Zhang, Lina Yao, Aixin Sun, Sen Wang, Guodong Long, and Manqing Dong. Neurec: On nonlinear transformation for personalized ranking. *arXiv preprint arXiv:1805.03002*, 2018.

[70] Yongfeng Zhang, Qingyao Ai, Xu Chen, and W Bruce Croft. Joint representation learning for top-n recommendation with heterogeneous information sources. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1449–1458. ACM, 2017.

[71] Lei Zheng, Vahid Noroozi, and Philip S Yu. Joint deep modeling of users and items using reviews for recommendation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 425–434. ACM, 2017.

[72] Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiusi Chen, and Jun Gao. Atrank: An attention-based user behavior modeling framework for recommendation. *arXiv preprint arXiv:1711.06632*, 2017.