

MASTER

Long- and Short-term Sequential Recommendation with Enhanced Temporal Self-attention

Fang, Zian

Award date:
2021

[Link to publication](#)

Disclaimer

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain



Department of Mathematics and Computer Science
Uncertainty in Artificial Intelligence Group

Long- and Short-term Sequential Recommendation with Enhanced Temporal Self-attention

Master Thesis

Zian Fang

Supervisor: Maryam Tavakol

Eindhoven, November 30, 2021

Abstract

With the tremendous growth of the Internet, recommendation systems have become increasingly significant since they assist consumers in efficiently filtering out products that match their preferences. The core of the sequential recommendation is to use the user’s history of interactions and additional information to model the user’s preferences and provide them with a list of recommendations that match their preferences. In real life, users’ interests and preferences fluctuate over time, and the majority of current sequential recommendation algorithms only consider long- or short-term user preferences, and most of them only consider the order of sequences and ignore the time interval between items and items. Although some proposed models, such as **TiSASRec** [21], has managed to take advantage of temporal information, there is still a great deal of features contained in the timestamps that can be exploited, and the algorithm itself actually only model the long-term preference.

Therefore, in this project, in order to capture the dynamical user interests, we propose a model, Ti-SACNN, which takes both temporal information and long- and short-term preferences into consideration based on **SASRec** [18]. Different from [21], we design an enhanced temporal self-attention module which further explores the features hidden inside the timestamps. And for the short-term preference, we utilize the vertical filter in the convolutional layer to capture users’ recent intention based on the last few interactions. We conduct comprehensive experiments on three public datasets, and the results show that our proposed model can effectively capture users’ dynamic preferences and it outperforms the comparison baseline methods on common evaluation metrics.

Preface

The completion of this master project represents the end of my master life in Eindhoven University of Technology. Although numerous obstacles occurred at the beginning stage, the research process is definitely educative and ends up with a fruitful result.

Thus, I would like to firstly thank my supervisor, Dr.Maryam Tavakol, for her careful supervision during my project. Maryam has been incredibly supportive and patient throughout the process, and she has always provided critical feedback on my study. With her advice and help, I was able to successfully complete this project.

I would like to express my gratitude to my parents for being a source of moral encouragement when I was in a state of confusion and anxiety. And I also would like to thank my friends for supporting me, encouraging me and giving me a lot of valuable advice. The days with them would be some of the most memorable of my life.

Finally, I'm grateful to TU/e for providing such an excellent platform that we can not only acquire a great deal of cutting-edge knowledge, but also conduct professional academic research and broaden our global viewpoint.

Contents

Contents	vii
1 Introduction	1
2 Literature Analysis	5
2.1 Recommendation System Based on Traditional Methods	5
2.1.1 Collaborative Filtering	5
2.1.2 Matrix Factorization	6
2.1.3 Factorization Machine	7
2.1.4 Markov Chain	7
2.2 Recommendation System Based on Deep Learning	8
2.2.1 Conventional Method & Deep Learning Hybrid Model	8
2.2.2 Deep Learning Based Model	9
2.3 Sequential Recommendation	11
3 Background	15
3.1 Attention Mechanism	15
3.1.1 Self-attention	16
3.2 Transformer	18
3.3 Convolutional Neural Network	21
3.4 Wavenet	24
4 Method	27
4.1 Proposed method	27
4.2 Problem formulation	28
4.3 Self-attention with temporal information	29
4.3.1 Position modelling	29
4.3.2 Embedding layer	29
4.3.3 Temporal self-attention:	30
4.3.4 Feed-forward network:	31
4.3.5 Wavenet	32
4.4 Short-term preference modelling	32
4.5 Prediction layer	34
4.6 Network training	34

5	Experiment and Result Discussion	35
5.1	Experiment setup	35
5.1.1	Dataset	35
5.1.2	Baselines	36
5.1.3	Evaluation Metrics	36
5.1.4	Parameter Settings	37
5.2	Result discussion	38
5.2.1	General performance of Ti-SAN	38
5.2.2	Influence of the self-attention block	39
5.2.3	Influence of the user	39
5.2.4	Influence of the Wavenet	40
5.2.5	Influence of maximum sequence length	41
5.2.6	General performance of Ti-SACNN	41
6	Conclusion and Future Work	43
6.1	Conclusion	43
6.2	Future Work	44
	Bibliography	45

Chapter 1

Introduction

Nowadays, people can no longer live without the wealth of services offered by the Internet and mobile App, and the personalized service provided by the recommendation system plays a key role. The function of the recommendation system is to recommend to users the products they are most interested in. In the field of e-commerce such as Amazon and Alibaba, or in the field of social networks such as Facebook and Twitter, or in the field of videos such as YouTube, Netflix, and TikTok, they all rely heavily on their own recommendation systems for their content distribution strategies, for example, 80% of movies watched on Netflix came from recommendations [10]. In today's world of massive amounts of data, users' needs are often uncertain when it comes to accessing information, or search engines do not provide good feedback on the keywords that users are searching for. At this time, the recommendation system can build a model to explore the potential interests of the user based on the user's history of interaction with the website or app, as well as information about the user's social attributes and friend network, and then provide personalized recommendations for each user based on the potential interests. This can not only satisfy the user's needs and enhance the user experience but also increase the conversion rate of the product or information, thus increasing the revenue of the app or website.

The core of a recommendation system is the recommendation algorithm. One of the early representative recommendation algorithms is the collaborative filtering-based recommendation algorithm [31], it is quite popular in the time mainly because of its simplicity and good effects. The core idea of collaborative filtering is clustering, for example, users with similar histories are used as references for recommendation systems. Because early recommendation systems relied heavily on user ratings of items, in many cases the recommendation problem could be transformed into a user rating prediction problem for items, using a user-item rating matrix to compute, hence the methods that use matrix factorization [19] to predict ratings is proposed. This type of approach maps users and goods onto a low-dimensional space and introduces the concept of hidden vectors, but the limitations of matrix factorization algorithms are gradually exposed as the scale of recommendation system increases and the problem of cold-starting. Today's web servers basically record user history behaviour through sequences or sessions, as the traditional recommendation algorithms mentioned above are heavily dependent on user purchase behaviour, so the performance of the algorithm suffers in this scenario. Hence, algorithms for sequence is designed. Sequential recommendation systems attempt to discover and predict sequential user behaviors, user-item interactions, and the

change of item popularity and user preferences overtime [38]. Most of the algorithms that were first used for sequential recommendations were relatively simple machine learning algorithms, such as Markov chain [30]. Sequential recommendation based on Markov chain usually only take the last click or good into account to predict the next item, so it performs well when the sequence is short. However, when the sequence become relatively long, the performance of the model can become worse significantly because it neglects the majority information of the sequence.

In recent years, with the continuous development of deep learning, recommendation system based on deep learning has attracted the attention of many researchers. The last few years have witnessed the remarkable success of deep learning technique in numerous application fields, including computer vision and natural language processing. Deep learning has the advantage of extracting and identifying effective latent feature representation from complex data. Because in the real environment, there always exists some non-trivial and non-linear relationships between users and items, by using deep learning, these kind of complex relations can be captured. In addition, complex relationships that exist between the data itself in massive amounts of data, for example context information, can also be uncovered by deep learning techniques. According to Zhang et al.[46], deep learning based recommendation systems can be divided into following subcategories: the Multi-Layer Perceptron(MLP), the Autoencoder(AE), the Convolutional Neural Network(CNN), the Recurrent Neural Network(RNN), the Restricted Boltzmann Machine(RBM), the Neural Autoregressive Distribution Estimation (NADE), the Adversarial Network(AN), the Attention Model(AM), Deep Reinforcement Learning(DRL) and hybrid models. In sequential recommendation, the common deep learning based algorithms are CNN, RNN, GNN and Attention Model. Methods such as RNN and attention mechanisms are widely used for the extraction of information within sequences. They can extract the complete sequence information of the user and model the sequential pattern of the sequence, thus providing a large improvement over traditional methods. However, RNNs alone cannot learn the relevance information between items within a sequence well, which often expresses users' true preferences more effectively. The attention mechanism in recommendation system can learn this relevance information very effectively and distinguish between the importance of features. Furthermore, it allows direct inspection of the inner workings of the mechanism, by visualising the attention weights of the inputs and corresponding outputs [22]. Hence, not only the performance but also the interpretability of the recommendation system can be improved. In this project, our research is mainly based on the attention mechanism based recommendation system.

In many past studies about sequential recommendation, researchers have generally considered only the order in the sequence, and also the temporal factor has often been ignored. As we all know, temporal information often contains a wealth of useful information, and temporal information is quite critical in some fields, such as time series forecasting. Therefore, we assume that including the temporal elements can help improve the recommendation model's effectiveness, and in recent works, such as **TiSASRec** [21], has incorporated the temporal information and has proven to be effective. Fig 1.1 shows a simple example to illustrate the importance of the time factor. We can see that there are instances in real life where the order of sequence is same but the user's interests change at different time intervals. If the purchase interval in the sequence is long, then the next item the model recommends for that user will be another type of biscuit, as biscuits appear most frequently in that user's shop-

ping sequence and long-term preference plays an more important role. However, if the time interval between the purchased last item (the chocolate in the figure) and the previous one is relatively close, then the next recommended item will most likely be a different chocolate because recent interests has a greater impact.

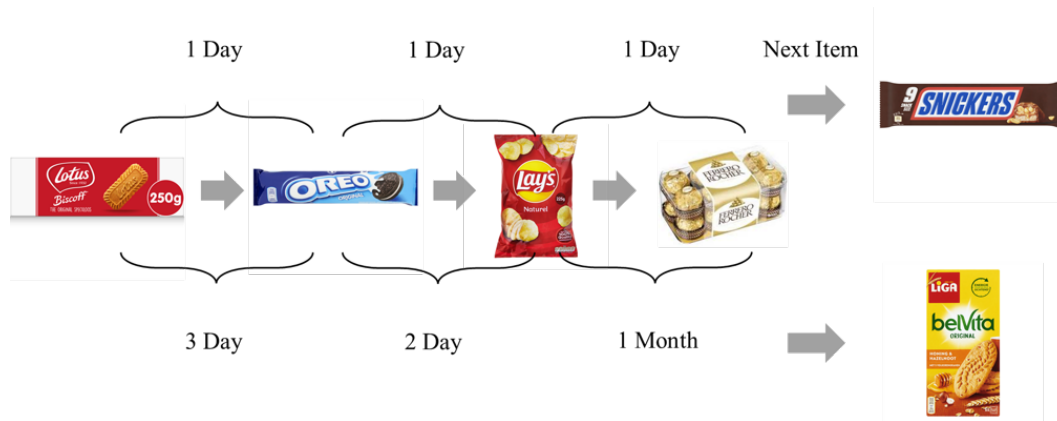


Figure 1.1: Same commodity sequence with different time intervals

Previous research has either focused on long-term preferences or short-term preferences. However, as we all know that the user preference is dynamic and complex, and neither long-term nor short-term preferences can accurately reflect users' true preferences. Hence, recommendation algorithms that use a fusion of long- and short-term preferences to model users' dynamic interests have become a relatively new research direction.

As far as we currently know, although lots of researchers have conducted research in this direction of long- and short-term fusion, few studies have combined long- and short-term preferences together with temporal information to model the sequential recommendation system. Hence, In this project, we propose a robust self-attention based sequential recommendation model which incorporates both long- and short-term preferences, as well as temporal information. By combining the advantages of these two approaches, our proposed sequential recommendation model can provide better recommendations to consumers.

Extensive experiments show that our model perform well on real-world datasets. Our main contributions are summarized as follow:

1. We introduce the time interval and its transformation as additional input in embedding, and we propose an enhanced temporal self-attention mechanism to capture the user's long-term preferences. It can not only extract features from items and absolute position embeddings, but also capture those features hidden inside the relative positions.
2. We utilize a special convolutional neural network to capture the user's short-term preferences. By combining users' long-term preferences with their short-term preferences, our approach will be able to better model their true preferences.
3. We conduct comprehensive experiments based on our proposed method, and we also examined the influence of the various components. Two assessment measures indicate that our model outperforms comparable methods and achieves satisfactory results.

The remainder of this thesis is organized as follows. In Chapter two, we conduct a review of the literature on recommendation systems, ranging from traditional methods to deep learning techniques. This chapter will discuss in full detail several state-of-the-art algorithms and methods that are relevant to this thesis. In Chapter three, we go into detail about the background of the techniques that we utilize in our algorithm. Chapter four provides an overview of the proposed model and then delves into detail about each part. Chapter five describes the experimental setup used to evaluate our models and summarizes the findings and conclusions from our experiments. At last, in Chapter six, we draw conclusions from our experiments and findings and outline several directions that can be improved in the future for research based on the observations and conclusions in this thesis.

Chapter 2

Literature Analysis

Over the past few decades, researchers have developed a variety of recommendation algorithms to improve the performance of recommendation systems. From traditional algorithms such as collaborative filtering to today's deep learning-based recommendation algorithms, the number is large enough to drive recommendation systems research forward. In this section, a selection of previous related works from conventional recommendation systems to deep learning-based recommendation systems, and especially the state-of-the-art algorithms in the sequential recommendation, are introduced.

2.1 Recommendation System Based on Traditional Methods

2.1.1 Collaborative Filtering

Collaborative filtering can efficiently filter out valid information from massive amounts of data, and it is one of the most effective recommendation algorithms available. Collaborative filtering relies only on the relation between user and item, based on the assumption that users with the same rating for the same item may have similar preferences. Collaborative filtering will recommend items based on the interests of other users with similar interests. It can also make recommendations by identifying items similar to items previously rated by the target user and calculating a similarity value by counting the frequency of simultaneous user or item occurrences.

















		Item			
		W	X	Y	Z
User	A				
	B				
	C				
	D				
	E				

Figure 2.1: User-Item Matrix in Collaborative Filtering

matrix factorization to decompose the item-item similarity matrix to the product of two low-dimensional latent factor matrices.

2.1.3 Factorization Machine

Factorization Machine(FM) [29] was firstly proposed by Steffen Rendle in 2010. The model solves the classification problem of large-scale sparse data mainly through feature combination. The equation for FM is as follows, where n is the number of features in sample, x_i is the i^{th} feature's value, θ_0 and θ_1 are the parameters of the model, and $\langle \theta_0, \theta_1 \rangle$ is the inner product of two vectors. This formulation allows the FM model to automatically combine features to generate non-linearities by introducing a second-order cross-feature combination term into the logistic regression model, which allows for more effective mining of non-linear information within the first-order features and increases the model's fitting ability and generalisability. The model also uses hidden vector dot product to learn the weights of the second-order cross terms, which can effectively avoid the problem caused by sparse data.

$$\hat{y} = h_{\theta}(x) = \theta_0 + \sum_{i=1}^n \theta_1 x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^n \langle \theta_i \cdot \theta_j \rangle x_i x_j \quad (2.1)$$

DeepFM [12], similar to Wide&Deep [7] model, is also obtained from the joint training of the shallow and deep models. The difference is that the Wide part is replaced by LR with FM, which has the ability to learn cross features automatically, avoiding the manual feature engineering work in the shallow part of the original Wide&Deep model. The DeepFM model contains DNN model. The FM can extract low-order features, while the DNN can extract high-order features.

FM can be seen as an extension of MF, dealing with user and item in addition to integrating other features. All features are converted to low-dimensional vectors, and the combination of weights occurs between any two features. If FM uses only user and item, the process is the same as MF. FM inherits the features of the embedding mechanism and extends them with other features. This makes FM more flexible and can be applied in a wider range of scenarios.

2.1.4 Markov Chain

In sequential recommendation, in order to capture the user's dynamic interests, the Markov Chain model is firstly introduced. This model assumes that the next click or item is only related to the previous one or several actions. **FPMC** [30] combines matrix factorization and Markov Chain together and combines the strengths of both approaches, providing a separate transition matrix for each user, capturing time-series data while dealing with highly sparse data, and modeling users' long-term preferences. Shani et al. [33] proposed a recommendation algorithm named Markov decision process (MDP). The simple MDP algorithm can be reduced to a first-order Markov chain, modeling the transfer probabilities between items and using the conditional probabilities calculated by the model to generate the recommendations. Chen et al. [5] utilized latent Markov embedding to predict music playlists.

Markov Chain-based recommendation system performs well when the data is sparse. However, when the sequence becomes complex, it has the problem of information loss and low scalability.

2.2 Recommendation System Based on Deep Learning

2.2.1 Conventional Method & Deep Learning Hybrid Model

Much previous work combines a neural network with traditional recommendation algorithms to get better recommendations. The DeepFM model, as mentioned in the previous section, is a combination of FM and DNN to obtain better results. In this subsection, more hybrid recommendation algorithms will be introduced.

CF & Deep Learning Methods

NCF(Neural Collaborative Filtering) [14] provides a generic framework for solving collaborative filtering problems using neural networks on the basis of implicit feedback. After inputting user and item embedding to a fully connected layer and multi-layer perceptron, the user-item interaction function can be learned. He et al. [13] presented the **ConvNCF**, which proposed employing CNNs to improve NCF. It models user and item interaction patterns using the outer product rather than the dot product. CNNs are applied to the outer product result and can capture high-order correlations between embedding dimensions.

AutoRec [32], an autoencoder-based collaborative filtering model, successfully applied autoencoder in collaborative filtering. By adding a two-level attention mechanism to a latent factor model, Chen et al. [3] created an attentive collaborative filtering model. It is made up of attention at the item and component levels. To characterize users, item-level attention is employed to identify the most representative objects, and component-level attention tries to extract the most useful information for each user. Additionally, several researchers have experimented with merging RNN and collaborative filtering. Devooght [9] explored the use of the LSTM for the collaborative filtering problem. The authors innovatively propose that the collaborative filtering method commonly used for traditional recommendations can be viewed as a time series prediction problem.

FM & Deep Learning Methods

Lian et al. [24] proposed **xDeepFM**, which is based on DeepFM. Because while in DeepFM, ordinary DNNs generate feature interactions implicitly and at the bit-wise level, in xDeepFM, a novel Compressed Interaction Network (CIN) is paired with a classic DNN with the goal of explicitly and vector-wise generating feature interactions. Not only can the xDeepFM learn certain bounded-degree feature interactions, but also low- and high-order feature interaction. However, because xDeepFM has high complexity and it is easy to overfitting, a more expressive but lightweight model called the High-order Attentive Factorization Machine (HoAFM) is proposed [35]. The model explicitly accommodates for higher-order sparse feature interactions.

Because in the previous work, which combined FM with the neural network, the embedding dimensions are assumed to be independent and high-order interactions are modeled implicitly. Due to these limitations, Convolutional Factorization Machine (CFM)[40] is proposed. CFM generates image-like data by modeling second-order interactions through outer products, capturing correlations between the embedding layers. All generated image data is

then stacked to form a collection of interaction matrices on which convolution is applied to learn higher-order interaction signals.

2.2.2 Deep Learning Based Model

In this subsection, We will demonstrate several representative recommendation models that are entirely based on a single deep learning algorithm or on a combination of various hybrid algorithms.

MLP Based Model

MLP(Multi-Layer Perceptron) is an essential part of deep learning. **Wide&Deep** [7] is a classic and general model which consists of a shallow(or single-layer) neural network-Wide part and deep multi-layer neural network-Deep part, with the output layer using softmax or logistic regression to combine the outputs of the Wide and Deep parts. The Wide part uses cross-products to produce features with good results and good interpretability, while the Deep part converts sparse features into dense features to further explore higher-order feature combinations, which has a strong generalization capability in the case of sparse data. By combining these two parts, the performance of the recommendations becomes better.

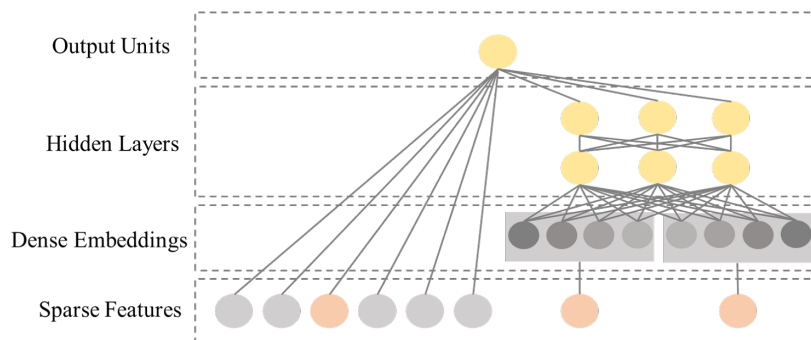


Figure 2.3: Taxonomy of deep reinforcement learning in recommendation systems

RNN Based Model

RNN(Recurrent Neural Networks) is a famous neural network that is quite effective when the input data is dependent and sequential. Session and cookie techniques enable web servers to obtain a user’s short-term preferences. Due to the severe sparsity of data, this is a largely underappreciated task in recommendation systems. Recent advancements have proved RNNs’ usefulness in resolving this problem.

Hidasi et al. [16] proposed a session-based recommendation model, **GRU4Rec**, based on GRU. The model begins by taking each item clicked by the user individually as an input to GRU4Rec, then goes through the embedding layer to obtain the item’s embedding, then through several layers of GRU, then several layers of MLP, and finally predicts the item that the consumer is most likely to click next time. Another highlight of this paper is that the authors proposed a session-parallel mini-batches algorithm to effectively train the model.

Because the GRU4Rec does not consider the side information, Hidasi et al.[15] proposed a session-based recommendation algorithm based on parallel architecture, which exploits the features from image feature, identity one-hot, and text feature vectors.

CNN Based Model

CNN(Convolutional Neural Networks) is a well-known deep learning model whose name comes from the fact that convolutional operations were introduced into this model. It has been very widely used in the field of computer vision, and in recent years, researchers have utilized CNNs for feature extraction in recommendation systems and find that it can perform well. According to Zhang et al. [46], applications of CNNs in recommendation systems can mainly be divided into three parts: image, text, and audio/video feature extraction.

Wang et al.[39] utilized CNN to examine the effects of visual elements on point-of-interest (POI) recommendation and presented a visual content enhanced POI recommendation system(VPOI). The model is based on the relationship between the address the image is tagged with, the relationship between the user posting the image, and the relationship between the user checking in at the address. In order to fully mine the features inside the text, **DeepCoNN** [47] models user behaviors and item attributes from review texts using two concurrent CNNs. This model overcomes the sparsity issue and improves the model's interpretability by utilizing CNNs to create rich semantic representations of review texts. The outputs of the user network and item network are concatenated to form the prediction layer's input, where the FM is used to capture their interactions for rating prediction.

Deep Reinforcement Learning

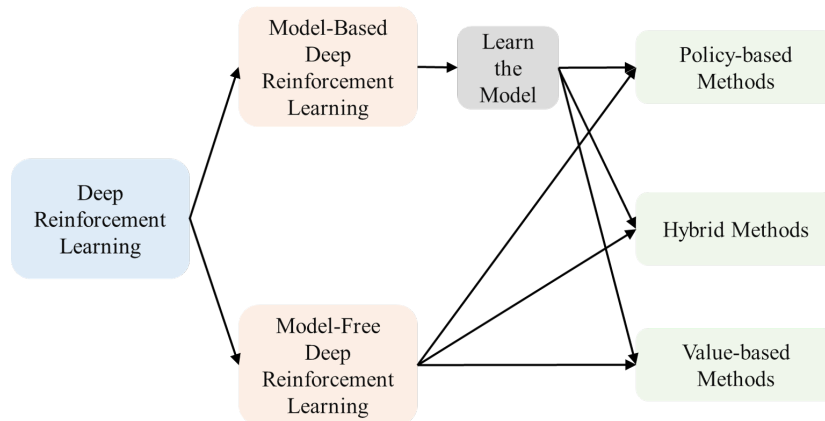


Figure 2.4: Taxonomy of deep reinforcement learning in recommendation systems [6]

Deep reinforcement learning has recently received greater attention in the field of recommendation systems because the preferences of some users are likely to change rapidly, resulting in some deep learning models based on existing datasets not being able to capture these changes effectively. Because the goal of deep reinforcement learning is to combine the power of deep learning and reinforcement learning to train an agent that can learn from the interaction trajectories provided by the environment, DRL is particularly well suited to learning from in-

teractions because the agent in DRL can actively learn from real-time feedback from the user and thus infer the user's dynamic preferences. Deep reinforcement learning can be divided into several types as shown in the fig 2.4.

Deep learning based recommendation systems reflect user interest and update the recommendation item by receiving feedback from the user (e.g. ratings or clicks), while deep reinforcement learning based recommendation systems update the policy by receiving rewards from the environment. Researchers in YouTube [4] utilized the reinforce recommendation system to get the most out of annual revenue optimisation. They apply the policy-based reinforcement algorithm to the recall session. The problem of how to utilize scene samples is addressed by a jointly trained model that predicts the selection probability of item for importance-sampling, transforming the on-policy algorithm into an off-policy. The probability of action space is also smoothed by a top-k correction, and the effect of this correction is demonstrated in simulations and online experiments.

Attention Based Model

In recent years, the attention mechanism has been extensively applied in the fields of image and natural language processing, and recommendation system based on the attention mechanism has emerged as a new avenue. The attention mechanism is motivated by human visual attention. It can distinguish the importance of each potential feature to improve the interpretability and the performance of the recommendation system.

The attention mechanism is often combined with neural networks such as DNNs, CNNs, RNNs, and GNNs. Based on the hidden features learned by these neural networks, the attention mechanism calculates their respective weights and then makes a better recommendation. Gong et al. [11] proposed a hashtag recommendation system based on attention mechanism and CNNs in a micro-blog. The model consists of a global channel that uses CNN to process all words and a local channel that employs an attention mechanism to process trigger words. CNN is utilized primarily to eliminate hand-crafted characteristics in this model, while the attention mechanism is employed to select informative words. Li et al. [23] proposed a hashtag recommendation algorithm based on an attention-based LSTM model. This study uses both RNNs and attention mechanisms to extract sequential characteristics and identify important terms from microblog data. The attention mechanism can be used to avoid some of the problems associated with RNNs, such as information loss owing to excessively long inputs.

2.3 Sequential Recommendation

In contrast to the recommendation systems discussed previously, sequential recommendation systems attempt to comprehend and model sequential user behaviors, user-item interactions, and the evolution of users' preferences over time. As a result, sequential recommendation systems generate more precise, customized, and dynamic recommendations. This subsection will introduce and discuss sequential recommendation systems that are based on various techniques.

Markov Chain

Markov chain is a traditional sequential recommendation algorithm. According to Wang et al. [38], There are two types of Markov chain-based recommendation systems: one is basic Markov chain and it calculates the transition probability directly, whereas the other one is latent Markov embedding which calculates between interactions based on Euclidean distance in Euclidean space. As mentioned above, **FPMC** [30] uses a combination of matrix factorization and decomposed first-order Markov chains to recommend. Due to the assumption that the current interaction is dependent on only the most recent interactions, Markov properties capture only short-term dependencies and ignore long-term dependencies and user-item interactions.

RNN

Due to RNN's inherent advantage in sequence modeling, RNNs are the most frequently used deep neural networks for sequential recommendations. As mentioned above, **GRU4Rec** [16] which is based on GRU has been proposed to capture the long-term dependencies in a sequence. Apart from the basic RNN structure, **HRNN** [28], a hierarchical RNN model which is based on GRU4Rec is proposed. This algorithm utilizes an additional GRU layer to better model user's preferences.

As mentioned above, RNN also has disadvantages. For example, because the dependency assumptions of RNNs do not exactly match the relationships between data in real-life sequences, there is a possibility that RNNs will produce invalid dependencies.

CNN

Recently, CNN has also been applied in sequential recommendation systems. Unlike RNNs, CNNs treat the input embedding as an image matrix and learn local features by convolutional filtering. **Caser** [34] is a model which is inspired by the application of CNN in text classification to capture users' short-term preference because the author emphasizes the impact of short-term information. The highlight of the Caser is that it presents three different sequential modes in a sequence, especially a skip mode, which represents the interactions among nonadjacent items, which is quite creative. For the filters in the convolutional layer, the model uses two approaches: horizontal filters and vertical filters to extract different sequence pattern information. The final output is the probability that user u will interact with each candidate item at the moment t . Different from Caser, **CosRec** [42] utilized the 2-D convolutional filter to better capture the skip mode, which is mentioned above.

Due to the filter size constraint inherent in CNNs, CNN-based sequential recommendation systems struggle to capture long dependencies. Hence, to be able to capture long-term dependencies, CNNs are often combined with other models, such as LSTM, to better capture the dynamic changes in the sequence. **RCNN**[41] is a model that combines the respective advantages of LSTM and CNN. It first obtains the hidden states from LSTM, then uses CNN to grasp the local features in these hidden states, and finally concatenates the output of CNN with the hidden states of LSTM to perform relevance calculation on the candidate items.

Deep Reinforcement Learning

Deep reinforcement learning is essentially a method that use neural networks as estimators of value functions, the main advantage of which is that it can use deep neural networks to automatically extract state features, avoiding the inaccuracy of manually defined state features and allowing the Agent to learn on more primitive states.

In sequential recommendation, although the current attention mechanism can already distinguish the contribution of different historical items to the recommended item, when a historical interest sequence contains a large number of different interests, the effect of the items that contribute to the recommended item will be influenced by other items with different interests, resulting in poor performance of the attention mechanism. Hence, in order to efficiently remove irrelevant items from the sequence, Zhang et al. [44] proposed a hierarchical reinforcement learning model. This model consists of two important components: the profile reviser and the basic recommendation model. The profile reviser allows for the correction of sequence information without expert annotation. By training the two modules jointly, the model can effectively remove the noisy items from the original sequence.

Attention Mechanism

In recent years, the attention mechanism has become a hot spot and plays a more and more important role in the current research field of sequential recommendation because it overcomes several of the drawbacks associated with other neural network methods and achieves performance over those recommendation algorithms based on CNN, RNN, etc. Several representative sequential recommendation algorithms based on attention mechanisms are presented in this section.

AttRec [45] is a model which is different from other models because it utilizes attention mechanisms to capture short-term preference while using metric learning to get long-term preference. Unlike the attention mechanism in other models, AttRec's attention mechanism is unique in that the Query and Key use shared embedding, and the value is not transformed non-linearly. This is because the author finds that the use of constant mappings is effective. In this model, Euclidean distance is used to model both short-term interests and long-term preferences, and then the final recommendation scores for candidate items are obtained by weighted summation.

STAMP [26] is a short-Term attention/memory priority model. The model also uses the same idea of long- and short-term preference modeling, capturing the user's generic interests from long-term memory of the session context while considering the user's current interests from short-term memory of the last click. However, if a false click occurs at the final click, current interest is likely to have a negative effect on the model. Ying [43] proposed a two-layer hierarchical attention network. In this algorithm, the first attention layer captures long-term dependencies through the main part of the input sequence, and the second attention layer generates the prediction by fusing the output of the first attention layer with the embedding of the last few interactions.

The advent of the **Transformer** [37] has made the self-attention mechanism the dominant

modeling approach to sequence recommendation, starting with the more recent classic **SASRec** [18]. It is a model that is based primarily on the encoder part of the Transformer, and the encoder portion's heart is the self-attention mechanism. Thus, despite the simplicity of SASRec's architecture, it outperforms the majority of previous state-of-the-art models. Self-attention is defined in this paper as follows: for the next item recommendation, the weights of the individual items vary according to the user's behavioral sequence, which is relevant for the recommendation scenario. Because sequence recommendation algorithms such as SASRec only consider the sequential relationships inside the sequence and do not consider the use of features such as relative time intervals of timestamps to mine out the hidden features in the sequence. Hence, on the basis of SASRec, by incorporating timestamp-related features into the embedding layer, **TiSASRec** [21] presents better performance.

Chapter 3

Background

In this chapter, the background knowledge which is involved in this thesis will be presented in detail.

3.1 Attention Mechanism

A rough description of the attention mechanism is "whatever you are doing, you concentrate on it." This mechanism is identical to what occurs when the human brain considers a problem. Bahdanau et al. [2] employ an Attention-like mechanism to perform simultaneous translation and alignment on machine translation tasks, and this is the first time the Attention mechanism has been applied to the field of natural language processing. The Google Machine Translation team published Attention is All You Need [37] in 2017, which heavily relies on self-attentive mechanisms to learn text representation. The self-attention mechanism has become a hot research topic in recent years.

Due to the fact that the attention mechanism is widely used in encoder and decoder models, the following provides an overview of how the attention mechanism works in this model. The workflow is depicted in Fig3.1. The sentence "I am a student" is fed into an LSTM encoding layer, which is then fed into another LSTM decoding layer. When the model is translating, the first output is the h_t in the diagram, because we want the model to think like our human brain, when the first word is decoded, the model can only focus on the word "I" and ignore the other words as much as possible. h_t is the implicit state at the first decoding moment, and the most relevant part of h_t should be the encoding state h_1 corresponding to "I", so the network needs to focus on h_1 when decoding the first word. Since the implicit states of the h_t and encoding parts are in the same embedding space, the similarity can be used to know which implicit state is most similar to the h_t . In this way, when decoding, the network can focus its attention as much as possible on the corresponding implicit state in the encoder. The formula for calculating the similarity coefficient is as follows. The sum of all encoded states is then weighted to obtain the context vector, which is then combined with h_t to obtain the output at the current decoding moment.

$$\alpha_{ts} = \frac{\exp(\text{score}(h_t, h_s))}{\sum_{s'} \exp(\text{score}(h_t, h'_s))} \quad (3.1)$$

$$c_t = \sum_s \alpha_{ts} \bar{h}_s \quad (3.2)$$

$$\alpha_t = f(c_t, h_t) = \tanh(W_c [c_t; h_t]) \quad (3.3)$$

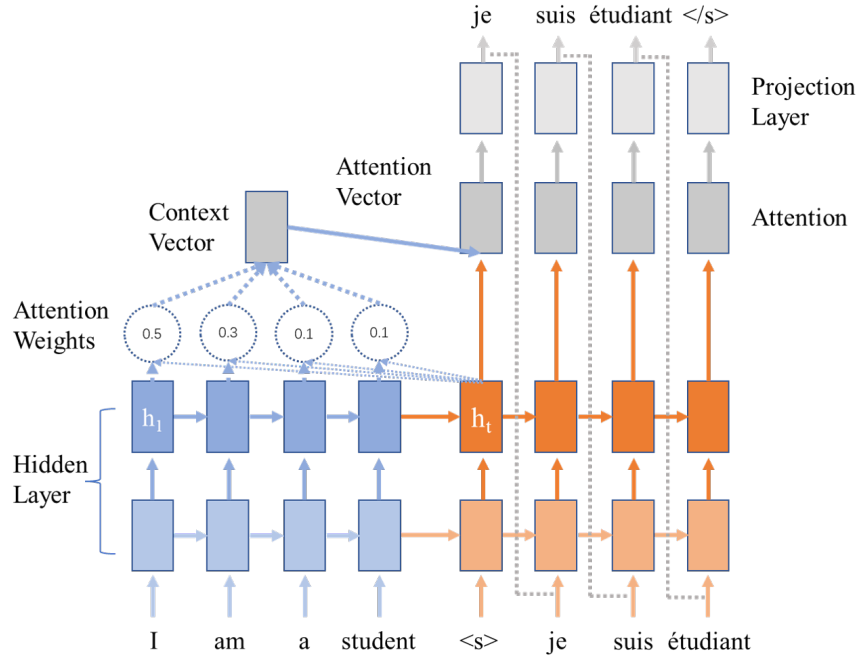


Figure 3.1: Attention mechanism

Attention mechanisms can be divided into three different categories as follows:

1. **Soft attention:** Soft attention is more focused on regions or channels. It is deterministic attention that can be generated through the network directly after the learning is completed. The key point is that soft attention is differentiable, and attention that can be differentiated can then be used to calculate the gradient through the neural network and then learn to get the weight of the attention through forward propagation and backward feedback.
2. **hard attention:** Hard attention is more point-focused, meaning that every point in an image has the potential to extend attention, while hard attention is a stochastic prediction process with more emphasis on dynamic change. Crucially, hard attention is non-differentiable and its training process often relies on reinforcement learning.
3. **self-attention:** It is widely used in the field of NLP for text processing. The self-attention mechanism is a variant of the attention mechanism, which relies less on external information and is better at capturing the internal relevance of data or features.

3.1.1 Self-attention

Because the attention mechanism utilized in this thesis is self-attention, this subsection will present the principle of self-attention in detail.

Self-attention mechanisms are a special form of attention mechanism in which the attention weights are related to their own sequence. Whereas the basic attention mechanism learns the attention of the entire contextual knowledge, the self-attention mechanism preserves the sequential information of the context and captures the relationships between sequence elements. As shown in the Fig3.2 and equation below, three matrices of the same dimension comprise the input: Query, Key, and Value. First, Query and Key are multiplied. To avoid the result being too large, the matrix is scaled by dividing it by the constant value of the dimension. The normalized weight vector matrix is obtained using the softmax function, and then matrix multiplication with Value is performed to obtain the weighted matrix.

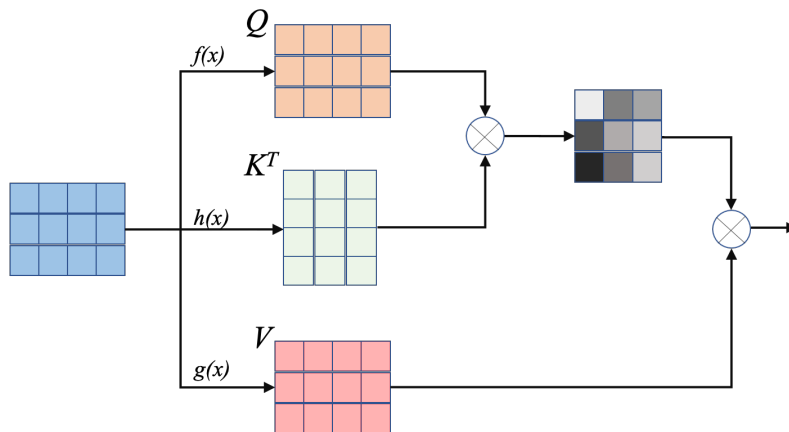


Figure 3.2: The architecture of self-attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (3.4)$$

Using translated text as an example, the self-attention mechanism can capture the correlation between the next word and the previous part. In the fig3.3 below, the next word is in red, and the shade of blue indicates the degree of association. The darker the color, the greater the association and vice versa.

The FBI is chasing a criminal on the run.
The FBI is chasing a criminal on the run.
The FBI is chasing a criminal on the run.
The FBI is chasing a criminal on the run.
The FBI is chasing a criminal on the run.
The FBI is chasing a criminal on the run.
The FBI is chasing a criminal on the run.
The FBI is chasing a criminal on the run.
The FBI is chasing a criminal on the run.
The FBI is chasing a criminal on the run.

Figure 3.3: The visualization of the correlations between words by self-attention[8]

In the field of recommendation systems, the principle of using self-attentive mechanisms

to make recommendations is similar to the principle of text translation mentioned above. The next recommended item can be seen as the next word to be translated, and therefore the relevance of this item to previous items in the sequence can be calculated by a self-attentive mechanism. An item with a high relevance will have a greater influence on the recommended item. Therefore, the use of the self-attentive mechanism leads to better recommendation results than other methods, and at the same time, the results are better interpretable.

Due to the absence of serial training, the self-attentive mechanism can parallelize the computation of the attention weight matrix, significantly reducing the training time of the model when compared to RNN-like models. Simultaneously, because each sequence element's attention weights are computed in conjunction with all other sequence elements, the path length can be considered to be 1, regardless of how far apart the two elements are in the sequence, allowing the attention mechanism to effectively solve the long-term dependency problem in long sequences.

3.2 Transformer

Before the emergence of **Transformer** [37], the mainstream algorithm for sequence data processing was RNN, which processed one node in a sequence at a time and came with a hidden vector to remember the previously processed nodes and other information, and modify the hidden vector when processing the current node. While the RNN is well-suited to processing sequential data on its own, its limitations are self-evident: Firstly, it is difficult to parallelize the nodes in the sequence because the RNN processes them sequentially; secondly, it is difficult to retrace the information of nodes that are far away from the current node, a problem referred to as the long path dependency problem. The proposal of the transformer solves both of these problems, and at the heart of the transformer is the self-attention mechanism. Fig3.4 demonstrates the architecture of the Transformer.

The first is the embedding of the Transformer, which uses not only word embedding but also positional embedding to represent the position of the word in the sentence. Because the Transformer does not use the structure of an RNN, it uses global information and cannot make use of the sequential information of words, which is important for NLP. Therefore, position embedding is used in the Transformer to preserve the relative position or absolute position of words in a sequence. The following formula is used in Transformer to calculate the position, where pos denotes the position, and i represents the dimension. Each position's dimension corresponds to a sinusoidal curve whose wavelengths form a geometric series ranging from 2π to $10000 \cdot 2\pi$. It makes it simple for the model to learn relative position easily since for any fixed offset k , PE_{pos} can be expressed as a linear function of PE_{pos+k} .

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d}\right) \quad (3.5)$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d}\right) \quad (3.6)$$

The next is the main part of the Transformer. This part of the Transformer consists of an encoder, which converts the input sequence into a hidden representation, and a decoder, which converts the hidden representation into an output sequence. Inside the encoders, there are six

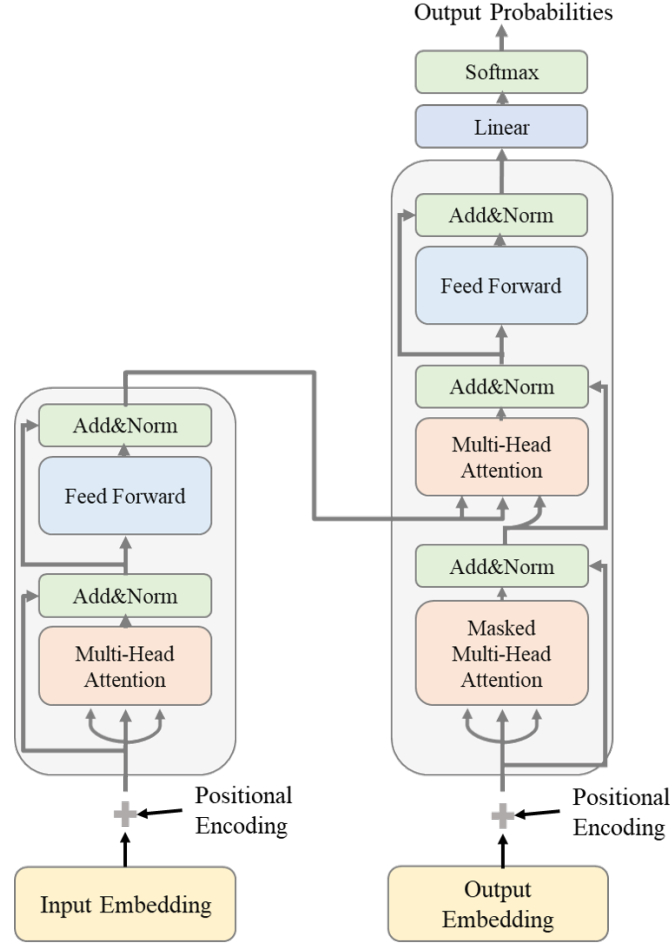


Figure 3.4: The architecture of Transformer[37]

small encoders. Similarly, inside the decoders, there are also six small decoders. The input of each small encoder is the output of the previous small encoder, while the input of each small decoder is not only the output of its previous decoder but also the output of the encoder. Each small encoder has a multi-head self-attention layer and a feed-forward network, and they are each followed by an Add&Norm layer. The Add&Norm layer and the feed-forward layer can be represented by the following formula, where X is the input embedding.

$$X = \text{LayerNorm}(X + \text{MultiHeadAttention}(X)) \quad (3.7)$$

$$X = \text{LayerNorm}(X + \text{FeedFoward}(X)) \quad (3.8)$$

Add stands for Residual Connection to prevent multi-layer network degradation and the issue of vanishing gradients, and Norm stands for Layer Normalization to normalize the activation values at each layer to speed up convergence. The Feed Forward layer is relatively simple, being a two-layer fully connected layer with *Relu* as the activation function for the first layer and no activation function used for the second layer. Different from the encoder, each small decoder has two multi-head attention layers and a feed-forward network, but each

part, like the encoder, also uses an Add&Norm layer. After the last decoder has processed the features, the result is then fed into a linear transformation layer and a softmax layer, where the linear transformation layer transforms the dimensionality of the decoder output vector to the dictionary length, a step similar to the initial word embedding operation. This vector is then probabilised by a subsequent softmax layer, and the result is the probability distribution of each input word over the target word.

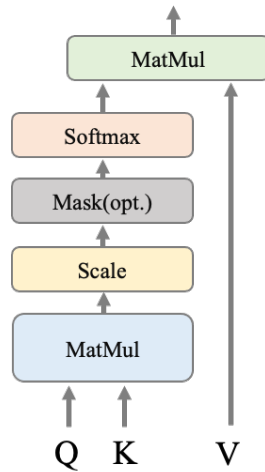


Figure 3.5: Scaled Dot-Product Attention[37]

As introduced in the previous section, the structure of self-attention can also be represented by fig 3.5. Self-attention is computed using the matrices Query, Key and Value. In practice, the input of the self-attention part is a matrix consisting of the word representation vectors, or the output of the previous encoder block. The Query, Key and Value are obtained by linear transformation of the input to the self-attention part.

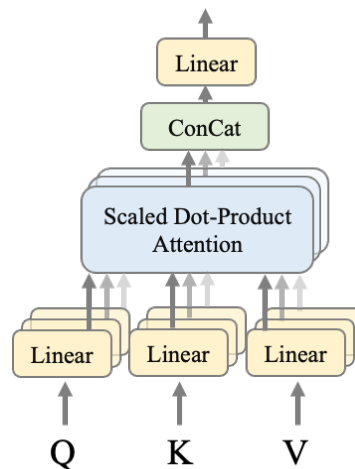


Figure 3.6: Multi-Head Attention[37]

To further refine the self-attentive mechanism layer, the researchers introduced the concept of a multi-headed attention mechanism, which provides multiple representation sub-spaces for the self-attentive layer. In fig 3.6 We have not just one set of Q, K, V weight matrices for the multi-headed self-attentive mechanism, but multiple sets, which means that each encoder and decoder uses multiple non-interfering self-attentive mechanism operations, each with a different Q, K, V . Following that, multiple distinct weight matrices Z are obtained, each of which is used to project the input vector into a distinct representation subspace. Following the multi-headed attention mechanism, we obtain multiple weight matrices Z , and we stitch the multiple Z 's together to obtain the output of the self-attention layer.

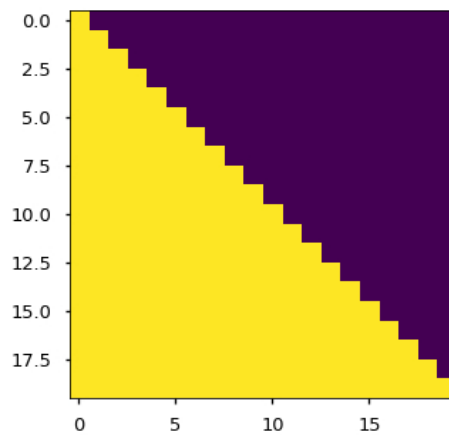


Figure 3.7: Mask matrix

As can be seen in Fig 3.7, the first multi-head attention layer of decoder adopts the operation of *Mask* because the translation process is sequential. For example, the s^i word is translated before the s^{i+1} word can be translated. Hence, the *Mask* operation prevents the i^{th} word from knowing information after the $i + 1^{th}$ word. The second multi-Head attention in the decoder is very similar to the one in the encoder. The primary difference is that the decoder calculates the K, V matrices using the encoder's encoded information matrix C . Then, Q is calculated using the previous decoder block's output Z , or the input matrix in the case of the first decoder. This has the advantage that when the encoder is decoded, each item can make use of the information contained in all of the encoder's items.

3.3 Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of feed-forward neural network that incorporates convolutional computation and has a deep structure. Research on CNN dates back to the 1990s, and **LeNet-5** [20] is often considered to be the first convolutional neural network to emerge. Fig 3.8 demonstrates the structure of **LeNet-5**. In recent years, with the rapid development of deep learning, CNN has become one of the most representative algorithms. CNN was initially used extensively in image processing, but has now found applications in a variety of other fields, including computer vision, natural language processing, and recommendation systems.

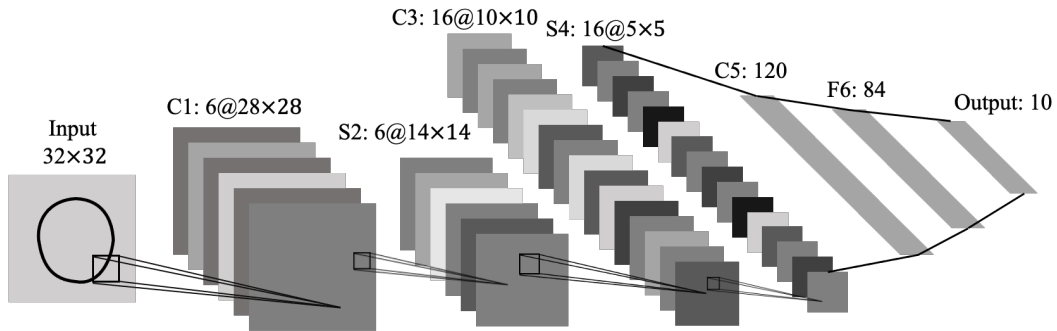


Figure 3.8: A full convolutional neural network [20]

Prior to the advent of CNN, image processing was a difficult problem for two reasons: first, the amount of data to be processed in images is too large, resulting in high costs and inefficiency. Second, it is difficult to retain the image’s original characteristics during digitization, resulting in poor image processing accuracy. CNN solves the first problem by down-scaling a large number of parameters into a small number of parameters and then performing the processing. CNN addresses the second issue by preserving the image’s features in a vision-like manner. Additionally, it is effective at recognizing similar images that have been flipped, rotated, or shifted in position.

As shown in fig 3.8, CNNs are multi-layer neural networks, with each layer consisting of multiple 2D planes, each of which in turn contains multiple independent neurons. A complete CNN model can be constructed using a combination of convolutional, pooling and fully connected layers. The convolutional and pooling layers are the two most critical network layers in a CNN, and they are described below.

1. **Convolutional Layer:** Convolution is used in CNNs to extract specific features. The fig3.9 depicts a simple convolution kernel performing a convolution operation on an input image to generate an output feature map.

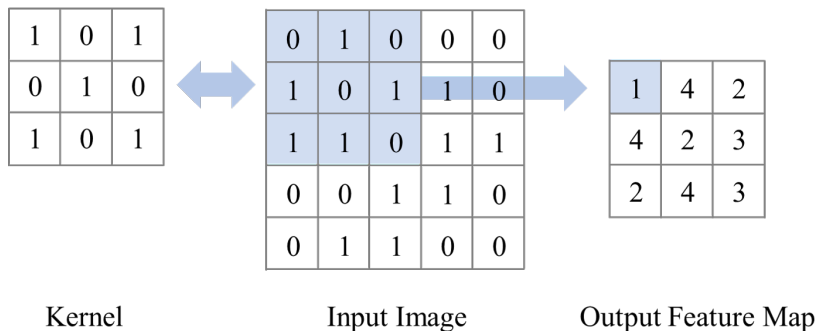


Figure 3.9: A simple operation of convolution in CNN

The CNN’s convolutional layer is the most critical component. Each element in the output feature map is calculated from the pixels in each blue region of the input image

and the convolution kernel in fig 3.9. Thus, the convolution kernel can be seen as a filter. Typically, a convolutional layer typically has multiple convolutional kernels, and the features extracted vary in size from one convolutional kernel to another. The most important feature of convolutional layers is their ability to learn features. In general, the first convolutional layer extracts only a few low-level features from the original data, after which the network's deeper layers can iteratively extract more complex features from the low-level features.

2. **Pooling Layer:** Due to the fact that the feature maps extracted from the convolutional layer contain a great deal of redundant information, the pooling operation is used to remove the redundant data and retain the most fundamental and important information. This not only reduces the size of the feature map, but also enables the reduction of its dimensionality. Similar to convolution, pooling is accomplished through window sliding, except that no windows overlap during pooling. Max pooling, average pooling, and sum pooling are three basic types of pooling. Fig 3.10 depicts a simple max pooling procedure.

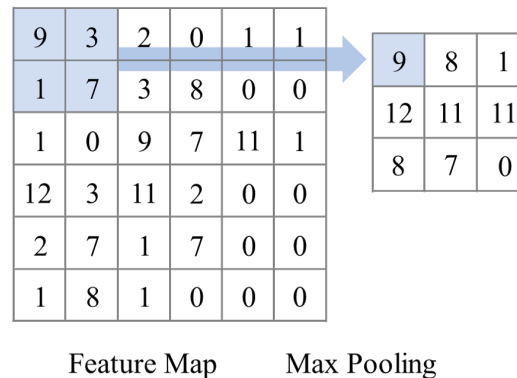


Figure 3.10: A simple operation of pooling in CNN

To avoid the issues associated with traditional neural networks, such as excessive parameters, loss of inter-pixel information, and limited network depth development, CNNs are not fully connected but rather organized as image matrices, and concepts such as local receptive fields, shared weights, and spatial sub-sampling have been introduced to significantly boost their performance:

1. **Local Receptive Fields:** Rather than connecting each neuron in CNNs to all neuron nodes in the subsequent layer, as is the case with traditional neural networks, each neuron is connected to only a subset of them, resulting in a much smaller weighting parameter. This is because, in images, it is typically the local pixels that are more connected to one another, while the pixels further away are less significant. As the network levels increase in depth, the deeper network continues to extract local information about the image's previous layer, eventually yielding global information about the image.
2. **Shared Weights:** A group of connections or multiple groups of connections in CNNs can each share the same weight parameter or the same convolutional kernel, rather than each connection having a separate weight.

3. **Spatial Sub-sampling:** In CNNs, Spatial sub-sampling techniques are used to compress the image data that would otherwise be input to the larger convolutional layer, resulting in a reduction in the total number of pixels in the output. This eliminates the possibility of overfitting due to an excessive number of weighting parameters and increases computational speed further by compressing the image space.

In conclusion, CNNs have several distinct advantages due to their unique structure of local weights sharing. Local receptive fields can help preserve the correlation information between image pixels, facilitate the extraction of higher-dimensional features, and perceive more information in the image, and then, after weight sharing and sub-sampling, the number of network parameters can be further reduced and the model's robustness improved.

CNN has also been utilized by researchers in recommendation systems. As mentioned above, **Caser** [34] is a classic CNN-based sequential recommendation algorithm. This model treats the input item embedding as an image, and utilizes both vertical and horizontal kernels to extract different features. In fig 3.11, the basic structure of **Caser** is demonstrated. The grey rectangle represents the vertical convolution filter and the red, green and blue rectangles denotes horizontal convolution filters of different sizes. After the operation of Max Pooling on the output of different filters, the output features will be concatenated, and then the probability of the next recommended item can be obtained through the fully connected layer.

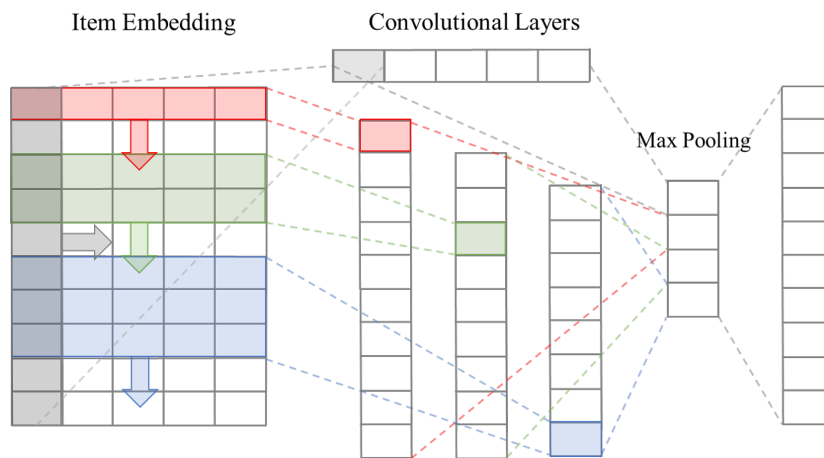


Figure 3.11: The basic structure of **Caser** [34]

3.4 Wavenet

Although recurrent neural networks (RNNs) and convolutional neural networks (CNNs) have yielded good results for time series modelling problems, they both have their limitations. **Wavenet** [36] was originally used in the field of raw audio wave-forms generation. The main component of this model is the casual convolution network, and it can be seen as a representative of CNNs applied to time series because it follows the order of the data. Compared to CNN and RNN, **Wavenet** has the following advantages:

1. Unlike RNNs, where subsequent time steps' predictions must wait for previous time steps' predictions to be completed, convolution can be performed in parallel because the same filter is used for each layer.
2. Compared to standard CNNs, **Wavenet** has the flexibility to adjust the size of its receptive field.
3. The back-propagation of **Wavenet** is significantly different compared to RNN, and it can effectively mitigate and avoid the gradient explosion and gradient vanishing by introducing residual connections and skip connections.

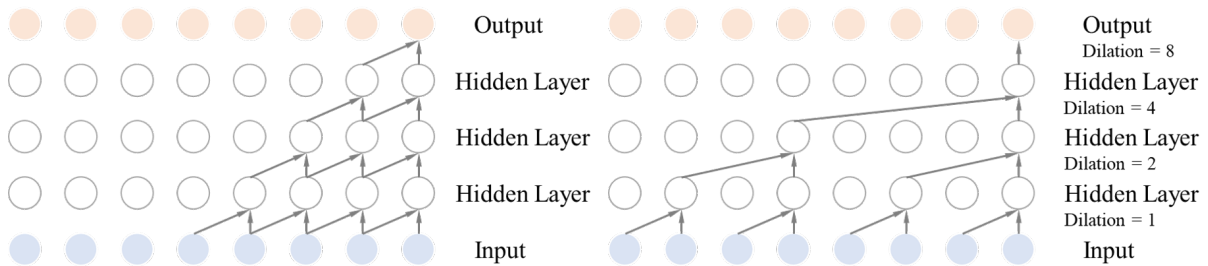


Figure 3.12: Causal convolution

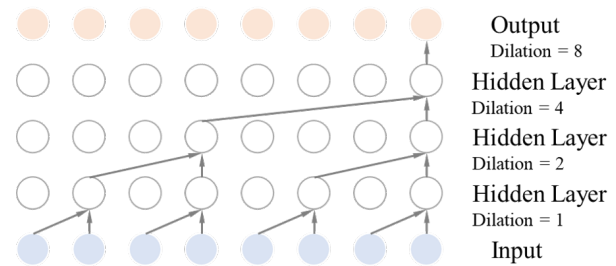


Figure 3.13: Dilated causal convolution

Fig 3.12 and fig 3.13 show the diagram of causal convolution, it ensures that the model output follows the order of data. That is, the model's output at time t will not depend on data from any future time step. However, if the standard causal convolution shown in fig 3.12 is used, in order to process long range sequence, the model should have a sufficiently large receptive field, which can only be achieved by increasing the depth of the network and the size of the convolution kernel linearly, however this will bring a rapid increase in the number of parameters. Hence, the dilated causal convolution shown in fig 3.13 can solve this problem easily by adjusting the dilation factor d . When the $d = 1$, the dilation causal convolution equals to the standard causal convolution, and when d is larger than 1, the convolution kernel is spaced $d - 1$ positions apart when finding the next convolution position, which allows the receptive field to grow exponentially. Because the dilated convolution is utilized, each layer will use padding and the size of padding is $(k - 1) * d$, where k denotes the size of kernel.

Although adding layers increases the richness of the CNN's ability to capture features at different levels, simply increasing the number of layers can lead to degradation of the network. Therefore, **Wavenet** solves this problem by introducing residual connection and skip connection. Residual connection converts learning a constant mapping function to learning a residual function $F(x) = H(x) - x$ by incorporating a constant mapping across layers of connectivity, which is a good solution to the problem of gradient and network degradation. Finally, **Wavenet** uses convolutional layers instead of fully connected layers, making the input and output dimensions consistent.

Chapter 4

Method

In this Chapter, we first introduce and formulate our proposed model **Ti-SACNN**, and then each component of the model is described in detail.

4.1 Proposed method

In this project, in order to capture users' dynamic preferences, we proposed a long- and short-term sequential recommendation model based on time-aware self-attention mechanism, **Ti-SACNN**. As shown in fig 4.1, the architecture of **Ti-SACNN** can be divided into three main modules:

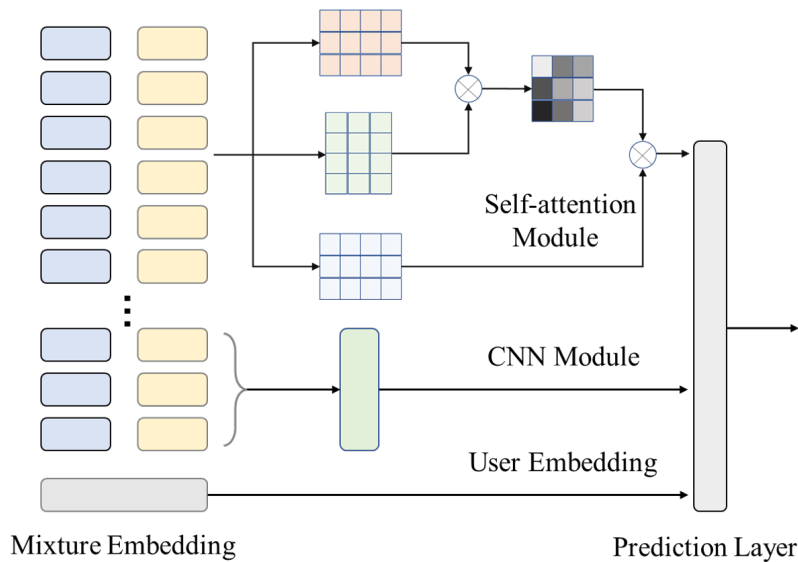


Figure 4.1: The overall network architecture of Ti-SACNN

The first module is the embedding module, in this part, we not only model the absolute position, but also take time intervals into account. The blue and yellow rectangles in the diagram represent the item embedding and the timestamp embedding respectively. Based on the time interval matrix, we create two different relative position embeddings. The first one

makes direct use of the time interval matrix, while the second one is a time interval vector after applying *log* transformation and scaling. The second module is the self-attention part based on the time-aware multi-layer self-attention structure, this module mainly focuses on capturing the long-term preferences of users in the mixture embedding of items and different temporal representations. The last module is the CNN module, which utilizes the characteristic of the convolutional filter to capture the user’s short-term preference from the recent interactions, and we use the green rectangle to represent the vertical convolutional filter in the diagram. In the last step, we concatenate the output of the self-attention module and CNN module with the user embedding, and then input it into the prediction layer to generate the top scoring recommendation items.

4.2 Problem formulation

The related notation is introduced in Table 4.1. Let U be the users’ set, and I be the items’ set. Given an user $u \in U$, his/her historical interaction sequence is denoted as S^u . Each interaction S_i^u in the sequence consists of the interaction item I_i^u and the corresponding timestamp T_i^u . In each time step t during our training process, the model generates the recommended items based on the previous t items, and absolute and relative positions. For the self-attention module, the input are the whole sequence $S^u = [S_1^u, S_2^u, \dots, S_{|S^u|-1}^u]$, the time interval matrix M^t , the absolute position matrix P^u and relative position matrix V^u , while for the CNN module, the input are the latest L interactions $S^u = [S_{|S^u|-L}^u, S_{|S^u|-L+1}^u, \dots, S_{|S^u|-1}^u]$. We expect the model’s output to be $S^u = [S_2^u, S_3^u, \dots, S_{|S^u|}^u]$.

Table 4.1: **Notation**

Notation	Description
U, I	User and item set
S^u	History sequence for user u
I^u	Interaction sequence of user u corresponding to S^u
T^u	Timestamp sequence of user u corresponding to S^u
P^u	Absolute position matrix
d	Dimension of latent vector
n	Maximum length of S^u
M^I, E^I	Item matrix and input item embedding
M^t, E^t	Time interval matrix and time matrix embedding
V^u	Relative position matrix
d^T	Maximum scaling range of the time interval
E_K^P, E_Q^P	Position embedding for <i>key</i> and <i>query</i>
$E_K^{Absolute}, E_Q^{Absolute}$	Absolute position embedding for <i>key</i> and <i>query</i>
$E_K^{Relative}, E_Q^{Relative}$	Relative position embedding for <i>key</i> and <i>query</i>
L	The number of interactions in CNN module
m	The number of vertical filters
c	The output of the vertical filter
F_a, F_c	The outputs of the self-attention and CNN module

4.3 Self-attention with temporal information

In many previous works, they have incorporated absolute position into the self-attention mechanism. For example, in **Transformer** [37], because it does not utilize the RNN’s structure but rather global information, it is unable to use the sequential information contained in words, which is essential for NLP. As a result, positional embedding enables the **Transformer** to determine the sequence’s order. As mentioned in Chapter 1, relative position is also a critical factor for sequential recommendation. Hence, in this project, we propose an enhanced version of self-attention that takes into account not only the absolute position but also the relative position.

4.3.1 Position modelling

Assume the input sequence is denoted as $S^u = [S_1^u, S_2^u, \dots, S_{|S^u|-1}^u]$, S_i^u can also be represented by (I_i^u, T_i^u) , where I_i^u denotes the i^{th} interaction item of the user and T_i^u is the corresponding timestamp. Since the length of different sequences varies, we need to set a hyper-parameter to regulate the length of the longest input sequence, and it is denoted as n in our experiment. Then, we convert the input sequence into the sequence which has a fixed length of n . If the length of the sequence exceeds n , then the last n interactions of the sequence are considered. If the length of the sequence is less than n , then padding items are added repeatedly to the left of the sequence. For I^u , the padding item is zero while for T^u , the timestamp of the first interaction is considered as the padding item.

Following **SASRec** [18], in order to help the model identify the order of the items, we define the absolute position for each sequence as $P^u = [1, 2, \dots, n]$. Inspired by **TiSASRec** [21], we model the time interval as follows. First, for each pair of adjacent interactions, the interval is denoted as $\Delta t_i^u = t_{i+1}^u - t_i^u$. Then, we take the minimum interval as Δt_{min}^u , and divide each interval by it. Hence, we can get the time interval matrix $M^t \in \mathbb{R}^{n \times n}$ of the user u , where $m_{ij}^t \in \mathbb{N}$. We keep the first row of M^t and denote it as $R^t \in \mathbb{R}^n$, and then we use the same method as in **TiSASRec** to process M^t .

$$\mathbf{M}^t = \begin{bmatrix} m_{11}^t & m_{12}^t & \dots & m_{1n}^t \\ m_{21}^t & m_{22}^t & \dots & m_{2n}^t \\ \dots & \dots & \dots & \dots \\ m_{n1}^t & m_{n2}^t & \dots & m_{nn}^t \end{bmatrix} \quad (4.1)$$

In the next step, for each interval in R^t , we perform a logarithmic operation on it. In our analysis, by transforming the temporal information in different ways, the model can capture more useful patterns from the user’s sequence. At last, we map the value of each time interval to the range $[0, d_T]$ and get the scaled time interval sequence $V_u = [v_1, v_2, \dots, v_n]$, where d_T is the maximum value after mapping and v_i denotes $Scaled(\log(\frac{\Delta t_i^u}{\Delta t_{min}^u}))$.

4.3.2 Embedding layer

Fig 4.2 shows the structure of the self-attention part of our model. We define the embedding of the items as $M^I \in \mathbb{R}^{|I| \times d}$, where d is the size of the latent dimension. Hence, for each

input sequence, the embedding can be denoted as $E^I \in \mathbb{R}^{n \times d}$. For Query and Key embedding, we create two additional learnable positional embeddings $E_K^P \in \mathbb{R}^{n \times d}$ and $E_Q^P \in \mathbb{R}^{n \times d}$ respectively. E_K^P is generated by concatenating the corresponding absolute position embedding $E_K^{Absolute} \in \mathbb{R}^{n \times d/2}$ and relative position embedding $E_K^{Relative} \in \mathbb{R}^{n \times d/2}$. Similarly, E_Q^P is made up of $E_Q^{Absolute} \in \mathbb{R}^{n \times d/2}$ and $E_Q^{Relative} \in \mathbb{R}^{n \times d/2}$, where $E_Q^{Absolute}$, $E_K^{Absolute}$ and $E_Q^{Relative}$, $E_K^{Relative}$ are the linear transformation from P^u and V_u , respectively. For the relative time interval embedding matrix, we get embedding $E^t \in \mathbb{R}^{n \times n \times d}$.

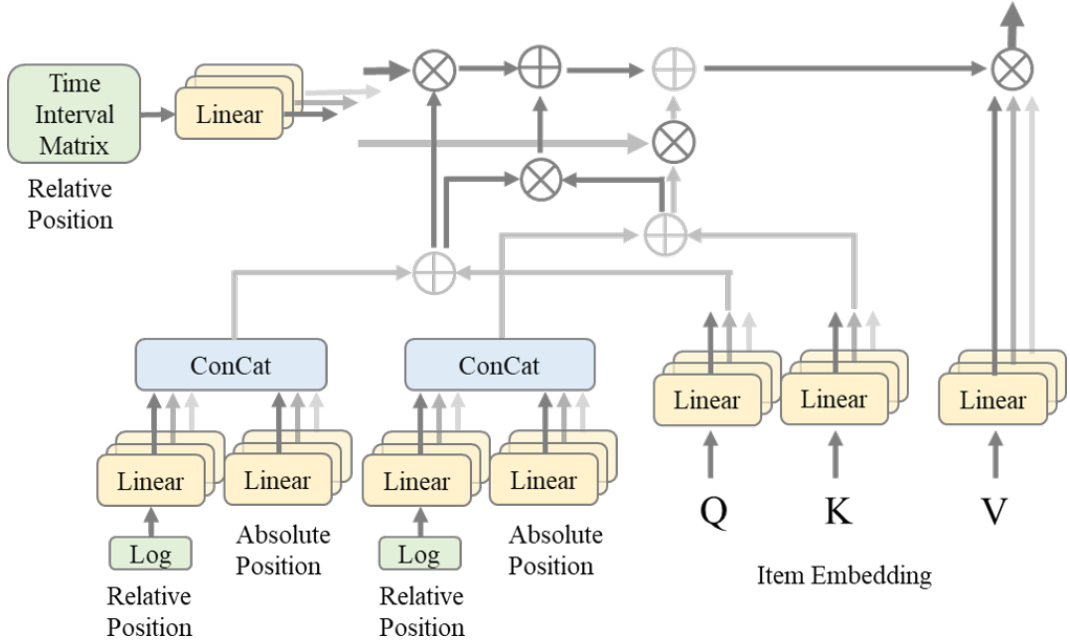


Figure 4.2: Architecture of the mixture self-attention

4.3.3 Temporal self-attention:

As described in section 3.1, the input to the self-attention module consists of *query*, *key* and *value*, and the output is calculated from the weighted sum of *values*. In our model, firstly, as with original self-attention, the same item embedding $E^I = [em_{s_1}, em_{s_2}, \dots, em_{s_n}]$, where $em_{s_i} \in \mathbb{R}^d$, is fed into these three components. Then, concatenating the embeddings of the transformed relative position and absolute position, and adding them to the *key* and *query* embeddings, respectively. Next, we compute the sum of the dot product of the *key* and *query* embeddings and the time interval matrix. Finally, we add them together and after softmax, we can get the output $Z = [z_1, z_2, \dots, z_n]$, where $z_i \in \mathbb{R}^d$.

$$z_i = \sum_{j=1}^n \alpha_{ij} (em_{s_j} W^V) \quad (4.2)$$

where $W^V \in \mathbb{R}^{d \times d}$ is the parameter matrix for *value*, and the coefficient α_{ij} and the output of the dot product e_{ij} is calculated as follows:

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^n \exp e_{ik}} \quad (4.3)$$

$$e_{ij} = \frac{\left(em_{s_i} W^Q + r_{ij}^q + p_i^q \right) \left(em_{s_j} W^K + r_{ij}^k + p_j^k \right)^T}{\sqrt{d}} \quad (4.4)$$

where $W^Q \in \mathbb{R}^{d \times d}$ and $W^K \in \mathbb{R}^{d \times d}$ are the parameter matrices for the *query* and *key*, respectively. Scaling factor \sqrt{d} is utilized to avoid the value of inner product becoming too large.

4.3.4 Feed-forward network:

Inspired by **Transformer** [37], following the self-attention module, we use two linear transformations with a *ReLU* activation function to introduce non-linearity to the model, where $W_1, W_2 \in \mathbb{R}^{d \times d}$ and $b_1, b_2 \in \mathbb{R}^{d \times d}$.

$$FFN(z_i) = ReLU(z_i W_1 + b_1) W_2 + b_2 \quad (4.5)$$

For simplicity, we define the whole self-attention network as a block F . Though a self-attention block F is capable of handling all input embeddings, it is typically the case that a multi-layered network structure captures a more diverse range of features. Hence, we stack the self-attention block, and the k^{th} ($k > 1$) block is defined as:

$$F^k = SAN(F^{k-1}) \quad (4.6)$$

Where $F^k \in \mathbb{R}^{n \times d}$. In addition, Like [18], we employ **LayerNorm**, **Dropout**, and **Residual Connection** to optimize our self-attention module:

1. **LayerNorm:** According to [1]. LayerNorm is a strategy that benefits network training by consolidating and accelerating it.
2. **Residual Connection:** Because deeper networks may have problems such as gradient vanishing and network degradation. Hence, in order to make the training process more stable, we utilize the residual connection following the feed-forward network to convey low-layer features to higher layers via residual connection.
3. **Dropout:** During training, we employ the Dropout technique on the embedding layer, etc., to address concerns about overfitting in deep neural networks.

$$f(z_i) = z_i + Dropout(FFN(LayerNorm(z_i))) \quad (4.7)$$

where $f(z_i)$ represents a set of operations inside a self-attention block. In each block, we firstly employ the LayerNorm on z_i before feeding into the feed-forward network, then the Dropout is applied on the output, and residual connection is utilized by adding z_i .

4.3.5 Wavenet

Inspired by the characteristic of **Wavenet** which is introduced in Chapter 3, we propose another solution to better optimise the model. Because the feed-forward network is simply a two-layer network, it is limited in its ability to fuse and extract features. Hence, in this solution, we innovatively utilize **Wavenet** to replace the feed-forward network.

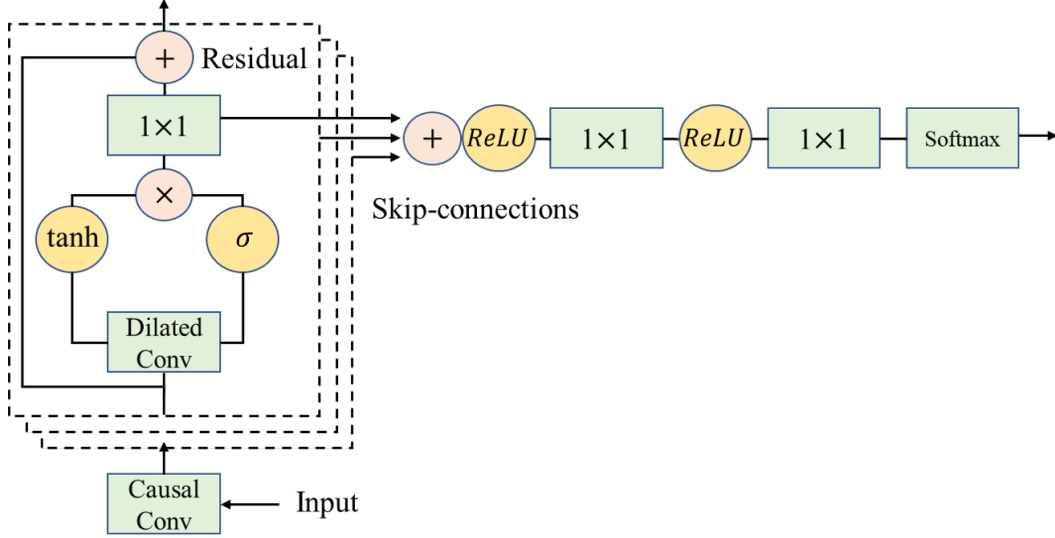


Figure 4.3: The layout of Wavenet

Fig.4.3 shows the layout of **Wavenet**. As described in Chapter 3, the core of **Wavenet** is the casual convolution network and dilated convolution network. By using the dilated convolution, the perceptive field of the model becomes larger, thus escaping the limitations of the fixed size of the standard CNN filter. At the same time, the model has the ability to handle longer sequences. Linked after the causal convolution are the gated activation units:

$$y = \tanh(W_{f,k} * x) \odot \sigma(W_{g,k} * x) \quad (4.8)$$

where $*$ and \odot represents the convolution operation and element-wise multiplication operation, respectively. k denotes the layer, and W is the weight matrix. f and g are filter and gate, respectively. The value domain of the activation function is required from -1 to +1, so the \tanh is adopted here, and σ is utilized to control the wave amplitude. In the end, the output of the **Wavenet** can be represented by the following formula, where the input $z = [z_1, z_2, \dots, z_n]$.

$$f(z) = \text{Softmax}(\text{ReLU}(\sum_{k=1}^l \text{WaveNet}(\text{LayerNorm}(z^k)))) \quad (4.9)$$

4.4 Short-term preference modelling

Recall our previous introduction about **Caser** [34], which utilises the convolutional neural network to treat item embedding as 'image'. Unlike the 3×3 and 5×5 convolutional kernels

that are widely used in images processing, **Caser** is designed with two special filters, a vertical filter and a horizontal filter. They have the same height and width as the embedding, so that the whole embedding can be covered by sliding the filters to capture the different features. However, experiment reveals that the horizontal filter of **Caser** not only contributes little to the final result, but also greatly increases the model’s computing cost. It is assumed that because the horizontal filter employed max-pooling to uniform the dimension of the final feature map, some critical features had their positions ignored.

Although in **Caser** the model can only capture the user’s short-term preferences based on their last few interactions due to filter size limitations, this is exactly what our model needs. By combining the long-term preferences captured by the self-attention module with the convolutional neural network’s short-term preferences, and with the help of temporal features, our model is well suited to modelling the dynamic preferences of users.

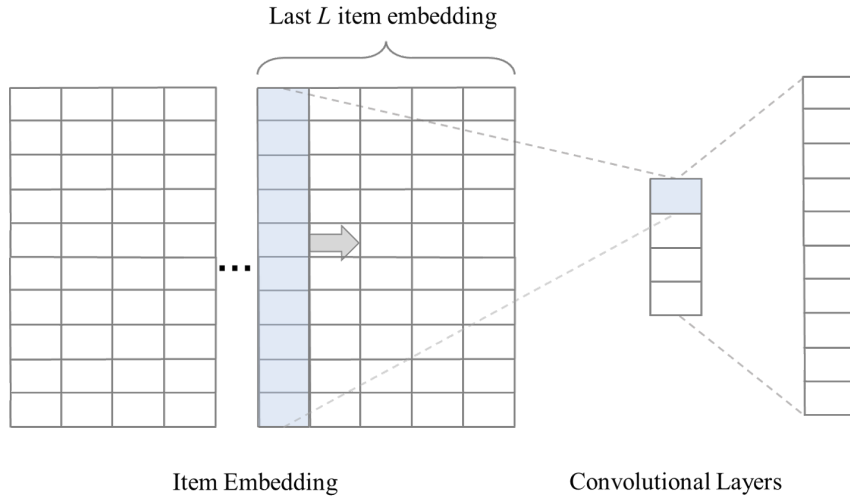


Figure 4.4: The architecture of the convolution module

Due to the drawbacks of horizontal filters, our model uses only vertical filters to capture users’ short-term preferences. Fig 4.4 depicts the architecture of our convolution module. Suppose that we denote the vertical filter as $f_v^m \in \mathbb{R}^{L \times 1}$, where L and m are hyper-parameters to set the number of recent interactions and the number of filters, respectively. Each vertical filter has a size of $L \times 1$ because each column of E^I is implicit, hence, dealing with several subsequent columns together is pointless. The filter f_v slides from left to right on the item embedding E^I column by column to generate the result c :

$$c^m = [c_1^m c_2^m \dots c_d^m] \quad (4.10)$$

Therefore, with m different vertical filters the model can aggregate the L previous item embeddings. According to [34], these vertical filters can extract point-level sequential patterns from items’ latent representations. Unlike the horizontal filter, because we need to preserve aggregated values for each hidden dimension, we don’t employ max-pooling here. Therefore, the output of our CNN module is $F_c \in \mathbb{R}^{m \times d}$.

$$F_c = [c^1 c^2 \dots c^m] \quad (4.11)$$

4.5 Prediction layer

After getting the output $F^a \in \mathbb{R}^{n \times d}$ of the self-attention module and the output $F^c \in \mathbb{R}^{n_v \times d}$ of the CNN module respectively, we concatenate them together with user embedding $E^u \in \mathbb{R}^d$. For the self-attention module, because our goal is to recommend the next item, hence we only need output's last dimension $F_n^a \in \mathbb{R}^d$ to compute the score for long-term preference. Because the CNN module is designed to capture user's recent intentions, so F_c is used to compute the score the short-term preference. The incorporation of user embedding E^u can make the recommendations more personalized.

$$r_I = \text{Concat}([F_n^a; F^c; E^u])M_I^T \quad (4.12)$$

where r_I is the score of the candidate item I . In conclusion, after training the model, we take the item and user embeddings, as well as relative and absolute position embeddings, and split the latest L item embedding as the model input, and then provide N recommended items with the highest values in the output layer for the user u at time step t .

4.6 Network training

In this project, we adopt negative sampling to optimize the output score r_i . For example, we take a sequence S_{train}^u of user u from the training set, and then we take the last $n + 1$ interactions to generate the input sequence and the positive sequence, where the input sequence is the first n interactions and the positive sequence is the last n interactions. Next, we generate a negative sampling sequence of size n , in which each item never appears in this sequence. Hence, we can provide a pair of positive sample o_p and negative sample o_n to train each input item at time step t , and we utilize the binary cross entropy loss as the loss function and employ *Adam* optimizer to optimize our model:

$$- \sum_{S_{train}^u \in S} \sum_{t \in [1, 2, \dots, n]} [\log(\sigma(r_{o_p, t})) + \log(1 - \sigma(r_{o_n, t}))] + \lambda \|\Theta\|_2^2 \quad (4.13)$$

In the above equation, $\Theta = \{E^I, E^t, E_K^P, E_Q^P\}$ represents model parameters. $\sigma(\cdot)$ is the sigmoid function, and λ is the regularization parameter. We utilize l_2 norm to limit the model's complexity.

Chapter 5

Experiment and Result Discussion

In this project, we conduct comprehensive experiments to gain a better understanding of the model’s performance in various scenarios. In this chapter, we will firstly introduce our experimental setup and then perform a thorough analysis of the experimental results based on the research questions that we designed.

1. **Q1:** How does the model with the improved temporal self-attention mechanism that we propose perform compared to the baseline model?
2. **Q2:** How does our model which considers both long- and short-term preferences perform compared to the baseline model?
3. **Q3:** How does our model with Wavenet instead of feed-forward network perform compared to the baseline model?
4. **Q4:** What effect would changes in the parameters have on the model’s efficiency? For example, the number of the self-attention blocks.

5.1 Experiment setup

5.1.1 Dataset

In our experiments, because the temporal information is necessary, hence, four datasets which contain timestamp attribute from three real-world domains are used to evaluate our approaches. Brief description about the datasets is listed as follows:

1. **MovieLens:** It is a famous and widely used benchmark in the field of recommendation systems. MovieLens mainly keeps a large database of user reviews of movies, as well as information about the movies themselves. Here we adopt Movielens-1M to evaluate our model.
2. **Amazon:** This is a dataset from the well-known online e-commerce platform Amazon. This dataset comprises a large number of product reviews written by users. Here we adopt two sub-categories, ‘Beauty’ and ‘Video Games’.
3. **Gowalla:** This is a check-in dataset from the gowalla website, which records a large amount of geolocation-based social information about users and is often used in the field of recommendation systems.

In all datasets, we treat the review or rating part as implicit feedback and sort the items by timestamps. For the MovieLens datasets, we follow the same preprocessing method from [18], and for the Amazon dataset, we filter out users with at least 10 interaction items and the items with no fewer than 5 interactions for experiments. For the Gowalla dataset in our experiment, we take the data in the last three months, and remove items with fewer than 20 interactions. Table 5.1 summarizes the statistics for all datasets after preprocessing. From the statistics in Table 5.1, we can infer that the Amazon Beauty and Game are two sparse datasets and compared with these two datasets, the MovieLens is much denser.

Following [18], for all datasets, we use the last interaction of the user’s sequence to be the test set, and we take the penultimate interaction as the validation set. The remaining part in the sequence is considered to be the training set.

Table 5.1: **Basic dataset statistics**

Dataset	#users	#items	avg. actions /user	#actions
MovieLens-1M	6040	3416	163.50	0.97M
Amazon Beauty	51071	18784	5.81	0.30M
Amazon Game	30866	11212	8.04	0.25M
Gowalla	6161	4440	18.95	0.12M

5.1.2 Baselines

To demonstrate the efficacy of our proposed algorithm in this project, it is compared with the following two baselines:

1. **SASRec** [18]: It is the first sequential recommendation model that only based on self-attention mechanism, and it outperform a number of well-known algorithms at the time it was proposed. The key to **SASRec**’s success is the adoption of a similar structure to **Transformer**.
2. **TiSASRec** [21]: It is actually an improved algorithm that based on **SASRec**. The most significant improvement is the incorporation of temporal information in the self-attention mechanism.

Both **SASRec** and **TiSASRec** can be viewed as the algorithms that only consider long-term preference. Here we don’t perform the comparison with other baseline models in sequential recommendation such as **Caser** [34], because they have been outperformed by **SASRec** [18], as well as **TiSASRec** [21] in the previous studies.

5.1.3 Evaluation Metrics

We evaluate the performance of the models using the following metrics, which are extensively utilized in related studies.

1. **HR@k**: The Hit-Ratio(HR) is a commonly used metric for recall and it observes whether the expected item exists in the top-k list generated by the recommendation system. The HR@k is calculated as follows:

$$HR@k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \mathbf{1}(r_u \leq k) \quad (5.1)$$

where r_u denotes the rank of the user u and $\mathbf{1}(x)$ is an indicator function. When x is true, the result of the function will be 1, otherwise 0.

2. **NDCG@k**: The normalized discounted cumulative gain(NDCG) calculates hit positions by assigning higher scores to the top rankings. In simple terms, this metric is mainly concerned with whether the expected items are in a more prominent position among all recommended items for the user, emphasizing the importance of the order. Hence, if the expected item ranks higher in the recommended list, the score of NDCG@k will be higher.

$$NDCG@k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{2^{r_u \leq k} - 1}{\log_2(r_u + 1)} \quad (5.2)$$

where the denominator of the equation $\log_2(r_u + 1)$ gets closer to 1 when the ranking gets higher.

5.1.4 Parameter Settings

In our experiment, for fair comparison, we use the author’s open source code for both two baselines, and we follow the same parameter setting reported in the corresponding papers.

Table 5.2: **Hyper-parameter settings**

Dataset	max sequence length	regularization
MovieLens-1M	50	0.00005
Amazon Beauty	20	0.00005
Amazon Game	20	0.00005
Gowalla	25	0.00005

We implement **Ti-SACNN** with *pytorch* based on NVIDIA Tesla P100. We select the size of latent vector in {20, 40, 60, 80, 100}, and the regularization rate is tuned amongst {0.0001, 0.001, 0.01, 0.1}. The learning rate is from {0.1, 0.01,..., 0.0001} and we set it to 0.001. The batch size and dropout rate are set to 128 and 0.2, respectively. The number of self-attention layer is 5, and the remainder of the parameter configurations are listed in Table 5.2. We utilize the validation set to tune hyper-parameters, and stop training after 20 epochs if the performance does not improve.

5.2 Result discussion

In our experiments, we first implement the model with enhanced self-attention mechanism, and then apply CNN module based on this model. Hence, we can observe the effectiveness of the two methods mentioned above, and compare their improvements. The results and analysis of the experiment are as follows:

5.2.1 General performance of Ti-SAN

Table 5.3: Recommendation performance with temporal self-attention

Dataset	Metrics	SASRec	TiSASRec	Ti-SAN	Improvement
MovieLens-1m	NDCG@5	0.5170	0.5386	0.5539	2.76%
	NDCG@10	0.5524	0.5706	0.5887	3.17%
	HR@5	0.6804	0.7101	0.7189	1.24%
	HR@10	0.7929	0.8038	0.8101	0.78%
Gowalla	NDCG@5	0.7709	0.7718	0.7723	0.06%
	NDCG@10	0.7911	0.7911	0.7912	0.01%
	HR@5	0.8724	0.8732	0.8733	0.01%
	HR@10	0.9420	0.9430	0.9417	-0.14%
Amazon Beauty	NDCG@5	0.2538	0.2678	0.2738	2.24%
	NDCG@10	0.2877	0.3016	0.3079	2.09%
	HR@5	0.3385	0.3492	0.3601	3.12%
	HR@10	0.4416	0.4500	0.4617	2.60%
Amazon Game	NDCG@5	0.4252	0.4307	0.4375	1.58%
	NDCG@10	0.4629	0.4717	0.4787	1.49%
	HR@5	0.5585	0.5701	0.5803	1.79%
	HR@10	0.6743	0.6862	0.6967	1.53%

Table 5.3 demonstrates the summary of the best results of the two comparison methods, and **Ti-SAN**, which denotes the **Ti-SACNN** without applying CNN module. The data with best performance of each metric is highlighted in boldface. The data in the improvement column is computed with the best comparison baseline. To answer **Q1**, our first observation is that, the experimental results show that **TiSASRec** can effectively improve the effectiveness of recommendations by introducing relative time intervals on the basis of **SASRec**. Except for Gowalla dataset, Ti-SAN outperforms the best baseline on both metrics, which represents that not only have more target items been selected for the recommendation list, but they are placed in better positions. In particular, our proposed approach achieves the most significant improvement on the Amazon Game dataset among all the datasets. This indicates that some features hidden in the temporal information can be further explored by some transformation methods, and these features will be beneficial in improving the performance of the model. However, the experimental results indicate that both **TiSASRec**, and our improved method are not applicable for the Gowalla dataset. Although the temporal factor may not have an impact on some data sets, it exposes the shortcomings of our approach that the scalability is insufficient.

5.2.2 Influence of the self-attention block

To answer **Q4**, we examine the impact of the number of self-attention blocks by NDCG@10. Fig 5.1 shows NDCG@10 for various number of blocks while keeping other settings unchanged. From the figure we can clearly find that the NDCG@10 of **TiSASRec**(red line) rises and then falls as the number of blocks increases, and reaches a maximum when the number of blocks equals 5. This is because the model achieve the best performance when the number of blocks is set properly, and the worse performance afterwards is due to over-fitting. However, according to the NDCG@10 of **Ti-SAN**(blue dotted line), it rises as the number of blocks increases and eventually tends to saturate. And it exceeds **TiSASRec** at the beginning due to the addition of the user embedding and when the number of the blocks is set to 6, it can approach the maximum. This is because our enhanced self-attentive mechanism introduces more additional temporal features, hence the model requires a deeper network to extract these features.

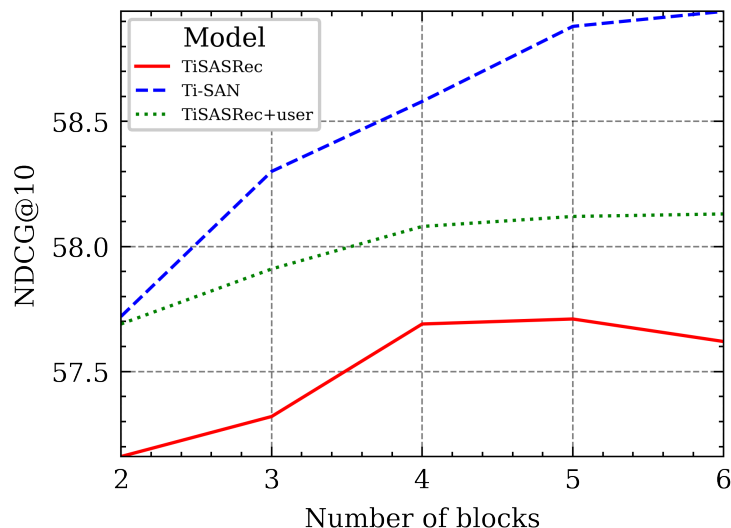


Figure 5.1: Influence of the self-attention block

5.2.3 Influence of the user

In [18], according to the experiments, the researchers claim that the addition of user embedding has no effect on the results because the model is strong enough to include the user features in its item embeddings. However, from our experiments, the user embedding plays an important role in the model. From fig. 5.1, it can be clearly seen that the introduction of the user embedding(green dotted line) can effectively improve the effectiveness of recommendations. Presumably this is because the user embedding contains personalized features that have not yet been learned by the model.

Another function of user embedding is to prevent overfitting of the network as the number of layers grows. Because from our observation in fig. 5.1, after adding user embedding, the overfitting in **TiSASRec** can be alleviated as the number of layers increases. Also, the results

of **Ti-SAN** prove this well. The model becomes easier to overfitting when additional temporal information is introduced, but this problem can be solved by adding user embedding.

5.2.4 Influence of the Wavenet

As introduced in Chapter 4, **Wavenet** can be considered as the CNN applied to time series. The introduction of *hole* can significantly increase the size of receptive field without requiring more layers. Therefore the fixed size of the convolution filter will not limit the algorithm’s ability to process longer input sequences. At the same time, by combining features from each layer allows the algorithm to learn a more comprehensive range of features. Hence, to answer **Q3**, here we use Wavenet instead of the original feed-forward network to endow the model with non-linearity.

Table 5.4: Recommendation performance with Wavenet(SASRec)

Dataset	Metrics	SASRec	SASRec-Wavenet	Improve
MovieLens-1M	NDCG@5	0.5170	0.5276	2.05%
	NDCG@10	0.5524	0.5640	2.10%
	HR@5	0.6804	0.6869	0.96%
	HR@10	0.7920	0.7984	0.81%
Gowalla	NDCG@5	0.7709	0.7769	0.78%
	NDCG@10	0.7911	0.7972	0.77%
	HR@5	0.8724	0.8778	0.62%
	HR@10	0.9420	0.9489	0.73%
Amazon Beauty	NDCG@5	0.2538	0.2548	0.39%
	NDCG@10	0.2877	0.2901	0.83%
	HR@5	0.3385	0.3394	0.27%
	HR@10	0.4416	0.4424	0.18%
Amazon Game	NDCG@5	0.4252	0.4270	0.42%
	NDCG@10	0.4629	0.4650	0.45%
	HR@5	0.5585	0.5516	0.56%
	HR@10	0.6743	0.6763	0.30%

Table 5.4 illustrates the results after replacing the feed-forward network with Wavenet in **SASRec**. Overall, the improved model achieved better performance on all datasets. In particular, on the MovieLens-1M dataset, this model achieves 2.10% improvement on NDCG@10, which is quite significant. However, on the other three datasets, there is only a slight improvement in the model’s recommendation performance. We think this is due to two main reasons. The first reason is that the average length of the sequences in the MovieLens-1M dataset is much longer than in other datasets, and Wavenet has the advantage of handling long sequences, so this method performs better on the MovieLens-1M dataset than the other datasets. The second reason is that the other datasets are more sparse compared to the MovieLens-1M dataset. Therefore, this method also has some limitations.

In table 5.5, we demonstrate the results after replacing the feed-forward network in the two baseline algorithms and our proposed Ti-SAN with Wavenet. From the results we find

that the improvement on TiSASRec is very slight, and there is even a drop on Ti-SAN. So, we can conclude that although this method performs well on SASRec, its performance decreases significantly when the model becomes more complex. Hence, we finally decide not to use this method in our model.

Table 5.5: Recommendation performance with Wavenet

Dataset	Model	NDCG@10	Improve
MovieLens-1M	SASRec-Wavenet	0.5640	2.1%
	TiSASRec-Wavenet	0.5740	0.6%
	Ti-SAN-Wavenet	0.5835	-0.9%

5.2.5 Influence of maximum sequence length

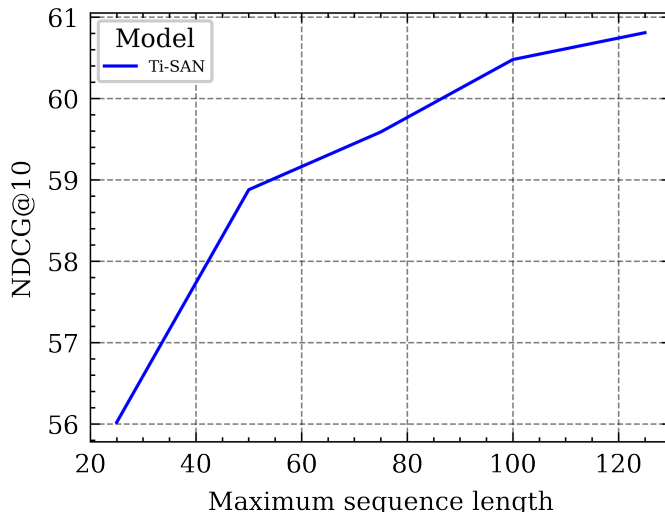


Figure 5.2: Influence of maximum sequence length

In this subsection, we discuss the influence of maximum sequence length. Fig 5.2 shows the NDCG@10 for maximum sequence length $\{25,50,75,100,125\}$, and we keep the other hyper-parameters unchanged. We can find that recommendation performance improves when the sequence becomes longer and gradually converges. And the longer the length, the longer the computation takes, so we select 50 as the default maximum sequence length.

5.2.6 General performance of Ti-SACNN

To answer **Q2**, we summarize the best results of **Ti-SASRec**, **Ti-SAN**, and **Ti-SACNN**. Because Ti-SASRec has outperformed the another comparison baseline mentioned above, here we omit it and only keep Ti-SASRec. At the same time, we also keep the best result of Ti-SAN to see whether the incorporation of the CNN module would make the model generate better recommendations or not. Because the operation of concatenation will change the number of the hidden units in item embedding, hence, for fair comparison, we use two

independent item embeddings instead of shared item embeddings in all the models. While the technique of shared embeddings can be effective in improving the performance of the model, our emphasis here is to examine the effectiveness after incorporating the CNN module. Here we set two columns to show the improvement from the CNN module and the total improvement compared to TiSASRec, respectively. According to the results, we can find that although there is a slight decrease in one row, there is still an improvement for most of them. For the Gowalla dataset, although the introduction of temporal features is not helpful, the CNN module improve the performance significantly. The NDCG@5 and NDCG@10 of the Gowalla dataset increases 1.66% and 1.47%, and this means that more recommended items have better positions. However, this method also has some limitations. Compared to the other datasets, the improvement of the MovieLens dataset is relatively slight, and the this is due to the fact that in this dataset, users' preferences are mainly long-term preferences. Also the improvement of the two Amazon datasets is not ideal, mainly because the short-term preferences are already dominant and adding CNN module does not make much sense. We find that the enhanced temporal attention mechanism that dominates the improvement in recommendation performance. Although the two methods we propose both have their limitations, by combining them we can obtain a more robust sequential recommendation model.

Table 5.6: Recommendation performance with short-term preference

Dataset	Metrics	TiSASRec	Ti-SAN	Ti-SACNN	Improve	Improve (Total)
MovieLens-1M	NDCG@5	0.5070	0.5221	0.5218	-0.06%	2.91%
	NDCG@10	0.5440	0.5569	0.5599	0.54%	2.92%
	HR@5	0.6604	0.6745	0.6780	0.52%	2.67%
	HR@10	0.7747	0.7864	0.7944	1.02%	2.54%
Gowalla	NDCG@5	0.7269	0.7275	0.7398	1.66%	1.77%
	NDCG@10	0.7471	0.7472	0.7582	1.47%	1.49%
	HR@5	0.8424	0.8425	0.8468	0.51%	0.52%
	HR@10	0.9120	0.9102	0.9121	0.01%	0.01%
Amazon Beauty	NDCG@5	0.2272	0.2321	0.2347	1.12%	3.30%
	NDCG@10	0.2548	0.2580	0.2615	1.26%	2.87%
	HR@5	0.3021	0.3061	0.3077	0.52%	1.85%
	HR@10	0.3832	0.3941	0.3993	1.32%	2.84%
Amazon Game	NDCG@5	0.3782	0.3924	0.3928	0.10%	3.86%
	NDCG@10	0.4140	0.4257	0.4287	0.70%	3.55%
	HR@5	0.5020	0.5149	0.5189	0.78%	3.26%
	HR@10	0.6129	0.6294	0.6304	0.16%	2.86%

Chapter 6

Conclusion and Future Work

In this chapter, we provide a summary of this project and possible directions for future improvement are listed.

6.1 Conclusion

In this project, our goal is to propose a robust sequential recommendation algorithm to capture the user’s dynamic preference in a better way. Inspired by **TiSASRec** [21] and **Caser** [34], we propose **Ti-SACNN**, which combines the advantages of both long- and short-term preferences, and additional temporal information to generate more accurate recommendations for the user.

In the first step, we use three public datasets from different domains and preprocess these data according to the previous works. Unlike other studies, we keep the timestamp data, and we set a fixed length for both item and timestamp sequences. Depending on the order of the items, we set the absolute position of the items. Also, based on the timestamp corresponding to the item, we build the time interval matrix and time interval vector. We concatenate the log-transformed interval vector and the absolute position vector, then add them to the item embedding and input them to the *Key* and *Query* of the self-attention block respectively. Then we multiply the time interval embedding with the newly obtained *Query* and *Key* respectively, and add up their results. The remainder of the calculation process is the same as the standard self-attention. Although we find that using Wavenet instead of the feed-forward network has better performance on SASRec, it is found through experimentation that this method is not applicable to our model. Because of the introduction of the transformed time interval vector, the input embedding has more features compared with [21], hence we utilize five self-attention layers here to fully capture those features.

In the second step, we use the vertical convolutional filter in [34] to capture the recent preferences of the user in the last few interactions. In the last step, we concatenate the output of the self-attention module with the output of the CNN module and the user embedding as the final output. This output and the candidate item embedding are then used to calculate the top scoring recommended items. Extensive experiments have demonstrated the effectiveness of our proposed model. On the Amazon Game dataset, our model achieved 3.55% improvement in NDCG@10 and also obtain satisfactory results on other datasets as well.

By combining the advantages of both methods, our approach is also proved to have greater applicability.

6.2 Future Work

Although our proposed model achieves satisfying performance, it still has a number of points that could be improved. From the perspective of the model, because it incorporates temporal features, preprocessing and calculation of temporal data also increases the time complexity of the model, resulting in a significant increase in computational cost. Therefore, in future study, we can explore some new methods to improve the model's effect without sacrificing too much computational cost. Also, it is worth researching how the Wavenet can be improved to make it more suitable for our model. Another drawback of our model is that we only implement the incorporation for short-term preferences based on different item embeddings. Hence, in future research we can work on utilizing the same shared embedding technique as in [18].

From a data perspective, we will test our model against as many different domain datasets as possible to evaluate its robustness. And since our current model considers only one transformation of the data, there are more aspects in the temporal information that we may explore further to aid the model in improving its efficiency. Also, we should define 'recent interactions' in a more rational way, for example by separating user sequences by days to get the last one or few days as 'recent interactions'. In addition, because most of the items in the current dataset have category attributes or can be classified, and user-category interactions are always denser than user-item interactions, hence, we can take full advantage of this finding, and more helpful additional features may be inferred from them.

Bibliography

- [1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. 31
- [2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016. 15
- [3] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 335–344, New York, NY, USA, 2017. Association for Computing Machinery. 8
- [4] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed Chi. Top-k off-policy correction for a reinforce recommender system, 2020. 11
- [5] Shuo Chen, Josh L. Moore, Douglas Turnbull, and Thorsten Joachims. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, page 714–722, New York, NY, USA, 2012. Association for Computing Machinery. 7
- [6] Xiacong Chen, Lina Yao, Julian McAuley, Guanglin Zhou, and Xianzhi Wang. A survey of deep reinforcement learning in recommender systems: A systematic review and future directions, 2021. 10
- [7] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhya, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide amp; deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, page 7–10, New York, NY, USA, 2016. Association for Computing Machinery. 7, 9
- [8] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading, 2016. 17
- [9] Robin Devooght and Hugues Bersini. Collaborative filtering with recurrent neural networks, 2017. 8
- [10] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4), December 2016. 1

- [11] Yuyun Gong and Qi Zhang. Hashtag recommendation using attention-based convolutional neural network. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI'16*, page 2782–2788. AAAI Press, 2016. 11
- [12] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for ctr prediction, 2017. 7
- [13] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. Outer product-based neural collaborative filtering, 2018. 8
- [14] 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, WWW '17*, page 173–182, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee. 8
- [15] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems, RecSys '16*, page 241–248, New York, NY, USA, 2016. Association for Computing Machinery. 10
- [16] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks, 2016. 9, 12
- [17] Santosh Kabbur, Xia Ning, and George Karypis. Fism: Factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13*, page 659–667, New York, NY, USA, 2013. Association for Computing Machinery. 6
- [18] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206, 2018. iii, 14, 29, 31, 36, 39, 44
- [19] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. 1, 6
- [20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 21, 22
- [21] Jiacheng Li, Yujie Wang, and Julian McAuley. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining, WSDM '20*, page 322–330, New York, NY, USA, 2020. Association for Computing Machinery. iii, iii, 2, 14, 29, 36, 43
- [22] Jiwei Li, Will Monroe, and Dan Jurafsky. Understanding neural networks through representation erasure, 2017. 2
- [23] Yang Li, Ting Liu, Jing Jiang, and Liang Zhang. Hashtag recommendation with topical attention-based LSTM. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3019–3029, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. 11

- [24] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. Xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '18, page 1754–1763, New York, NY, USA, 2018. Association for Computing Machinery. 8
- [25] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003. 6
- [26] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. Stamp: Short-term attention/memory priority model for session-based recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '18, page 1831–1839, New York, NY, USA, 2018. Association for Computing Machinery. 13
- [27] Youngki Park, Sungchan Park, Woosung Jung, and Sang-goo Lee. Reversed cf: A fast collaborative filtering algorithm using a k-nearest neighbor graph. *Expert Systems with Applications*, 42, 05 2015. 6
- [28] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. *Proceedings of the Eleventh ACM Conference on Recommender Systems*, Aug 2017. 12
- [29] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000, 2010. 7
- [30] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 811–820, New York, NY, USA, 2010. Association for Computing Machinery. 2, 7, 12
- [31] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. *Collaborative Filtering Recommender Systems*, pages 291–324. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. 1
- [32] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, page 111–112, New York, NY, USA, 2015. Association for Computing Machinery. 8
- [33] Guy Shani, Ronen I. Brafman, and David Heckerman. An mdp-based recommender system, 2015. 7
- [34] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding, 2018. 12, 24, 32, 33, 36, 43
- [35] Zhulin Tao, Xiang Wang, Xiangnan He, Xianglin Huang, and Tat-Seng Chua. Hoafm: A high-order attentive factorization machine for ctr prediction. *Information Processing Management*, 57(6):102076, 2020. 8

- [36] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. 24
- [37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. 13, 15, 18, 19, 20, 29, 31
- [38] Shoujin Wang, Liang Hu, Yan Wang, Longbing Cao, Quan Z. Sheng, and Mehmet Orgun. Sequential recommender systems: Challenges, progress and prospects. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6332–6338. International Joint Conferences on Artificial Intelligence Organization, 7 2019. 2, 12
- [39] Suhang Wang, Yilin Wang, Jiliang Tang, Kai Shu, Suhas Ranganath, and Huan Liu. What your images reveal: Exploiting visual contents for point-of-interest recommendation. In *Proceedings of the 26th International Conference on World Wide Web, WWW '17*, page 391–400, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee. 10
- [40] Xin Xin, Bo Chen, Xiangnan He, Dong Wang, Yue Ding, and Joemon Jose. Cfm: Convolutional factorization machines for context-aware recommendation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 3926–3932. International Joint Conferences on Artificial Intelligence Organization, 7 2019. 8
- [41] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Jiajie Xu, Victor S.Sheng S.Sheng, Zhiming Cui, Xiaofang Zhou, and Hui Xiong. Recurrent convolutional neural network for sequential recommendation. In *The World Wide Web Conference, WWW '19*, page 3398–3404, New York, NY, USA, 2019. Association for Computing Machinery. 12
- [42] An Yan, Shuo Cheng, Wang-Cheng Kang, Mengting Wan, and Julian McAuley. Cosrec: 2d convolutional neural networks for sequential recommendation, 2019. 12
- [43] Haochao Ying, Fuzhen Zhuang, Fuzheng Zhang, Yanchi Liu, Guandong Xu, Xing Xie, Hui Xiong, and Jian Wu. Sequential recommender system based on hierarchical attention networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 3926–3932. International Joint Conferences on Artificial Intelligence Organization, 7 2018. 13
- [44] Jing Zhang, Bowen Hao, Bo Chen, Cuiping Li, Hong Chen, and Jimeng Sun. Hierarchical reinforcement learning for course recommendation in moocs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):435–442, Jul. 2019. 13
- [45] Shuai Zhang, Yi Tay, Lina Yao, and Aixin Sun. Next item recommendation with self-attention, 2018. 13
- [46] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system. *ACM Computing Surveys*, 52(1):1–38, Feb 2019. 2, 10

- [47] 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*, WSDM '17, page 425–434, New York, NY, USA, 2017. Association for Computing Machinery. 10