NTNU
Norwegian University of
Science and Technology

Inter-Session Temporal Modeling in Session-Based Recommendation using Hierarchical Recurrent Neural Networks

Bjørnar Vassøy

Master of Science in Computer Science
Submission date: June 2018
Supervisor: Massimiliano Ruocco, IDI
Co-supervisor: Eliezer de Souza da Silva, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Recommendation has been a highly relevant and lucrative field of expertise for quite some time. Since the end of the last millennium, session-based recommendation has emerged as an increasingly applicable branch of recommendation. One of the main contributions to this is the ever increasing level of competition in e-commerce and web-services. The increased competition both leads to smaller user-bases and more sparse user histories, because user information is being spread between competing service providers. Furthermore, streaming services and e-commerce services tend to operate on very large and sparse amounts of recommendable entities. Thus, modern recommendation environments are often very sparse, both with regards to consumables and consumers. Over the last few years, the use of Recurrent Neural Networks (RNNs) has shown great promise in the field of session-based recommendation. Different additions have been proposed for extending such models in order to handle specific problems or data properties. Two of such extensions, are modeling of inter-session relations and modeling temporal aspects of the recommendation. The former can allow the session-based recommendation to utilize extended session history and inter-session information when providing recommendations for the current session. The latter has been used to both provide state-of-the-art return-time prediction and also for improving the recommendation itself. In this thesis we propose a model that combines these two extensions through joint modeling of inter-session relations and inter-session temporal relations. In an effort to improve the quality of the inter-session modeling, some effort was also put into improving the personalization of the model. The model is shown to improve the recommendation on two datasets over strong recommendation baselines. At the same time, it is shown to provide state-of-the-art predictions for when users will initialize their next sessions, and the performance is compared with strong time prediction baselines. We also introduce a novel tuning parameter for tuning the model's focus on short-/long-term time-modeling, which can be used for adjusting to different datasets and/or use-cases. Through experimenting with synthetic time distributions, that are loosely based on the users they generate data for, we show that the model is robust with regards to different time-models and is able to capture user behaviour. Finally, the model provides an initial recommendation for the first action in the next session, before said session is initialized, to accompany the time-prediction and providing the user an initial suggestion. This recommendation is shown to be comparable, or better, than the first (regular) recommendation after being provided the initial selection of the new session.

Sammendrag

Anbefaling har vært et høyest relevant og lønnsomt felt av ekspertise i lang tid nå. Siden slutten av sist millenium har nytten av økt-basert anbefaling økt drastisk. En av de største årsakene til dette er den kontinuerlige økningen av konkurranse innen e-handel og webtjenester. Større konkurranse fører både til mindre brukerbaser og mer sparsomme brukerhistorikker, ettersom brukerinformasjon blir spredt mellom konkurrerende tjenesteleverandør. Videre, strømningleverandører og e-handel tjenester bruker å operere på store og sparsomme mengder med anbefalelses enheter. Dette medførere at moderne anbefalingsmiljø er veldig sparsomme, både med tanke på brukerdata og forbruksvarer. I løpet av de siste få årene har bruken av Recurrent Neural Networks (RNN) gitt lovendes resultater innen feltet økt-basert anbefaling. Ulike tillegg har blitt foreslått for å utvide slike modeller til å håndtere spesifikke problemer eller dataegenskaper. To slike tillegg er modellering av inter-økt relasjoner og modellering av tidsaspekter ved anbefaling. Førstnevnte kan la økt-basert anbefaling utnytte utvidet økt-historikk og inter-økt modelleringsinformasjon for å tilby anbefaling for den pågående økten. Sistnevnte har blitt brukt til å både tilby state-of-the-art returtid prediksjoner og til å forbedre selve anbefalingen. I denne avhandlingen foreslår vi en model som kombinerer disse to utvidelsene gjennom felles modellering av inter-økt relasjoner og inter-økt tids modellering. I et forsøk på å forbedre kvaliteten av inter-økt modelleringen, ble det også brukt litt energi til å forbedre personaliseringen til modellen. Modellen er vist til å forbedre anbefalingen i to dataset over sterke grunnlinjer. På samme tid er den vist å tilby state-of-the-art prediksjoner for når brukere vil starte sine neste økter, og ytelsen blir sammenlignet med sterke tidspredisjon grunnlinjer. Vi introduserer også en original justeringsparameter for å justere modellens fokus på kort-/langtids tidsmodellering, som kan bli brukt for å justere til forskjellige dataset og/eller bruksmønster. Gjennom å eksperimentere med sytetiske tidsfordelinger, som er løst basert på brukerene de generer data for, viser vi at modellen er robust med tanke på ulike tidsfordelinger og er i stand til å fange brukermønstre. Til slutt, modellen tilbyr en initial-anbefaling for første handling i den neste økten, før denne økten er startet, for å ledsage tidsprediksjonen og tilby brukeren et initial-forslag. Denne anbefalingen er vist å være sammenlignbar, eller bedre, enn den første (vanlige) anbefalingen etter å ha blitt gitt initial-valget i den nye økten.

Preface

This thesis was written in the spring of 2018 for the Department of Computer Science(IDI) at the Norwegian Univeristy of Science and Technology(NTNU). The author wishes to express his deepest gratitude for the guidance and help provided by supervisor Massimiliano Ruocco and co-supervisor Eliezer de Souza da Silva. Additional thanks goes out to Erlend Aune for his contributions.

Bjørnar Vassøy
Trondheim, June 11, 2018

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The introduction will present the background and motivation for the proposed model, as well as formalizing the goals for the thesis.

1.1 Background and Motivation

Recommendation is a broad field with many different approaches and properties. It can therefore be beneficial to structure the types in to distinct categories. The distinctions can often be found in the recommending environment, particularly in which type and quantity of data is available. The user, group or entity, hereby referred to as simply user, to give recommendation can for instance be known(access to user history), not known at all or somewhere in-between. Another type of distinctions can concern quantitative properties of the data, this can for instance be the sparsity of the dataset or simply its overall size. There can also be requirements related to the final performance of the system, which can put restrict modeling choices. This can be illustrated by questions like: what is the "cost" of a prediction? Is it feasible to get real-time predictions with the system, or do we have to reduce the computational cost of predicting? It may be trivial to create a highly accurate model, but if the model needs to process terabytes of data for every single recommendation, it might not be applicable in for instance a real-time recommendation setting.

1.1.1 Background

A highly relevant type of recommendation systems, that have received astonishing little attention considering its relevance, is session-based recommendation. In modern times people, corporations and governments are becoming increasingly

aware of privacy concerns. Governments have seemingly increased their effort in keeping up with the digital revolution and new regulations and restrictions concerning information are becoming more prevalent and/or are revised for the ever changing state of the digital world. For instance, the GDPR(General Data Protection Regulation)[1], became enforceable throughout most of Europe as recent as 25.05.18, and enforces many new restrictions on personal data. There also seems to be a greater focus on the dangers of the spreading of private information as well as quite a few recent highly profiled cases where large web-communities have been hacked and sensitive data have been spread online. This, as well as the ever increasing number of competing e-commerce sites, are making session-based recommendation more relevant than ever. People may refrain from using services that requires log-in, services may not be allowed to store certain information about a user, or the average user of the service might only have logged on a few times because they use competing services as well. All this makes the ability to provide good recommendation given limited user history, e.g. one or more sessions, both very useful and lucrative.

Until quite recently, session-based recommendation systems were usually based on item-to-item methods which in essence are types of nearest neighbor methods. The general idea behind these are to look at the last item in the session and recommending a new similar item. Another approach is to look at the current session so-far, and recommend items based on a similar old session. The process of evaluating similarity does not scale well with item complexity, or the size of the database, in case a session-to-session approach is used. This becomes even worse if one tries to improve the recommendation by looking at more than one previous items. Good item similarity measures often relies heavily on descriptive features, preferably a great number of such features, and methods for comparing these. In general, the quality of the similarity measure heavily affects the retrieval time.

While there have been efforts into making nearest neighbor methods scale better with size and complexity of data, like approximated retrieval [Muja and Lowe, 2014] and other data structures for fast retrieval of similar entries [Liu et al., 2006], the worst case time complexity is still quite intimidating. There is also usually a trade-off between accuracy and time complexity when applying approximate retrieval methods, which has to be carefully considered. Another issue with nearest neighbor solutions are that they struggle to capture finer temporal dynamics. For the most basic nearest neighbor approaches, the order of the items in the history is totally irrelevant for the recommendation. If a user first watched movie A, then B, and was recommended C, then a user who first watched movie B, then A would also be recommended movie C by such solutions. This might

[1]Link to official GDPR webportal: https://gdpr-info.eu/

make sense in many cases, but is generally a naive assumption. A simple extension to the simpler nearest neighbor solution, that could consider some temporal aspects, is to add a simple time based discount. This way the most recent item is the most relevant and the ones before this are discounted based on their position in the history. One could also add another discount factor based on actual time since item was consumed as opposed to its position in the history alone. If the last movie a user watched, was watched many months before she finally logged onto the service again, it is not necessary true that she would want to continue where she left off. One can observe that in an attempt to be better at using temporal dynamics in the recommendation, the model has become increasingly more complex. Not to mention that the simple suggestions provided are all based on discounting, which can only remove information and not explicitly "enhance it". They are also making serious assumptions about nature of the temporal dynamics, e.g. the relation between time and relevance.

After taking a closer look at some problems with nearest neighbor models it should be apparent that a model that can abstract the "similarity measures" of large quantities of discrete items, as well as inherently capturing finer temporal dynamics of a recommendation/consumption sequence, could preform well in session-based recommendation. At least in the sense of addressing many of the shortcomings of the nearest neighbor solutions. A type of model that is not only capable of this, but excels at it, are recurrent neural networks (from here-on abbreviated to RNN). In the last couple of years RNNs have been used for session-based recommendation to great success, often outperforming nearest neighbor methods by large margins.

1.1.2 Motivation

One of the problems with fully generalized session-based recommendation where each session is considered completely independent from all other sessions, is that it is very difficult to give accurate recommendations during the first steps of a new session. It is hard for the system to identify a "user preference" given little information. In order to remedy this, a few recent models [Ruocco et al., 2017; Quadrana et al., 2017] have tried to model inter-session relations, while continuing to model the intra-session relations. The intuition behind this is to gather useful information from a limited history of previous sessions and using this to help the initial recommendations in new sessions. This type of modeling requires that the users are not anonymous and that, at the very least, some information of their activity can be stored. Inter-session modeling can also serve as an implicit form for personalization in the sense that the history will most likely contain some of the user's characteristics. Models that utilize this could be very relevant in ser-

vices where the users have to log in, but where the service for some reason cannot store extended information about their users. Examples of such reasons can be legal restrictions, capability concerns or a need to optimize the recommendation. Another reason for limiting the user history is that even though more information most likely will give better recommendation, it is not given that it is practical to go through all relevant user history for each recommendation. In such cases, a limited history with the most relevant user information, might be the best possible compromise. Note that inter-session modeling can be used in models that do not consider any intra-session modeling, for instance next-basket recommendation where the session is considered a basket. Such models could work well in for instance an online retail service, where there are explicit sessions with multiple items, but where the order of the items often is approximately random. Models that both models inter-session and intra-session relations can be a better fit for domains where the intra-session order can be of importance. Examples of this can be music services, online discussion boards and video/media/art sharing services.

Another aspect of session-based recommendation that has been given some attention recently, is to improve the temporal awareness of the recommendation or to model temporal aspects explicitly. It is easy to see that time can affect recommendation to a large degree. Food eaten for breakfast is usually quite different from the food served for dinner. Music listened to when commuting to work can be very different from the music played at parties attended during weekends. Christmas themed movies tend to be more popular during Christmas. Thus, the ability to capture such dynamics in a recommendation system can be very valuable in many recommendation domains. Additionally, the ability to predict *when* to recommend something could be very useful. Commercially, through well timed advertisement/increased satisfaction with service or better estimation of how much to stock up of different products. But also for more universally beneficial purposes, like a health diagnosis system identifying potentials for future health concerns, given sequences of symptoms. There is also the possibility that consideration of user consumption history might also be helpful in time modeling. A user who is starting to lose interest in a service might start exhibiting a certain behaviour in the period leading up to less activity levels, and a user who is getting more interested in the service might exhibit another type of behaviour. Thus, recommendation and time-modeling appears to be mutually beneficial in many domains, for which joint models could have a lot of potential. Although there have been some work put into this in the past, the application in session-based recommendation, as opposed to recommendation of single actions, is still relatively unexplored.

1.2 Goal and Research Questions

Goal Jointly provide state-of-the-art session-based recommendation and inter-session time-modeling

The motivation for jointly performing multiple tasks can be that the tasks benefit each-other, or that the removed need for multiple models is worth potential deterioration of performance. By attempting to jointly provide state-of-the-art session-based recommendation and inter-session time-modeling, we would like to see if there is any grounds for performing these two tasks jointly. Is it possible to achieve state-of-the-art performances on both of these tasks simultaneously? Since the return-time prediction can/should be performed before the next session, there is also grounds for providing an initial recommendation along with this prediction. This way, the users will have a suggestion ready when they do return.

Research question 1 Provide state-of-the-art return-time predictions for future sessions

Research question 2 Provide state-of-the-art recommendations given the selections, so-far, in current session

Research question 3 Provide initial recommendation for first selection in future session

1.3 Contributions

The main contributions of this master's thesis are:

- We apply the marked point process proposed in Du et al. [2016] in the new domain of hierarchical session-based recommendation, and model inter-session time-gaps through high-level inter-session modeling and time modeling.

- We enhance inter-session modeling, by extending session-representations with contextual information, in order to support a complex personalized hierarchical-RNN architecture in jointly training for three different tasks.

- Our proposed architecture is capable of both providing estimations for when a user will initialize a new session and providing an initial recommendation, before said session is initialized. After initialization, it provides real-time recommendations through combining inter- and intra-session modeling.

- We introduce a novel tuning hyperparameter in order to allow adaptation to different time-gap distributions and tuning the short- vs long-term consideration of the model.
- We show that the time modeling is robust and capable of learning many different time-distributions through testing on both real-world inter-session time-gaps and synthetically generated ones.
- We evaluate the proposed architecture on two real-world datasets, where it is shown to significantly outperform strong baselines in time-prediction of the most frequent time-gaps, while simultaneously improving recommendation over strong recommendation baselines.

A full paper based on the thesis has been submitted to the Conference on Information and Knowledge Management 2018(CIKM)

Chapter 2

Background Theory

This chapter is a brief introduction to the background theory. The purpose is not to make the reader experts within the different fields, but to provide a basic foundation that hopefully will allow even readers with no background in machine learning or ANNs to grasp the overall intention and function of the proposed model. The main topics discussed are ANNs, RNNs (in particular its modern variants LSTM and GRU) and marked point processes.

2.1 Artificial Neural Networks

Artificial neural networks were first introduced in its first iteration in McCulloch and Pitts [1943] and are essentially networks of linear algebra. Like the name entails, ANNs are inspired by actual biologic neural networks found in nature, but aside from a few borrowed concepts, that have been further simplified and adapted for hardware and computational ease, these are two very different things in terms of functionality. The history of ANN is long and rugged and contains many smaller and greater innovation resulting in modern ANN-based models that are capable of out-performing humans on extremely abstract tasks like image recognition [He et al., 2015] and exceedingly complex task like playing the board game Go [Silver et al., 2016].

2.1.1 Key concepts

ANNs are essentially networks of vertices and edges through which numbers can flow, be transformed/altered, and finally outputted. Using analogy from bilogical neural networks, the vertices(often referred to as neurons in ANN terminology) are inspired by neurons and the edges between these are inspired by synapses.

"Communication" between neurons are conducted through the synapses(edges) and the neurons themselves can "fire" signals through their output synapses based on the signals they have received. The edges in ANNs are weighted, which means that the numbers passing through an edge is scaled by the edge's weight. In the vertices of ANNs, the different contributions from incoming edges are aggregated, then passed through an activation function before being propagated through the out-going edges. The activation functions are usually classified as either linear or non-linear. Linear activation functions apply a linear function to the aggravated input. One of the more common linear activation functions is to simply pass the aggregated input through unaltered. Linear activation functions are good for simpler problems where linear relations are expected, or can be a good approximation. Examples of such linear relations could be the conversion between Celsius and Fahrenheit, or electric resistance of material for small variations in temperature. An example of an ill-suited relation for a linear activation function, is human growth per year. While the growth may be stable for intervals of many years, there are usually periods of increased and decreased growth, not to mention that it usually comes to a full stop at some point. More complex activation functions are non-linear, where typically a sigmoid function, hyperbolic tangent function or a rectified linear function is applied to the aggravated input. The use of non-linear activation functions can allow the modeling of more complex functions, because they do not impose a linear relation between the input and the output. A common example that illustrates the limitations of linear activation functions, is that a network which exclusively uses linear activation functions is not be able to model the simple XOR problem no matter the number of neurons and computational power. XOR(eXclusive OR) is the simple function of returning 1 if inputted a single 1, otherwise returning 0. By adding simple non-linearities, like thresholds that outputs zero unless the input is greater than the threshold, the XOR-problem can easily be solved by a 3-neuron network.

The simpler ANNs are ordered into different layers of neurons, where each neuron is connected to each neuron or input/output of the neighbouring layers. The propagation through the layers is usually strictly synchronized, meaning that all the outputs of one layer are calculated, usually in parallel, then sent to the next layer as input. The next layer then calculate all of its outputs before propagating to the next etc. These structures are called feed-forward layers and still appear in most ANN model, albeit increasingly rarely throughout the entire model. In essence, ANNs solve problems by modeling a high dimensional (non-)linear function in order to approximate the target given the input. For instance, if a network is used to determine if the inputted images contain a cat or a dog: the network will try to find a function that outputs multi-dimensional points, when being provided cat images, that can be well separated from those outputted when

being provided dog images. These points are arrays of values where each entry corresponds to one dimension. The separation between the two classes will be a multi-dimensional plane through the point space. It might be easier to visualize by only considering two dimensions, where the problem would be to output points in the 2D-plane which can be separated well by a line, according to class. Thus, if an image result in a point on the cat-side of the line/plane, it is classified to be a cat image and vice versa.

2.1.2 Training

The output of a simple ANN is fully determined by the input, the weights and the network "structure", which includes topology(layers and number of neurons within each layer) and activation functions. The structure is typically kept static while the weights are changed. For reasonable ANN structures, multiple different functions over the input can be achieved by only changing the weights. Identifying how to change the weights in order to learn a new function is not so straightforward, especially for networks with many layers. The most common solution is a loss function, and then perform gradient descent through gradients found using a backpropagation algorithm. The loss function is devised to define the correct behaviour and outputs an error based on expected output of network and actual output. Backpropagation identifies how each weight should be adjusted in ANNs, i.e. should it be greater or smaller to decrease the error. Gradient descent is essentially done by finding how a small change in each weight affects the loss and then subtracting from each weight based on this differentiation. Thus, adjusting the weights in the "direction" found to decrease the loss. In ANNs, weights affect the input to the next layer, whose weights affect the input to the layer after and so forth. This means that differentiation with respect to a weight is a product that involves the partial derivatives of weights and activations that appear later in the network. This is also the reason why *back*-propagation goes *backwards* through the graph, as the gradients of one weight is independent of what comes before but dependent on what comes after. The final update of the weights can be done in a few different ways: after each training instance (stochastic training), after the full trainingset (batch training), or after a subset of the trainingset (mini-batch training). Batch and mini-batch training tends to be more stable because the adjustments are based on more "opinions", which has a normalizing effect and can keep the weights from fluctuating heavily between updates. These training schemes can also reduce the number of times the expensive backpropagation has to be performed. For reasons like these, batch and mini-batch training are more common than stochastic training.

Figure 2.1 is a simple illustration that visualizes several of the ANN concepts

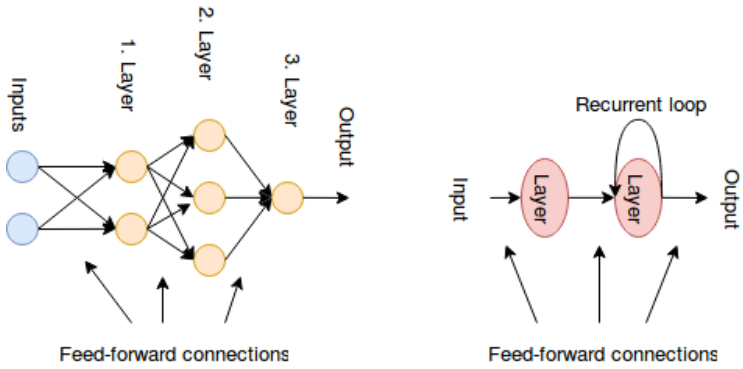

Figure 2.1: Simple illustration showing key concept and structure of ANNs as well as a type of recurrent loop that could be used in RNNs

discussed in this section, and also the principals of recurrent edges in RNNs, which will be discussed in the next section.

2.2 Recurrent Neural Network

Recurrent neural networks are formally "ANNs with one or more feedback connections". A feedback connection is a directed cycle in the network structure. In practice, this usually done by storing the output of one part of the network during the processing of some input, and feeding this back to an earlier part of the same network when future input is processed. This is often thought of as an inner state or memory in the network, because the processing of new input is dependent on previous input. Thus, RNNs are typically applied on sequence data where there is some sort of dependencies between the units in the sequence, such that the memory of earlier units can be useful when processing new units. This can for instance be words in a sentence or letters in a word. RNNs have been used in many state-of-the-art natural language processing models because of its ability to capture dependencies in sequence data.

2.2.1 Elman

The Elman RNN was proposed in Elman [1990] and is one of the more common of the simple earlier RNN architectures. When the Elman RNN is given an input, the input is first passed through a set of weights and then added to the output of the last state passed through another set of weights. An activation

function is applied to the resulting vector, resulting in the new state which is both propagated through a feed-forward layer with an activation function, to get the output of the RNN, and to the next timestep as the new state of the RNN.

$$\begin{aligned} h_t &= \sigma_h(W_h X_t + U_h h_{t-1} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y) \end{aligned} \tag{2.1}$$

Equations 2.1 shows the update done by the Elman RNN. x is the input, y is the output, h is the hidden state, σ are activation functions, and W, U and b are weight parameters/bias. The subcripts $_h$ and $_y$ is used on weights, biases and activation functions in the respective updates for the hidden state and output, while subscripts involving t are used on arguments to indicate the timestep they belong to. Similar notation will be used further on.

2.2.2 Training and vanishing gradients

When looking at equation 2.1 one can observe that the output y_t is not only dependent on x_t but also on h_{t-1}. h_{t-1} will in turn be dependent on x_{t-1} and h_{t-2}, and this trend will continue until the first input and the initial hidden state is reached. The weights W_h and U_h as well as the bias b_h will be applied for each of these h and x. Regular backpropagation is not capable of handling such models, so a specialized class that is capable of this, called backpropagation through time (BPPTT), has been devised. The key strategy introduced by these is to unroll the timesteps in time and adding the different gradients of the same weights, but from different timesteps, together for the combined gradient. Apart from this, they are not that different from the regular backpropagation algorithms.

A problem with the simpler RNNs are that they are heavily affected by the vanishing gradient problem. The vanishing gradient problem stems from the aforementioned observation that the gradients of a weight early on in in a many layered network is the product of many partial derivatives of the following weights and activation functions. When some of the involved terms are very small, we end up getting small numbers multiplied with small numbers, resulting in even smaller number. If this is the case, the intensity of the gradients of the earlier weights in a multi-layered network can end up being very close to zero, and it becomes very difficult to both train and utilize such weights. This problem is further increased if many activation functions are the classic sigmoids or hyperbolic tangents, because their activations tend to get saturated in the upper and lower range, which result in very small gradient. Exploding gradients, albeit not so common, can also be a problem if many of the partial derivatives conversely are very large, and can result in very unstable learning or total divergence of weights. RNNs are inherently deep and many-layered because of the unrolling

of time. Moreover, each unrolling is often accompanied by one or more activation functions. This means that it can be very difficult to train earlier weights given an early timestep but according to how this influenced the error of a later timestep.

2.2.3 LSTM

Long Short-Term Memory (LSTM) were proposed in Hochreiter and Schmidhuber [1997] with the intention to deal with vanishing and exploding gradients, that often makes training simpler RNNs very difficult. The overall structure of LSTM's is pretty versatile and there are a lot of different variations. The key components are the cell/unit itself, which contains the cell state, and three different "gates". The gates are: the input gate(controls what to be accepted from the input), the forget gate(controls what information in the state/memory to be forgotten), and the output gate(controls what to be outputted by the cell). The gates allow the LSTM to discard information deemed irrelevant and only add information deemed important across different timesteps. Since the cell state is never in its entirety passed through an activation function, the cell can propagate memory that is less affected by the vanishing gradient problem than the more simpler RNNs. This can make LSTM-based models capable of learning very long dependencies while still being able to also capture shorter dependencies.

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
g_t &= \sigma_c(W_g x_t + U_g h_{t-1} + b_g) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ g_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
\tag{2.2}
$$

The update rules of one type of LSTM can be found in the equations 2.2. f_t is the forget gate, i_t is the input gate, g_t is used with the input gate to update the cell state, o_t is the output gate, c_t is the new cell state and h_t is both the output and the new hidden state. $\circ$ is the Hadamard product and the different activation functions are denoted by σ. Note that here the three main gates share the same type of activation function, which is usually the sigmoid activation, but the activation functions used in g_t and h_t have different subscripts. Both of these are usually hyperbolic tangent functions, but it is not unusual that they are different from each-other. Note how the new cell state is the sum of a hadamard product involving the old cell state and the forget gate, and a hadamard product involving the input gate and g_t, thus no activation function is directly applied to the previous cell state.

2.2.4 GRU

Gated Recurrent Units (GRU) Cho et al. [2014] is a quite recent addition to the family of RNNs that have already been used to great success in many state-of-the-art models. It is heavily inspired by the LSTM, and the largest differences between the two is that the GRU switches out the gates with a "reset" gate and an "update" gate. Furthermore, the GRU only operates with one hidden state, whereas the LSTM both has a hidden state and a cell state. A bit simplified, the update gate can be said to replace the input and the forget gate, while the reset gate is similar to the output gate. One added benefit of using GRU over for instance the LSTM described by equations 2.2, is that they are often significantly less computational demanding and can reduce the training and computational time considerably.

$$\begin{aligned} r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\ z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\ n_t &= \sigma_h(W_n x_t + r_t \circ (U_n h_{t-1} + b_{nh}) + b_n) \\ h_t &= (1 - z_t) \circ n_t + z_t \circ h_{t-1} \end{aligned} \tag{2.3}$$

The update rules of a common GRU can be found in the equations 2.3, where r_t is the reset gate, z_t is the update gate and n_t is used to update the hidden state. The activation functions of the named gates are typically sigmoids while the σ_h activation function is typically a hyperbolic tangent function.

2.3 Point process

Point processes are part of probability theory and describe distributions of points in a mathematical space. For instance, the probability of sampling a specific point in a *uniform* point process should by definition be the same as the probability of sampling any other point in the domain.

2.3.1 Poisson

In the Poisson point process, each random event is completely independent of each other. Thus, the presence of one point does not increase or decrease the likelihood of any other point what so-ever. The inter-event time between Poisson point processes is described by the exponential distribution.

$$f(t) = \lambda e^{-\lambda t}, t \geq 0 \tag{2.4}$$

The exponential distribution can be seen in equation 2.4, where λ is the intensity or rate. The intensity describes the rate at which events are expected to occur,

and is fixed for the exponential distribution. For many other point processes, the intensity is conditional on for instance time.

2.3.2 Hawkes

The Hawkes process, sometimes referred to as self-exciting process, is a class of point processes that does not consider events to be independent. The intensity of a Hawkes process at time t is conditional on each event that has happened between t_0 and t. Hawkes processes are often used to model relations where the occurrence of one point increases the probability of a new occurrences.

$$\lambda(t) = \lambda_0 + \sum_{j:t_j<t} \gamma(t, t_j) \tag{2.5}$$

Equation 2.5 is a general Hawkes process's conditional intensity function. $\gamma(t, t_j)$ is the contribution of the event that happened at time t_j and is called a kernel. A typical parameterization of a Hawkes kernel is $exp(-\beta(t-t_j))$, which discribes a situation where the occurence of event increase the probabiliy of future events. The Hawkes process is more expressive than the Poisson process by being able to also model dependent inter-event times, which is arguably more realistic in most domains, instead of just fully independent ones.

Figure 2.2 is a simple illustration depicting how the intensity value of a Hawkes process can be affected by close occurring events and consequently how events increase the probability of future events in the immediate future.

2.3.3 Marked point processes

Marked point processes are point processes where each point has a belonging marker, in other words, it has both a position and a marker/class. Furthermore, the probability distribution of future marked points may be dependent on both the time since- and markers from previous points.

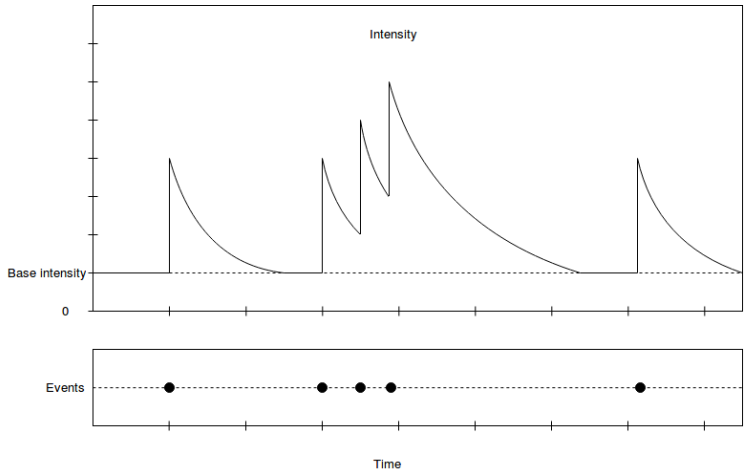

Figure 2.2: Simple illustration showing how the intesity of an arbitrary Hawkes process can look like during a series of events.

Chapter 3

State of the art

There are two main topics of interest when considering the relevant state-of-the-art for the model proposed in this thesis, which are session-based recommendation and temporal modeling in recommendation. This chapter will present and describe selected state of the art from these topics, focusing on the most relevant research. A lot of focus has also been given to the "original" RNN-based recommendation model Hidasi et al. [2015] because it is usually used as the basis of more recent and specialized RNN-based models.

3.1 Session based recommendation

Like discussed in the Introduction 1.1.1, session-based recommendation have up until recently mainly been handled by item-to-item/ neighborhood-based methods, Sarwar et al. [2001]. While such methods can perform very well under the right circumstances, they typically struggle in a general session-based recommendation context. Many of the problems keeping such methods from achieving good results in session-based recommendation, can be attended by considering sessions as sequences of items and using a RNN-based model.

Hidasi et al. [2015] is widely credited to be the first to apply a RNN-based recommendation model and achieving state-of-the-art results. The model they propose is designed to make the most out of very limited information. They assume that the users of the model are fully anonymous in the sense that the model does not have any user-history beyond the live sessions. Furthermore, it assumes every session to be independent. Thus, the only information the model can utilize in order to customize the recommendation for a user during a live session, is the items said user has selected so far in the current session. The model provides rec-

ommendations for future selections, which is based on all preceding selected items within the session. The structure they found to give the best results is quite simple: a single layer of GRU units followed by a single feed-forward-layer. The GRU units, having recurrent connections, provide the state and captures the temporal dynamics of the sessions, while the feed-forward layer outputs non-normalized scores for each item. Figure 3.1 is a simple depiction of the proposed architecture.

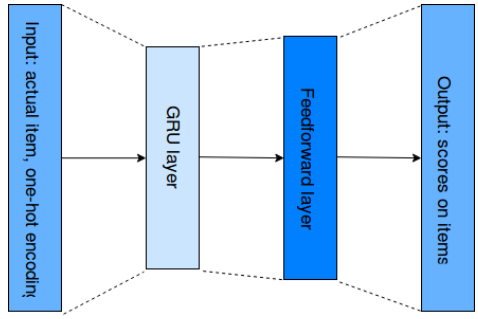

Figure 3.1: Figure of the best performing architecture (adapted from Hidasi et al. [2015])

Furthermore, they propose three different additions to the training procedure. First of which is a "session-parallel mini-batching"-scheme, where sessions of different lengths are handled by interleaving rather than padding each session to a fixed max length. Sessions are batched together, but since they are of different lengths, sessions are continuously replaced as they are processed. The benefit of this is not needing to process a lot of "paddings", nor having to find and extract relevant output, stemming from non-padded entries, from the output. Subsequently, there is no need for splitting overly long sessions into smaller ones in order to reduce padding. There is also a potential benefit of having equally many targets affecting the loss for each mini-batch, as opposed to the number of non-padded ones in the case of ignored paddings. The last point is less detrimental for large batches where the statistical difference between number of contributing sessions will be pretty stable even for padded sessions. Conversely, interleaved sessions makes managing the state and performing weight updates more challenging because one need to consider the start and end of each session. The second addition is a sampling of the output in order to avoid the need for evaluating the score of every item for every single prediction. Instead they sample this by only computing the score of the target item as well as a few negative examples, which are set to be the other targets in the mini-batch. This selection of negative

examples both removes the need for explicit sampling, while still ensuring that items are adjusted according to popularity. The last point is achieved because popular items will statistically appear more frequently in these targets. This is useful because the user is more likely to be familiar with the popular items, so if the user still prefers a less popular item, it might indicate a dislike of said popular items. The last addition is testing of different loss functions. In particular they test two different pairwise-ranking losses. Pairwise means that the scores are judged in pairs and not independently. One would in general want a positive item to have a greater score than a negative one. In the context of the sampling proposed, the positive item will be the target item while the negative items will be the other targets in the mini-batch. First of the two is the BPR ranking loss, which proposed in Rendle et al. [2012]. The second loss is called TOP1 and was devised by the authors themselves. The losses are pretty similar, but differ in their behavior for very small and very large differences between the target and the evaluated sample, caused by the log function used in BPR. The article also compare these with using cross-entropy loss, which is a common pointwise loss function. For pointwise loss functions, the final loss is only dependent on the score of the positive examples, the scores of negative samples does not affect it at all.

The model proposed was tested on two different datasets and achieved a 20-30% gain in the evaluated measures over the best performing baseline, which is stated to be an item-to-item method which recommends based on returned historical session retrieved through a cosine-similarity measure and the current session so-far.

In Ruocco et al. [2017] the intra-session model from Hidasi et al. [2015] is altered, and extended with a second level of RNNs in an attempt to capture inter-session dynamics. So the proposed model is a hierarchical RNN where one level considers inter-session dynamics and the other considers intra-session dynamics. As opposed to assuming fully anonymous users and totally independent sessions, this extension allows some simple and low-cost user history to be considered in order to provide better recommendation for live sessions. The user history is in the form of abstract representations of previous sessions. The inter-session level RNN is fed a finite number of the most recent session-representations in chronological order, and the output is used as the initial hidden state of the intra-session level RNN. The motivation behind this is to handle the cold-start problem. The cold-start problem is very prevalent in highly generalized RNN models, and stems from RNN's reliance on state/memory. The RNNs captures temporal dynamics within sequences, the longer the sequences the more data is available to infer

from and the longer dependencies can be identified. Thus, for small sequences, e.g. the first few items of a session, simple RNNs tends to perform worse. A simple intuition is that when being fed a single item, the model will struggle to identify which type of behavior this is, because there likely are many that fits the bill. When the sequence is longer on the other hand, it can exclude many options. So the purpose of the inter-sessions layer is to provide some form for information that will help the model in the first troublesome steps. By using RNN units in the inter-session layer, Ruocco et al. [2017] attempts to capture both finer temporal dependencies in the most recent sessions as well as functioning as some form for personalization. Two different session representation methods are proposed. First, and arguably simplest, is an average pool of all the item embeddings within a session. It is pointed out that such a solution would fail to capture any ordering of the items since any permutation of the same items will result in the same representation. The second method exploits the nature of RNNs, where the last output of the intra-session level RNN is used as the session representation. In contrast to Hidasi et al. [2015], Ruocco et al. [2017] achieved the best results when adding an embedding layer for the items. Additionally Ruocco et al. [2017] did not use the interleaved session mini-batching scheme, instead opting for padded sessions. Nor did they sample the output or use any pairwise losses. GRU were used in both RNN layers. The results shows a gain in the evaluated measuers of 7-12% on a dataset containing activity from the music service LastFm over the best baseline. The best performing baseline was an implementation of Hidasi et al. [2015] where the same omitting of interleaved sessions, sampling and pairwise loss had been done. For the LastFM dataset, the average pooling of item embeddings proved to be the better session representation. For the other dataset, which contains sub-forum activity on the popular Internet-community reddit, the model achieved a 24-32% gain in the evaluated measures in recommending sub-forums for the users. However for this dataset, the use of the last output of the intra-session RNN as session representations gave the best results. Figure 3.2 depicts the proposed model using last hidden state as session representation.

Quadrana et al. [2017] proposes an architecture that is very similar to the model in Ruocco et al. [2017]. They also propose using an inter-session RNN layer in order to handle the cold-start problem when one has access to a limited user history in the form of previous sessions. However, they mainly refers to this as personalization and not inter-session modeling. This article shares many of the same authors as Hidasi et al. [2015] and applies all the additions of that model in the new hierarchical model. The main difference between this hierarchical model and the one presented in Ruocco et al. [2017] is that this is only considering a

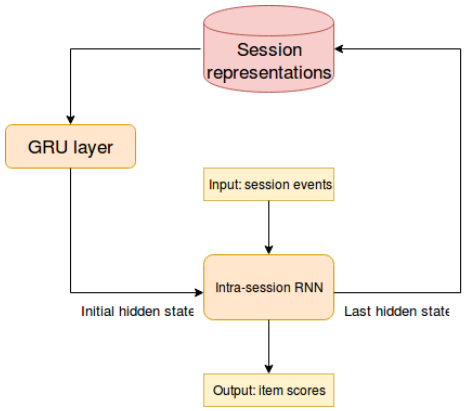

Figure 3.2: Figure showing the proposed hierarchical model (adapted from Ruocco et al. [2017])

session-representation scheme based on the last hidden state of the intra-session RNN. Additionally, these session-representations were created by passing the last hidden state to a final single layer feed-forward layer with a hyperbolic tangent activation function. They also experiment with propagating output of the inter-session network to all time-steps in the intra-session RNN, which complicates the model somewhat, but achieve slightly better results for one of the datasets evaluated. Another point of interest is the usage of an intra-session RNN baseline where multiple sessions are concatenated together and are considered to be a single long sequence. This performed quite a bit better than the non-concatenated intra-session baseline on a dataset with highly repetitive sessions, but worse than the best hierarchical setup for all datasets and all evaluation metrics. The hierarchical model proved to be the best performing overall by significant margin.

Other notable work using RNNs for recommendation, have focused on how to best handle context- and feature rich input like Liu et al. [2016] and Hidasi et al. [2016]. In Liu et al. [2016] a simple intra-session RNN model is modified by applying different sets of weights based on the context of the input. This context can for instance be time and date of different granularities, location, or weather at the time of an event. This is an attempt to capture finer contextual dependencies by using a dynamic structure, rather than enhancing the input with the context explicitly. In Hidasi et al. [2016] the focus is put on how to handle feature-rich input as opposed to input with a lot of context. An input can for instance be an

image with a caption which has both textual and image features. A lot of different RNN-based models are discussed: the simple baseline disregards all but one of the different features and a somewhat more advanced model concatenate the different features to a single input. The main contribution of the article are three different models using parallel RNNs for different features. These differ in the way outputs of the different parallel parts are combined for the final recommendation. In general the parallel models were found to outperform the alternatives.

Tan et al. [2016] proposes a lot of different techniques that can be applied in augmentation and pre-processing of data, designing of model and during training, to improve the accuracy and make faster prediction/recommendation in RNN-based models for session-based recommendation. The models proposed are based on a single layer of RNN units, more specifically GRU units, similar to Hidasi et al. [2015], and they use click-streams in e-commerce as data. The data is augmented by creating smaller sessions from the original sequences by extracting sub-sequences of all viable lengths that still includes the first event. This allows them to consider many more sequences than they started out with. Dropout is also applied in the embedding of the items, resulting in a probability that any item within any sequence can potentially be dropped during training. This is in order to make the model more robust for noisy input as well as for utilizing the, often observed, generalizing effect of dropout. The article also proposes to only train on sufficiently recent data by defining a temporal threshold where all data before this threshold are discarded. This is very relevant in data where clear trends can be observed, because things that were popular in years past might well be all but forgotten in the present time. Another proposed addition uses information from the future items in the session when recommending the next item, and is claimed to be specially relevant if there is little data available. This essentially trains a teacher model in parallel on the reversed sequences, which provides a secondary "predicted" target containing information about the future targets. The loss for the regular student model is then a weighted sum of the loss considering the real target and considering the predicted target. A final addition is proposed, mainly in order to make the recommendations faster, and is essentially to output an item embedding instead of the individual scores for each item. The latter is the usual practice, but scales pretty bad with the number of recommendable items. Different models applying different combinations of the mentioned additions were tested in order to find good synergies and consequences of using the different techniques.

Jannach and Ludewig [2017] presents a nearest neighbor method that actually

achieves better results than the RNN model proposed in Hidasi et al. [2015] on multiple datasets. This method is adapted specifically for session-based recommendation and the first step is to find similar sessions to the live session using an item cosine difference similarity measure. Then each item is given a score based on how many times they appear in the retrieved similar sessions. In order to optimize the score calculation, each item is given an index table containing the sessions they appear in. They also experimented with a hybrid/ensamble version where the recommendation was a weighted sum of the nearest neighbor recommendation and a RNN-based recommendation. The hybrid version ended up outperforming both single models by a significant margin and it was discussed that this indicates that the nearest neighbor's strict similarity focus was complimented/corrected by the RNN's ability to model sequence dependencies. The article further discusses the different benefits of the different models. Nearest neighbor methods have the benefit of having very small setup-time as well as being easy to extend with new data. The nearest neighbor method was also found to be more robust to reduction of the data size. On the other hand, the time of single recommendation was found to be twice that of the RNN model and this difference only increases for more data/larger action-spaces. However, the training time of the RNN model was observed to be significantly larger than the time needed to set up the nearest neighbor method and in order to add new data to RNN models, one usually has to re-train the entire model.

In Ludewig and Jannach [2018] the authors from Jannach and Ludewig [2017] compare multiple different models of multiple different types and introduces some specialized models for the domain of session-based recommendation. Among the represented models are association rule models, nearest-neighbor based models, matrix factorization models and the RNN-based model of Hidasi et al. [2015]. The models are evaluated on many different measures like the more standard reciprocal rank, precision and recall measures, but also so-called quality measures like coverage rate, hit rate and a "popularity" measure. The coverage rate and popularity measure will both be able to tell if the models have a unreasonable high bias towards recommending the most popular items, and conversely, whether they are able to recommend many different items. The models are evaluated on a total of 9 different datasets: four of which are from e-commerce, the next four are music related and in the last dataset they consider "read" events from a sports news portal. Among their more specialized methods are 4 different item-to-item methods which consider sessions as the "item" and the specific selections as "attributes" as they did in their other work in Jannach and Ludewig [2017]. The first of which is very standard, but the three others, referred to as "session aware", apply different scaling based on the selections' positions in the sessions. This way

they add some order dynamics to the method, which the more simpler item-to-item methods lack. They also design some association rule models with similar thought put into incorporating order dynamics. Their results show that for most of the datasets, Hidasi et al. [2015] is outperformed on many of the measures, usually by one of the session aware item-to-item methods or the extended association rule methods. This is also the case for the more standard measures and not just the additional quality measures introduced. They also look into memory usage and computational time required for each recommendation. As one should expect, the item-to-item models scale worse with data size, both in memory and computational time. However, they claim that the upper computational time is still well within what is realistic to allow in a real-time recommendation setting due to fast look-up operations, approximations and/or sampling. On the other hand, the same methods scale much better than the model from Hidasi et al. [2015] when it comes to number of recommendable entities. The association rule based models requires the least amount of memory and computational time for both of the datasets used in this analysis. They conclude by stating that while the introduction of RNNs in session-based recommendation has resulted in some promising results, there is still much work to be done into improving their handling of sequential information as simpler models have been shown to outperform them on many datasets.

3.2 Temporal modeling in recommendation

Even though RNNs are inherently able to capture some temporal dependencies, this is largely based on the order of the events in a sequence. Things like time gap between events or sessions, the season, the year, the weekday, and the time of the day, are all temporal aspects that may influence the ideal recommendation in many domains, but which cannot easily be captured by the order of events alone. There have been some attempts at both making recommendation time-aware in the sense of it being able to use such temporal aspects to improve its recommendation, and also some that combines this with attempting to predict the time until the next item or session by explicitly modeling time.

Liu et al. [2016] proposed a temporal-aware extension to RNN-based recommendation that is very similar to their aforementioned contextual extension. Here they re-apply the idea of training contextual dependent sets of weights, but this time the weights in question are used on the feedback loop in the RNN and not the new input. So the hidden state from the last timestep is modified by these temporal dependent weights. Because time is continuous it is not realistic to train different weights for each different time, nor would it serve any purpose. Instead,

the gap-time between events is made discrete by dividing the time-gap range into buckets and classifying the time-gap by the bucket containing it. The classification is then used to decide which weights that are to be used during training or recommendation. The aforementioned contextual extension can also involve temporal context like season, time of day etc and therefore further increases the temporal-awareness. Using the proposed model, they observed increased accuracy over their best baseline on two different datasets.

The model proposed in Du et al. [2016] is not just time-aware but puts effort into modeling the inter-event time itself. In addition to recommending the next item for a user, it attempts to predict the time of return, which is the time the user will return to the service, or the time until the next event. The architecture of the model consists of a single layer of RNN units, GRU units were used in the best setup, which is fed marker and timestamp pairs in sequences. The output is fed to two different linear layers for the marker prediction and the time prediction respectively. The time prediction is modeled as a marked point process with an intensity function which is conditional on the history and the time to the next event. The history component is the output of the hidden state fed through a linear layer and contains abstract information of both previous inter-recommendation gap times as well as the markers themselves. During training, the temporal part of the intensity function is the actual time target, while during the prediction it is an variable that is used in an expectancy calculation to find the prediction. The final loss function is a weighted sum of the temporal contribution and the marker contribution. These are respectively the negative-loglikelihood of the point process given the time and the negative-loglikelihood of the outputted multinomal score of the target marker. The results are compared with many different baselines for both marker prediction/recommendation and time prediction. Both real datasets and synthetically generated datasets, which were generated by common point processes, were used for the evaluation. The model outperforms all baselines on marker prediction and time predictions on the real datasets. For the synthetic time predictions, it achieves highly comparable results to models based on the actual point process that produced the data, and beats most other. This showes that the model is robust and able to capture a multitude of different time models. Lastly the model was trained using only marker prediction loss and time prediction loss respectively. This showed that models trained only using marker prediction loss performed worse on marker predictions than the one trained using both losses, and the same for the reverse case. Thus, proving the two different tasks are mutually beneficial in the proposed model.

The model proposed in Jing and Smola [2017] is similar to Du et al. [2016] in combining recommendation and time-modeling, but used survival analysis to model the time instead of a marked point process. Another difference is that this model is to be used on next-basket recommendation, not just a single sequence of items. Furthermore, the time modeling is on the inter-session time-gaps, not the time between each selection. The sessions are thought of as single entities with no internal dynamics, making the recommendation a next-basket recommendation. This means that they use a single level RNN for inter-session modeling directly, in contrast to Ruocco et al. [2017] and Quadrana et al. [2017], who both do some form of inter-session modeling, but do so in one of the two levels of their hierarchical RNN models. These hierarchical models use the other level, as previously mentioned, to do intra-session modeling. This is not considered in Jing and Smola [2017], and they can use a single level RNN model, making the main part of the architecture more similar to those of Hidasi et al. [2015] and Du et al. [2016] even though the modeling is higher-level. Using LSTM in the single level RNN gave them the best results, so they chose this over GRU. The LSTM is fed a pretty complex input that is a concatenation of four different embeddings. These are: an embedding of the actions in a full session, an explicit user embedding, an inter-session gap-time embedding and an embedding of weekday time. The last is information about time of day and weekday combined such that the model can distinguish between activity on, for instance, Monday morning from activity on Friday afternoon. The user embedding allows the model to personalize the model beyond the limited user history fed for each recommendation because the same user embedding is used in all of the user's training data. The output of the LSTM unit layer is, similarly to Du et al. [2016], fed to two different linear layers, one for time prediction and one for recommendation. The same output is also fed to a unit they call "3-way factor unit" along with the user embedding. The output of this is added to the output of the respective linear layers and used for the final recommendation and time prediction. In order to make the evaluation of the survival analysis simpler and more efficient, a complex integration is approximated by a summation. In order to do this, the model outputs a vector of rates for different time intervals, and a separate method for setting the final gradients had to be defined. They handled the infinite time horizon by setting a fixed upper time bound after which no prediction could be made. The final model's recommendation capabilities was compared with two factorization based baselines and one based on neural networks. It was shown to outperform all of these on two different datasets. The time prediction was compared with expressive point processes, one of which was a Hawkes point process, and achieved smaller time errors than all of these.

Another approach for modeling time in RNN-based recommendation architecture is to incorporate the temporal support into the RNN units themselves. Zhu et al. [2017] propose three LSTM variant with explicit temporal support through explicit time gates. They note that while RNNs have been shown to perform well in different recommendation setting, most recommendation setting have significant temporal aspects that are not present in NLP(Natural Language Processing) and other domains where RNN-models have excelled. RNN units are good at modeling orders of entities, but does not offer any inherent support of time intervals between the entities. That in mind, they propose three different LSTM variation with the intention of better incorporating the time into the sequence modeling. The simplest proposed architecture adds a single time gate, which is used to further filter the input. This way the time since the last recommendation can affect how the recommendation is considered in the current recommendation. Through the cell state of the LSTM, the influence of each time interval is propageted to future recommendation, thus making the model capable of long-term time consideration. The second and third proposed models both extend LSTM with two different time gates. The second architecture use one gate for filtering the last time-gap's influence on the current recommendation, while the second gate filters how much of it should be remembered long-term. The third architecture is similar, but combine the input and forget gate and makes the first time gate participate in the filtering of the previous cell-stat's contribution to the current recommendation. They evaluate the models by not assuming the presence of sessions and feeding in the user histories continuously. This was done on two different datasets and it was observed that all proposed models outperformed all evaluated baselines, the strongest of which being the session-based RNN-model from Hidasi et al. [2015]. Furthermore, they looked at the performance as the number of fed test-cases increased, which showed that the models were able to improve even after a great number of test cases. Thus exhibiting a capability of exploiting long-term dependencies very well. Finally they tested a dynamic training scheme where the models are first trained on long user histories, but is continuously updated by being trained on a fixed number of new observations as these become available. The motivation being that the old history might become increasingly irrelevant for the new recommendation, so the models could benefit from having seen the newest data. Doing this they observed a significant improvement in Recall both for their own models and for the Hidasi et al. [2015] model.

Chapter 4

Architecture

In this chapter the proposed model will be described in great detail. This will be done by looking at different parts of the system in turn and explaining the overall functionality and purpose of the parts. In the case where multiple options have been considered, the different options will be explained and the final choice will be justified. While the explanations will be thorough, they will, for the most part, not go into finer implementation details. The exceptions to this is where non-intuitive solutions were applied to handle challenges or assure correctness. In order to give a general understanding and overview, a high-level description of the model and its functionality will be presented in the first section along with a full figure. The code of all models and baselines, as well as pre-processing, testing, and datahandling, can be found in this GitHub repository [1].

4.1 Key idea

The key idea is to design a joint model based on a hierarchical RNN structure similar to those found in Ruocco et al. [2017] and Quadrana et al. [2017], thus, both capable of intra- and inter-session modeling. By also considering inter-session modeling, we want a more personalized model than the one used in Hidasi et al. [2015], and seek to further improve this by including explicit user modeling similar to what is done in Jing and Smola [2017]. The time modeling is to be based on the marked point process defined in Du et al. [2016], but where marker history with inter-**marker** time-gaps is switched out for abstract session representations, user modeling and inter-**session** time-gaps. Modeling the time between session is more similar to the motive in Jing and Smola [2017], however,

[1] GitHub repository: https://github.com/BjornarVass/Recsys

their model does not perform intra-session modeling and considers sessions to be single "baskets". Furthermore, we seek to provide an initial recommendation for the next session before getting feedback from said session. While the intra-session recommendation found in Hidasi et al. [2015], Hidasi et al. [2016], Ruocco et al. [2017] and Quadrana et al. [2017] all provide recommendations within sessions after every selection, none of them provide a recommendation for the very first selection.

4.2 Overview

The proposed model is essentially a hierarchical-RNN model with added temporal modeling. The inputs to the model are representations of previous sessions along with belonging contextual information, followed by the real-time selected items in live sessions. The outputs are the predicted time of the next session, the recommendation of the first selection in said session, and the live recommendations. The former two can be evaluated before the new session is initialized. The live recommendation, on the other hand, is dependant on feedback from the new session. The selected items are fed to the model, which then provides recommendation(s) for the next item(s). This last type of recommendation will be referred to as "intra-session recommendation" while the recommendation of the first item will be referred to as "initial recommendation". The main goal of the model is to provide estimates for the start-time of the next session. Additionally, the intra-session recommendation should ideally benefit from the time modeling or, at the very least, not deteriorate significantly because of it. The initial recommendation is mostly considered an added bonus that is not important enough to warrant any optimization that can negatively affect the other tasks. Figure 4.1 depicts the full proposed model. The figure is quite high-level and the finer details will be thoroughly described in the following sections. Key things to note are the Inter-session modeling and how data flows through this as well as the three different outputs of the model. The inter-session RNN and the intra-session RNN together makes out the hierarchical RNN which is described in detail next.

4.3 Hierarchical RNN

The Hierarchical RNN is inspired by work in Ruocco et al. [2017], and consists of an inter-session RNN and an intra-session RNN. In this model some small changes to the input of the inter-session RNN was done in order to better support the inter-session modeling and time modeling.

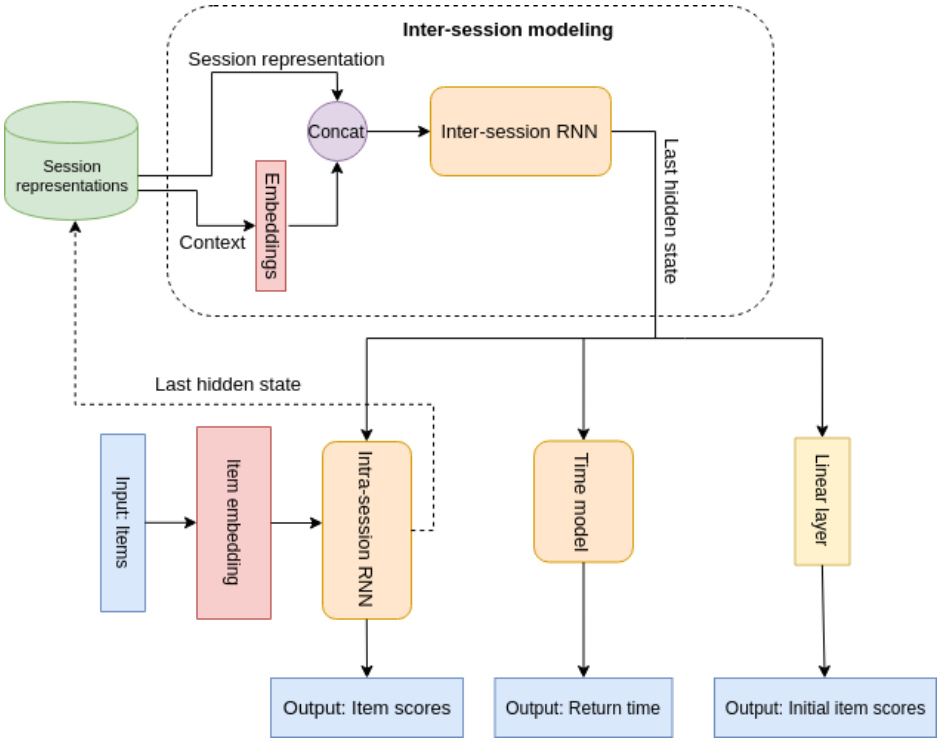

Figure 4.1: Figure showing the full proposed model. Some finer details have been abstracted away to make the figure clearer.

4.3.1 Session-representation

Ruocco et al. [2017] considered two different schemes for creating session representations. The first of which is to create an average pooling of all the item embeddings in the session. The other is to use the last hidden state of the intra-session RNN after the session to be represented has been fed to the network. We decided on focusing on the latter scheme after preliminary tests showed that this had better synergy with our model.

4.3.2 Inter-session RNN

Like in Ruocco et al. [2017], the inter-session RNN is fed a fixed number of preceding session-representations. However, in our model, relevant contextual

information is concatenated to this input. Specifically information about the user, and the time since the last session, in the form of embeddings. The embeddings will be described in further detail in 4.4. Figure 4.2 illustrates how the

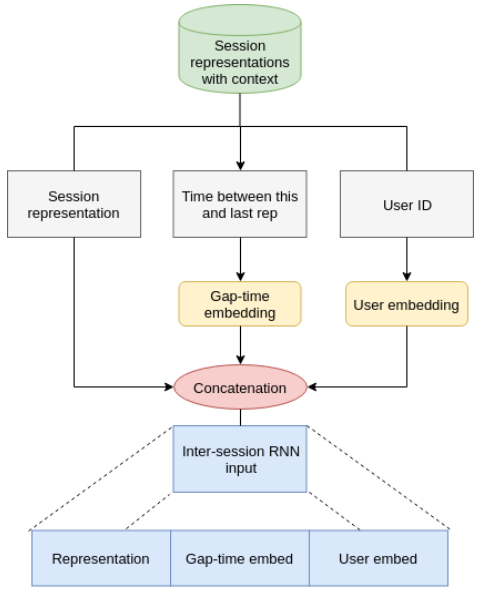

Figure 4.2: Figure showing the embeddings and concatenation involved in providing input to the inter-session RNN

different embeddings are gathered and concatenated with the session representation to create the input to the inter-session RNN.

Despite the additions, one of the tasks of the inter-session RNN remains to output an initial state for the intra-session RNN, like in Ruocco et al. [2017]. This is done by propagating the final hidden state of the inter-session RNN, which serves as an abstract representation of the inter-session information/user behavior. The intuition is that this information could help the intra-session recommendation to identify, in fewer steps, the current interests of the current user, and circumvent the cold-start problem. In our model, we use the same last hidden state for the two new tasks of predicting the time until the next session and the first selection within it. By including additional contextual information in the inter-session modeling part of the model, we seek to support the two new tasks while continuing to support the original intra-session recommendation task.

4.3.3 Intra-session RNN

The intra-session RNN uses the last hidden state of the inter-session RNN as initial state, and is then fed embeddings of the items in a new/live session. For each item embedding in the input, the corresponding output of the RNN is passed to a linear layer which outputs scores for each target items. The recommendation given the last item and the hidden state of the RNN, will be the item(s) with the highest score(s). The intra-session RNN will inherently perform intra-session modeling since it is fed the session selections in sequence, and by having the inter-session RNN provide the initial hidden-state, the intra-session recommendation will be based on both intra- and inter-session modeling.

4.3.4 RNN choices

GRU units are used in both the intra-session and the inter-session RNNs. GRU was chosen due to its ability to remedy the vanishing gradient problem and since it was found to work better than LSTM for this problem and model structure. Of the many different flavours of GRUs, the decision fell on the default GRU implementation found in the deep learning framework that was used, PyTorch. Optimization through choice/design of RNN-cell is not considered to be within the scope of the thesis, so we did not spend any effort beyond testing both the default LSTM-cell and the default GRU-cell before making the decision. The update rules that are used can be found in equation 2.3, where σ_g is set to be the sigmoid activation function and σ_h is set to be the hyperbolic tangent activation function

Dropout is applied to both the input and the output of both RNNs. While Ruocco et al. [2017] observed improvements only when applying dropout when considering the LastFM dataset, dropout was found to also improve the performance of our model on the Reddit dataset.

4.3.5 Last hidden state extraction

The extraction of the last hidden state has to be done in both the inter-session RNN and the intra-session RNN. The final hidden state of the inter-session RNN is used for all three model tasks, and the final hidden state of the inter-session RNN is needed for storing "hidden state"-based session-representations. Due to the use of padded sequences, where paddings are ignored, this extraction is not completely straight forward. Because of the highly parallel nature of SIMD(Single Instruction Multiple Data) operations, which are used extensively in ANN computations on both GPU and CPU, it is in many cases more efficient to compute the output for both valid entries and paddings alike. Trying to filter out the paddings

during computation can cause a lot of branching which is handled poorly by SIMD operations. Thus, the correct final hidden states is gathered from the full output by using the non-padded sequence lengths to index the outputs.

4.4 Embeddings

There are a total of three different types of embeddings used in the model. The purpose of embeddings is to learn finer dynamics and representations of the embedded entities. For instance, if two different artist often are listened to by users with similar tastes, an embedding layer would most likely learn artist representations that are quite similar to each-other. By using RNNs, learned representations can reflect certain temporal dynamics in addition to the more general "similarity measure". A simplified example of such representational knowledge could be: "Artist A is almost always listened to before Artist B". Embeddings can also learn dissimilarity and non-linearity. For instance, when considering time-gaps between sessions, there might be a higher correlation between gap durations of 24 and 48 hours(daily/periodic behaviour) than between 12 and 24 hours, even though the quantitative difference is less in the latter example. The dimensional size of the embedding directly affects its expressional complexity, consequently the size of the embeddings used should be scaled by how complex the contextual data is and how many unique embeddings are needed.

4.4.1 Item embedding

First and foremost is the embedding of the input items. This is a rather large embedding table with an embedding for each unique item in the dataset as well as the 0-embedding reserved for the "padding-item". The 1-indexed item IDs are used to index their respective embeddings. The learned embeddings are only directly handled by the intra-session RNN and consequently only trained by the resulting loss of this part of the model. However, both schemes for getting session-representations are in some way affected by the item embedding. This means that the item embeddings will affect earlier parts of the network, but then as input, not computational graphs. Hence, their gradients are unable to flow back to the embedding layer, and cannot be used to train the embeddings further. An additional detail here is that the 0-entry of the embedding table could be set to be a vector of zeros. This will allow average pooling to be done haphazardly over all item embeddings in a session because the 0-embeddings will not contribute with anything, which is useful in case the average pooling session-representation scheme is to be used. The number of valid embeddings can easily be found using the sequence lengths and be used in the division to get a representative average. Because the model ignores the paddings, this embedding is never updated during

training meaning that the described strategy above can always be used. Due to the large number of items present in all considered datasets, the dimensionality of this embedding is the largest by far.

4.4.2 Inter-session gap-time embedding

The time gaps between sessions are first normalized and then divided into discrete buckets. The resulting bucket ID's can then be used to index embedding tables/layers to propagate the corresponding embedding. Two different normalization schemes were examined, each having their own pros and cons. Both are first given an upper bound, which sets a threshold of the time-gap after which we don't consider the user active enough to be provided accurate time predictions. The time-gaps that are greater than this bound is set to the upper-bound, resulting in them ending up in the very last embedding no matter the normalization scheme. The more straight-forward of the two normalization schemes, simply divides the time-gap range into uniformly sized buckets. The benefit of this is that all the values in the time-gap range will belong to a bucket of equal size, causing no time-gap to be in the same bucket as a much higher or lower time-gap. A disadvantage of using this is that the earlier "popular" buckets can be overcrowded and the later ones can end up being almost empty, which can make such embeddings hard to train. One also needs a high resolution to cover the finer differences in the smaller time-gap ranges, which further increases the problem of sparse buckets. The second normalization looked at is one where the time-gaps are first transformed with a log function, before the transformed range is divided uniformly into buckets. This results in a more evenly distributed number of observed time-gaps in the different buckets, but at the cost of cruder resolution for larger time-gaps where the corresponding buckets covers much more time in the log transformation than the earlier ones. The larger time-gap buckets may therefore become over-generalized. It was found that the log scaled scheme worked pretty good for small resolutions, but was overall out-performed by the uniform scheme with higher resolution. Since having high resolution was observed to be a non-issue, both with regards to model performance and run-times, the uniform scheme was deemed the better option.

4.4.3 User embedding

User embedding is an explicit embedding of a user that can learn user specific behaviour beyond the limited history of session representations. This is inspired by Jing and Smola [2017] where the same type of embedding was used. This could be very useful when there are long user histories. For instance, if a user behaves a bit unusual and sporadic in the last few sessions, a model with user embeddings can have information about long term user behaviour, which can help

making sense of/override the recent noisy behaviour. A slight problem of using such embeddings is that the model cannot easily be used for new users without extensions. A way to handle this problem could be to add an "unknown user" embedding. This could for instance be trained on the known users after the main training, but only updating the weights in the user-embedding. This is far from optimal since the rest of the model might be biased towards having specific, as opposed to general, information about the user. In this case, the training of the general embedding would just attempt to make the best out of a bad situation. For our use-case, it was deemed reasonable to only consider known users since improved personalization is a sub-goal of the thesis. Thus, all setups of the model utilize explicit user embeddings.

4.5 Time modeling

The main goals of this model is to predict the time until the next session is initialized, given a fixed length history in the form of abstract session representations and corresponding contextual temporal information. This is both because knowing the time of return can be very useful but also to examine whether the improved time modeling can improve the other tasks through better utilization of temporal information, and the other way around. By basing the time prediction on previous time-gaps and user selection history, we hope to achieve better time prediction than those of models that only considers time-gaps alone.

4.5.1 Parameterization

The time modeling is heavily inspired by Du et al. [2016], where time modeling is used to both predict the time of the next item recommendation given a single sequence of previous items, and to improve the recommendation itself. In their model, the time-gaps between selected items are considered to be drawn from a marked point process. The parameterization of the marked point process is defined by the authors, and is both dependent on the previous selection history, with corresponding inter-selection time-gaps, and on the time since the last selection. The most important detail here is that the history is provided by an RNN. Since the output of RNNs can contain abstract information about previously processed input, RNNs can be used for creating an abstract history or summary of inputted sequences. Instead of devising a complex scheme for parameterizing all the history and the contained dependencies, this is left completely up to the output of an RNN. We adapt this marked point process for applying it in our hierarchical model. The main difference is that the main part of our history are abstract session-representations and not individual items. Consequently, the corresponding time-gaps are the times between the concurrent session representa-

tions in the history. This inter-session modeling with application of concatenated embeddings is more similar to what was done in Jing and Smola [2017]. However, Jing and Smola [2017] did not consider any intra-session modeling and used a non-hierarchical single layered RNN in a next-basket recommendation model.

$$\lambda^*(t) = \exp(v_t^\top \cdot h_j + w_t \cdot (t - t_j) + b_t) \tag{4.1}$$

The resulting intensity function can be seen in equation 4.1, where h_j is the j-th hidden state, t_j is the last timestamp in the last session, t is the time variable, v_t is a vector of weights with the same dimensionality as h_j, w_t is a single weight and b^t is a bias term. $v_t^\top \cdot h_j$ comprises the historical influence on the intensity function, while $w^t \cdot (t - t_j)$ is the current influence. Note that t and t_j always appear together. $t - t_j$ can therefore be replaced with g_j which is a variable of the time since the last session. We will use this notation when we consider fixed time-gaps, as provided during training. When considering time-prediction we will stick with $t - t_j$ as this makes it clear that the time-gap is dependent on the variable t.

$$f^*(t) = \lambda^*(t)\exp\left(- \int_{t_j}^{t} \lambda^*(\tau)d\tau \right) \tag{4.2}$$

$$f^*(t) = \exp\left\{ v_t^\top \cdot h_j + w_t \cdot (t - t_j) + \frac{1}{w}\exp(v_t^\top \cdot h_j + b_t) - \frac{1}{w}\exp(v_t^\top \cdot h_j + w_t \cdot (t - t_j) + b_t) \right\} \tag{4.3}$$

The full conditional density function of the marked point process is given in equation 4.2. In equation 4.3 the intensity equation 4.1 has been substituted into the conditional density function 4.2, resulting in the full expression of the marked point process. Time was modeled as number of days, both because this granularity fits all evaluated datasets, and since finer granularites made it difficult to scale the learning rate of time specific weights. Scaling of learning rates was found to be necessary in order to avoid that these weights diverged/exploded, which is not all that surprising, considering that the time-gap is multiplied with a weight within an exponential function in the time-loss equation 4.6.

4.5.2 Temporal additions to the model

The three main additions introduced in order to model the time are: the weights v_t and w, and the bias b_t. v_t is a vector of weights and b_t is a scalar bias term. In the model, these two terms are combined in a single linear layer with bias. This linear layer is applied to the output of the inter-session RNN and results in a single scalar output. w is a single weight and is also modeled as such in the model.

4.5.3 Performing the time predictions

An intuitive way of getting a single prediction from a distribution is to find the expectation, which essentially is a weighted sum/area over the probability distribution. For discrete distributions, the expected value can be found by summing the products of the discrete entities and their respective probabilities. In the same estimation for the continuous case of density functions, the summation is replaced by integration.

$$\hat{t}_{j+1} = \int_0^\infty t \cdot f^*(t)dt \tag{4.4}$$

Equation 4.4 is the expectation calculation of the distribution, where $\hat{t}_{j+1}$ is the expectation and is considered to be the time prediction. A slight challenge is that the density distribution equation of the point process 4.3 cannot be integrated analytically. Thus, prediction has to be approximated by numerical integration. To handle the infinite upper integration bound, a simple upper cut-off time is defined. While this is another approximation, the approximation becomes negligible when setting the cut-off time a bit higher than the vast majority of gap-times found in the data. Additionally, defining a point in time after which predictions cannot be made, can help the model perform better by reducing the impact large infrequent time-gaps has during training, relative to the impact of smaller and more frequent ones. Focus put into modeling infrequent time-gaps is, more often than not, wasted effort in the first place. Firstly because there are not enough data beyond a certain point to accurately learning such predictions. Secondly, because for many services the users can safely be considered inactive after a certain time of inactivity has passed. If, or when, the user becomes active again is a problem of its own, and will often only be noise in a recommendation setting. In general, uncertainties tend to grow larger with time, making it difficult to make accurate predictions far into the future.

The model utilizes two upper bounds concerning time-gaps. The first of which describes the upper target bound. Time-gaps that originally are larger than this are set to the upper target bound before they are used as time prediction target and before their time-embedding is retrieved in order to be fed to the inter-session RNN. The second bound is the upper integration bound and is set a bit higher than the upper target bound. If this had been set to the same as the upper target bound, the modeled probability distribution would have to be an infinite valued spike at the upper target bound for the model to output the upper target bound. By setting the upper integration bound a bit higher, we don't have to expect the modeled probability distribution to have such infeasible shape in order for the time predictions to be evaluated to a value close to the upper target bound. Consequently, this theoretically opens up the possibility of predictions that are

greater than the upper target bound. For reasonable upper bounds, this will most likely never happen due to loss function optimization and the discussed futility of predicting far into the future.

4.6 Initial recommendation

In Ruocco et al. [2017] the model is only applied after a user has selected an initial item, which is fed to the model for recommending the second item. Our model predicts the time until the start of the next session as well. This being the case, it would be useful to also predict/recommend the very first item as well in order to provide the user with an initial recommendation when she eventually do return. In order to do this, a single linear layer is added to the model. The new layer is fed the last hidden state of the inter-session RNN, like the time modeling and the intra-session RNN. The goal is to use inter-session information from the hidden state to recommend the first item in the future session. The layer outputs scores for all the recommendable items and is therefore very similar to the linear layer after the intra-session RNN. Figure 4.3 shows the data-flow and different

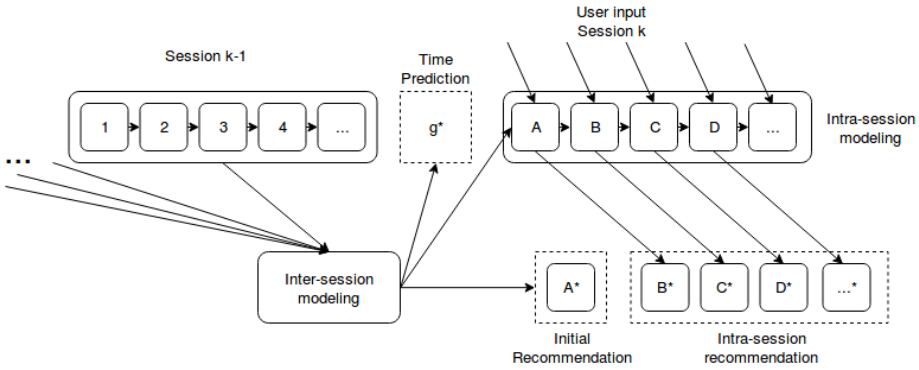

Figure 4.3: A simple figure summarizing the dataflow and different tasks preformed by the model. The different tasks are illustrated by three dashed boxes.

tasks of the model with an example session containing the items A, B, C, D etc. The individual recommendations have been denoted with the target letter, but also with an asterisk to make it clear that the recommendation might not be correct. Note the difference between the initial recommendation and intra-session recommendation, as this might not be apparent without experience in dealing with sequence models. The figure also makes it clear how the time prediction

and initial recommendation only considers inter-session modeling, while the intra-session recommendation also have intra-session modeling available.

4.7 Loss function

As stated in the overview and later individually elaborated, there are three different outputs of the full model. These are the intra-session recommendation, the initial recommendation and the prediction of when the next session will start. Both recommendation outputs are in the form of non-normalized scores for the recommendable items, while the time output is the scalar contribution of the history $v_t^\top \cdot h_j + b_t$ from Equation 4.1. These two types of output are treated by different types of loss functions, and then all three losses are combined for the backpropagation.

4.7.1 Recommendation loss

For both different recommendation losses, cross-entropy is used. Cross-entropy works by first normalizing all values between zero and one, by applying softmax, and then taking the negation of the log transformation of these values. Usually when applying cross-entorpy loss, all losses except that of the actual target are evaluated to zero, which is also the procedure applied in this model. Negative log transformation can be used as loss because the scaled scores are all between zero and one. In the ideal situation, the scaled score of the target will be one, making its negative log transform zero. Negative log transformation diverges towards infinity when the input approaches zero, meaning that small scores will result in large losses.

The model ignores losses originating from padded items by masking these out. This was deemed the best option when judging pros and cons of ignoring/partly ignoring padded items. If all losses had been used, the padding item would be the most frequent item in all datasets, and would most likely be recommended often. An hybrid option is to consider the loss of the first padded-item, but ignoring the remaining. This could make the model consider the padding as an "end-of-session symbol", consequently teaching the model to recommend the padding item when it believes the session is over. While such results could be interesting, the use-case of such knowledge is pretty limited and was therefore not applied in this model.

$$L_{rec}(s_{j+1}, i) = -\log\left(\frac{e^{-s^i_{j+1}}}{\sum_{k=1}^{N} e^{-s^k_{j+1}}}\right) \tag{4.5}$$

Equation 4.5 is the full expression of the loss criterion used for recommendation. i is the id of the target item, s_{j+1} is the item scores and N is the total number of unique items.

4.7.2 Time loss

Any conditional probability density functions will result in "one" when integrated from negative infinity to positive infinity by definition. The higher the values in an interval in the density function, the higher the probability of of drawing from said interval. Ideally, one would like value of the density function to be as high as possible given the target. This means that the negative log transformation could be used for getting a loss measure for time as well.

While the integration over the legal range over any probability density function will result in one, this does not restrict it from being greater than one in small intervals. Thus, we might end up taking the negative log transformation of a value greater than one, which results in a negative loss. This does not cause any problems since the object is not to have a loss of zero but to minimize the loss. A negative loss is obviously less than zero and therefore even better than zero.

$$L_{time}(t_{j+1}, h_j, w) = -\left(v_t^\top \cdot h_j + w_t \cdot g_{j+1} + b_t + \frac{1}{w}\exp(v_t^\top \cdot h_j + b_t) - \frac{1}{w}\exp(v_t^\top \cdot h_j + w_t \cdot g_{j+1} + b_t)\right) \tag{4.6}$$

The resulting loss is the negative log transformation of equation 4.3 which can be seen in equation 4.6, where g_{j+1} is the target time-gap.

Finally we introduce a new tuning parameter for customizing the impact of the target time-gap. The key motivation for this introduction was the observation that the large losses from longer time-gaps could make the model focus on middle-long time-gaps even though both datasets mostly contain short-time gaps. The hyper-parameter, κ appears as an exponent of the time-gap. Setting it to 1, makes it identical to 4.6 but if it is set somewhere between 0 and 1, it will scale down the contribution of the longer time-gaps relative to the shorter ones. It could also be set to a value higher than 1 if one wants to scale the contribution of the longer time-gaps up relative to the shorter ones. The tuning parameter can be set differently, based on observation and motives, for different datasets with different time-gap distributions.

$$L_{time}(g_{j+1}, h_j, w, \kappa) = -\left(v_t^\top \cdot h_j + w_t \cdot g_{j+1}^\kappa + b_t + \frac{1}{w}\exp(v_t^\top \cdot h_j + b_t) - \frac{1}{w}\exp(v_t^\top \cdot h_j + w_t \cdot g_{j+1}^\kappa + b_t)\right) \tag{4.7}$$

The final time loss expression ends up being 4.7 where we have added the κ tuning parameter. The introduction of κ is also supported by looking at the gradient of the negative log-likelihood with respects to w^t: $\frac{\partial -\log f^*(t)}{\partial w_t} = -g_{j+1} + \frac{c_1}{(w_t)^2}\exp(g_{j+1}w_t)(g_{j+1}w_t - 1) + c_2$ (where c_1 and c_2 are fixed terms which do not depending directly on the time-gap g_{j+1} or the weight w_t). This expression has two main conditional contributions: one linear from the time since last session(short-term) and one exponential from the history of past interactions(long-term). Hence, applying an exponent to the time-gap in equation 4.6 will affect these two contributions differently, which can be exploited in order to tune the influence of long and short time-gaps relative to each-other. This intuition can also be seen by observing that the gradient is approximately linear for small time-gaps, but exponential for long time-gaps.

4.7.3 Combined loss

In order to combine the different losses into one single loss, every different loss are summed up over the entire mini-batch. In order to get the most accurate feedback during training, the average of the losses is found by dividing the summed losses with the number of **non-padded** contributions. Otherwise a mini-batch with a lot of padded items could appear to have a much smaller loss than it really does.

$$L_{total} = \alpha L_{time}^{avg}(G_{j+1}, H_j, w, \kappa) + \beta L_{rec}^{avg}(S_{j+1}, I) + \gamma L_{rec}^{avg}(F_{j+1}, K) \qquad (4.8)$$

The combined loss is set to be a weighted sum of the three different averages and is given in equation 4.8. I is the target items of the intra-session recommendation and K is the target items of the initial recommendation. S_{j+1} is the scores of the intra-session recommendations, F_{j+1} is the scores of the initial item predictions and G_{j+1} is the target time-gaps. The ^{avg} superscript on the loss contributions is added to indicate that this is the averaged losses that disregards paddings, and that this is the full expression of the mini-batch loss. Note that for each session there are only one initial recommendation target and one time target, but there are necessarily as many intra-session recommendation targets as the current session-length. α, β and γ are scaling factors used to weight the different loss contributions and are hyperparameters of the model.

4.8 Training

The training of the model relied heavily on PyTorch [2], which is the framework used to implement the model. No changes were done concerning the gradients and how these were found. Additionally the library implementation of the Adam

[2]PyTorch GitHub repository: https://github.com/pytorch

optimizer [Kingma and Ba, 2014] was used with the default parameters. This was chosen for its use of momentum and adaptive learning rates while remaining computational viable. Dropouts and consequent output scaling are used during training, but turned off during evaluation. This is also handled at a library level. The use of mini-batches deserve further explanation and is given its own sub-section.

4.8.1 Mini-batch scheme

The mini-batches were created such that no users were represented with more than one session in any mini-batch. Otherwise some mini-batches could end up specializing the model too much towards a single user. Because the users can have different numbers of sessions, a scheduler, that prioritize users with many sessions left, was implemented. For significant differences in number of sessions per user, the scheduler can still end up outputting many mini-batches towards the end with very few user. This is because even though some users are present in every single mini-batch, if their number of sessions is great enough, most other users may exhaust all their sessions by filling the remaining slots in the mini-batches before the last batch is compiled. This is avoided by stopping at the moment a mini-batch is less than half-full. Because the scheduler has to prioritize users with many sessions and we split training and testing data chronologically, the scheduler ends up being deterministic. This means that each epoch goes through the same mini-batches in the same order. Ideally, the scheduler should be randomized, but this is difficult to achieve without the overhanging probability of ending up with many more half-full mini-batches. Considering the relatively small number of epochs needed for training, a badly distributed epoch towards the end of training might end up adding a lot of bias to the final model. In the end, the deterministic option was deemed to be the better of the two, which is partly justified by the fact that many of the potential problems with deterministic batches are remedied by using relatively large datasets. For instance, having many mini-batches makes it improbably for the model to overfit on the mini-batch order.

Chapter 5

Experimental Setup

In this chapter will describe everything related to how the experiments were conducted. This involves a discussion of the different datasets that were used for the evaluation as well as how these were pre-processed. Then the hyperparameter settings will be listed and explained, before rounding off by describing the baselines and the different experiments.

5.1 Datasets

The evaluation was done on two different datasets. These are the LastFM dataset from Bertin-mahieux et al. [2011] and the Reddit dataset [1], and are the same datasets as the ones used in Ruocco et al. [2017]. Last.fm is a music website where users can keep track of the songs they listen to as well as sharing this with their peers. Its data is in the form of tuples containing user-id, artist, song and timestamp, each representing a single listening event. Reddit is a popular forum/discussion website, where users can share and comment on different news, creations, pictures and other topics. Its structure is divided into different subforums, named subreddits, which define the topics of the posts to be posted there, the interests or affiliations of the users who frequent them. This data is in the form of tuples containing a user, a subreddit and a timestamp, and each of these represents a single event where the user has commented on a post within the specific subforum at the time of the timestamp. Table 5.1 contains key statistics of the final pre-processed datasets used in the different experiments.

[1] Subreddit interactions dataset: https://www.kaggle.com/colemaclean/subreddit-interactions

	Reddit	**LastFM**
Number of users	18,271	977
Number of sessions	1,135,488	630,774
Avg # sessions per user	62.1	645.6
Average session length	3.0	8.1
Number of items	27,452	94,284

Table 5.1: Table containing statistics of the different datasets after pre-processing

5.2 Pre-processing

The format of the data in both datasets are quite simple, but it was deemed necessary pre-process the data in a few different ways. These are mainly concerned with ordering it into distinct sub-sequences/sessions, removing noisy or irrelevant data and defining the items/markers in the data. Most of the pre-processing is done exactly like in Ruocco et al. [2017], with the exception of a small change applied during the splitting of long sessions to ensure that time modeling only considers time between identified separate sessions.

5.2.1 Dividing into sessions

Neither of the datasets were explicitly ordered into different sessions, so this had to be done in the pre-processing. The scheme for doing this is based on the time between the consecutive events for a single user. If the inter-event time is above a certain threshold, it is assumed that they belong to different sessions. For the Reddit data, this threshold can be quite large was set to 1 hour or 3600 seconds. This is justified by the fact that a user might still be browsing subreddits, posts and comments, but be perceived as inactive because they don't comment themselves. The threshold can arguably be set a bit less in the LastFM dataset, because the events represents songs listened to, and the vast majority of songs last way less than 1 hour. The threshold for the LastFM dataset was set to half an hour or 1800 seconds, which should cover the length of most songs and also allows for small breaks without identifying continued activity as a separate new session.

During run-time, the final sessions are split into input and target sequences, where the input sequence is the full session except for the last item. Conversely the intra-session recommendation target sequence is the full session except for the very first item. In order for this to work, the session must at least be two items long, which is enforced throughout the following pre-processing steps.

5.2.2 Marker definition

The markers/items in the Reddit set was simply set to be the subreddit provided in each data entry. In the LastFM set however, the items were set to be the artist rather than the song title. This was done because the sheer number of different song titles is very great and the resulting data would be extremely sparse, making it highly unlikely that any decent recommendation accuracy could be achieved. The number of different artists is still very large and the resulting data is quite sparse, but not exceedingly so. For the most part, people tend to like songs from the same artist or at least have a bias towards liking other songs of the same artist. Thus, considering two different songs of the same artist as the same marker, is not completely unreasonable.

5.2.3 Removing consecutive reoccurring items

In some cases, a user might comment multiple times in a row on posts belonging to the same subreddit, or listen to the same song or artist multiple times in a row. While this is valid behavior, to feed in such data could train the model to often output the same marker it was fed. It is not very useful to recommend a user posting in a certain subreddit, that same subreddit, nor recommending a user listening to a certain artist to listen to the same artist. In both cases the user is perfectly aware of the item they are currently "consuming" and does not need to be reminded of it. Such recommendation could be useful in a more complex model where one could do further recommendation by first recommending subreddit/artist then post/song, but such behaviour is not considered for this model. Thus, consecutive reoccurring items are handled by reducing these to only one occurrence.

5.2.4 Splitting long sessions

While real sessions lengths can have arbitrary lengths, this is difficult to incorporate in the training of RNNs. The sequences need to be unrolled for a certain number of timesteps in order to get meaningful contributions of the previous timesteps. While one can train for half a session and then continue in the next mini-batch by feeding in the last hidden state of the first half of the session, it is difficult to allow the gradients to go across separate mini-batches. This is because the initial hidden state will be a scalar input, not a computational graph that can be backpropagated over. Additionally, certain trends for session lengths can usually be found, so setting the maximum session length to a value that is a bit higher than the average length, can often cover the vast majority of the sessions. There are a lot of benefits by setting a maximum session length that covers most, but not all, session lengths when using session-paddings. For instance, setting the

maximum session length to the longest observed session, will most likely result in session-paddings making up a large fraction of the total data, which is very inefficient. The largest session lengths can additionally be outliers which ideally should be ignored by the model and definitely not have a large say in its design. Furthermore, even though GRU and LSTM can learn long dependencies, this is not by any means unbounded in practice. While GRU/LSTM models usually perform better after being provided e.g. 5 vs no items in a sequence, but the difference is usually pretty small between e.g. 20 and 40 items. This in mind, a maximum session length L was defined and set to be $L = 20$, a number found by looking at the different session length statistics of the datasets. Sessions with lengths l in the interval $L < l < 2L$ were split into two separate sessions. Sessions with lengths $l >= 2L$ were discarded entirely, because such behaviour was found to be significantly rare and raising the possibility that they stemmed from bot activity. Note that this is done after other pre-processing steps that affects the individual session lengths.

Because sessions that are split because of their length are not considered separate sessions, trying to model the inter-session gap-time of these could be a source of noise. This is avoided by setting the last timestamp in the first half, and the first timestep in the second half, to be the start time of the full session. This way the inter-session time-gap will be calculated to be 0 and the contribution will be masked away from the time loss, making sure that the model does not train on these time-gaps.

5.2.5 Paddings

After the sessions have been split, the sessions are still of varying lengths, but now no sessions are longer than L. In order to have equally long sessions that easily can be handled by RNNs, the sessions have to be padded such that all sessions are of length L. Sessions were therefore padded with 0s. Note that legal items have been given unique integer ids starting with 1 and ending with the number of unique items. This way the paddings can easily be distinguished from the actual items, and subsequently ignored by the model.

5.2.6 Splitting into test and training set

In the end the datasets are split into separate testing and training set. 80% of the data was put into the training set and the remaining 20% went into the testing set. The split was done for each user and chronologically, such that the most recent 20% of a users sessions went into the testing set. The intuition behind this is that a recommendation system should strive to make the best recommendations for the future sessions. Trends and interests comes and goes, so achieving good

results on the newest data should be the best way of evaluating the model.

A potential problem with the split strategy described above is that splitting within each user according to a fraction will usually not result in a fine chronological split. One user might have ceased to be active, thus having only old sessions, while another might just have started using the service. This can result in hindsight information being available for the models to exploit. Say one artist becomes very popular at one point, and that this is covered in the training set of most users. The model then has the ability to generally favour this artist more than if the popularity surge was not observed at all. When a user that have not observed this surge in popularity within its training data is evaluated, the model may appear to anticipate a sudden interest in said artist around the point the artist became popular. In reality it may simply have exploited hindsight information from other users. However, note that since the baselines and the proposed model are tested on the same datasets, any hindsight information will be available for all of them. So unless one model is able to exploit such information much better than the others, the comparison can still be reliable. It is possible to create a dataset which is split strictly chronologically, but this does not guarantee anything about how the users' data is distributed between testing and training. Since the proposed model utilizes explicit user embeddings, users with little or no training data will have close to randomized embeddings, or highly biased due to having observed very few observations. A solution could be to balance the users better chronologically in the pre-processing, but then one would have to disregard some data. In the end we opted for the the first strategy since this requires less pre-processing, ensures good user data distribution, utilizes almost all data and since we have no reason for believing that any model will be able to exploit hindsight information better than the others.

5.3 Hyperparameters

The model contains a large number of hyperparameters, three of which are set specifically based on the dataset used, while the others are universal. The sheer number of hyperparameters combined with the great training time means that it cannot be claimed with confidence that all are set optimal, but the settings are of course the best values that were observed.

5.3.1 Data-specific hyperparameters

Table 5.3.1 contains the data specific hyperparameter settings. The number of different items or classes is much higher in the LastFM dataset than in the

Hyperparameter	Reddit	LastFM
Item embedding size	50	100
Epoch nr	28	24
Min-time	1h	0.5h

Table 5.2: Table containing dataset specific hyperparameters

Reddit dataset, so it is not surprising that the best embedding size was found to be greater as well.

5.3.2 Universal hyperparameters

There are a great number of universal hyperparameters so these can for structuring purposes be divided even more.

Learning and loss

Hyperparameter	Value
Learning rate	0.001
Learning rate, time	0.0001
Dropout rate	0.2
α	0.45
β	0.45
γ	0.1

Table 5.3: Table containing universal hyperparameter settings concerning learning and loss function.

Table 5.3.2 contains the different learning rates, the dropout rate and the settings of loss scaling hyperparameters found in equation 4.8. The learning-rate of weights w in equation 4.6 had to be reduced because otherwise it has a tendency of diverging, which caused other weights to diverge. The learning rate of v_t in the same equation was also reduced the same amount because otherwise the time loss oscillated heavily between batches. These are collectively referred to as "time" in table 5.3.2 because they are the only parameters strictly related to the time prediction.

Hyperparameter	Value
Batch size	100
Max Sequence length	19
Max number of representations	15

Table 5.4: Table containing universal hyperparameter settings concerning batch control

Batch control

Table 5.3.2 contains the hyperparameter settings concerned with batch control. These are the number of sessions in one batch, how long the sessions are and how many previous session representations are provided for each session. The sequence length is one less than the session length described in the pre-processing because the input sequence and intra-session recommendation target sequences are both missing one item each(last and first items respectively) of the full session.

Context embeddings

Hyperparameter	Value
User embedding size	10
Time-gap embedding size	5
Time-gap embedding number	500
Upper time target bound	500

Table 5.5: Table containing universal hyperparameter settings concerning the context embeddings.

Table 5.3.2 list universal hyperparameters concerned with the context embeddings. The number of user embeddings is not listed as this is not a hyperparameter since it is set by the number of unique valid users in the evaluated dataset. The table also contains the "upper time target bound" as this is a hyperparameter that affects the time-gap embeddings.

5.4 Temporal baselines

The time results were compared to Hawkes-process, see section 2.3.2, based baselines. A Hawkes-process was chosen because it is good for modeling situations where activity increases the probability of more activity in the near future. This

is often a simple, but good, approximation of user behaviour in many different domains. This allows Hawkes-processes to model periodic behaviour, that switches between frequent activity and infrequent activity, much better than for instance a Poisson-process, which assumes a static rate of activity.

5.4.1 Baseline hyperparameters

Hyperparameter	**Value**
History length	15/training data length
Number of prediction samples	100

Table 5.6: Table containing hyperparameter settings concerning the temporal baseline. The history length has two alternatives since the baseline was fitted in two different schemes.

Table 5.4.1 contains the hyperparameters settings of the Hawkes-based baseline.

5.4.2 Setup

Users are considered to be independent so the Hawkes-process baseline is used to model one user at a time and having one dimension. For each time prediction, the Hawkes-process is fitted on the last 15 time-gaps observed by the same user with a MAP EM method (Maximum APosteriori Expextation-Maximization) [Morse and Chodrow, 2016]. 15 was chosen since it is the same as the number of previous session representations, consequently also the number of time-gaps, the proposed RNN-based model is provided. The hope is that the limited recent history will contain traits of the user's current short-term behaviour but at the same time also contain traits of some of the more "static" long-term behaviour. This is used throughout the testing, meaning that the baseline is trained/fitted on data that is only tested on in the proposed RNN-based model. This is a feasible solution since fitting the Hawkes process on few observations can be done in run-time, which means that there is no real reason not to use the most recent data.

Simply fitting a Hawkes-process and using it to generate a single event, cold-start, would return an exponential distributed prediction with intensity equal to the base intensity of the Hawkes-process. In order to account for how the previous events affect on the intensity function, the initial intensity is simulated by feeding in the same 15 most recent time-gaps. Generation is done by applying an Ogata's thinning method [Ogata, 1981]. Ogata's thinning is used to simulate time dependent intensity functions by considering the intensity to be part-wise

static. Events are generated using the newest evaluation of the intensity, but these can be rejected according to a probability that decreases over time based on re-evaluations of the intensity. Because the generation can vary heavily, it is done 100 times for each prediction, where the resulting numerical prediction is the average time of the first generated event in the 100 runs. The implementation was adapted from [2], and the main changes are the added abilities to simulate intensity during a sequence of events and to provide an initial intensity to the generation. This setup of the baseline will be referred to as "short-term fitted" in order to differentiate it from the following alternate baseline.

5.4.3 Long-term fitting alternative

A second "long-term fitting" scheme was also applied using the same parameterization. For each user, the Hawkes-process is fitted once on the full training-set, which then is used for the full testing-set with no re-fitting. This allows the baseline to have observed the full training-set, which is the same as the proposed RNN-based model. However, due to the limitations of the baseline parameterization, it will be very difficult to both model long-term and short-term dependencies. The result will likely be parameters tuned to maximize the long-term performance on the full training data, weighting the old observation equally to the newest. On the other hand, this might give the baseline enough data to capture the individual user behaviour better than by only using the most recent sessions, which certainly could benefit the recommendation given consistent user behaviours. This will be referred to as "long-term fitted" baseline.

5.5 Intra-session recommendation baselines

The intra-session recommendation was compared with two different RNN-based models. No other baselines were considered on the ground that both baselines had been shown to outperform other types of models in other work, also on the same datasets evaluated in this thesis.

5.5.1 RNN

The simplest baseline is referred to as *RNN*, and is the best performing baseline considered in Ruocco et al. [2017]. It is an edited adaption of the model in Hidasi et al. [2015] and its main parts are an item embedding layer followed by a GRU layer and ending in a final feed-forward layer.

[2]Steve Morse's Github repo: https://github.com/stmorse/hawkes

5.5.2 HRNN

HRNN is an abbreviation for "Hierarchical Recurrent Neural Network" and is the model proposed in Ruocco et al. [2017]. It shared many structural similarities with our model, as our model was built upon it. It was shown to significantly improve upon *RNN* in the work done in Ruocco et al. [2017] and is therefore considered a strong baseline.

5.5.3 CHRNN

CHRNN is an abbreviation for "Contextual Hierarchical Recurrent Neural Network" and is *HRNN* extended with the contexts added in the proposed model. Equivalently, it is the proposed model trained by only considering intra-session recommendation loss. It is used extensively in the discussion on whether or not the intra-session recommendation benefits from the joint training.

5.6 Syntetic time-gaps

In order to test the robustness of the method, some tests were conducted where the model was tested on synthetic time-gaps. Two different types of distributions were used for generating synthetic time-gaps. When applying the proposed model, the real time-gaps of the data were replaced with synthetic ones. While this does not preserve most dependencies between session content and time-gaps, both distributions are used in a way that preserves some connection with the data of the users they generate time-gaps for.Thus, there is some correlation between the synthetic time-gaps and the users they are assigned to. The goal of the test is therefore two-fold: test the models ability to fit other time-distributions and test how well the model can utilize user-modeling.

5.6.1 Hawkes distributed time-gaps

The same Hawkes-process parameterization that was used in the temporal baseline 5.4 was fitted on the time-gaps of each user in turn. This results in parameters based on the users' long-term behaviours, which then are used to generate a long continuous sequence of events that are used to get the synthetic time-gaps. Since the baseline is based on this type of Hawkes-process, it should perform very well on the synthetically generated data, making this test a very tough benchmark for the proposed model.

5.6.2 Normal distributed time-gaps

The second type of distribution used to create synthetic data are gaussian distributions where the standard deviation are set to be half a day and the mean are set to each user's mean time-gap. This way the different users will have different distributions but the same variance. Because the normal distribution is symmetrical, the error of a good model should converge towards a value close to the standard deviation as the best strategy is to always predict the user's mean time-gap.

Chapter 6

Result and Discussion

In this chapter all the results will be presented and then discussed. In order to do so in an ordered fashion, the evaluation metrics used will first be described and argued for. This is then followed by the detailed description of the most relevant results and a thorough discussion of these.

6.1 Evaluation Metrics

In order evaluate the results of the model, a few different evaluation metrics were applied. Because the model both provides recommendations and predict the time, it is difficult to identify an evaluation metrics that tells a lot about the performance in both tasks, or the combined performance. In the end a total of three different metrics were used.

6.1.1 Recall@k

Recall@k is a metric often used in recommendation and is evaluated by checking if the target recommendation is among the top k scored items. k is typically set in the range 5 to 20. The returned score is the number of times the target recommendation was within top k, divided by the total number of recommendations. Because the sessions can be of different lengths, recall@k is first calculated and averaged over for each position in the maximum session length. So we have one score for items that are third in their respective sessions and so forth. The aggregated values are then combined cumulatively step-wise such that the third score is the combined score of items that are in the three first positions. This also means that the final score is the cumulative score over all items. k-values of 5, 10 and 20 are used.

6.1.2 MRR@k

Mean Reciprocal Rank is also useful in recommendation evaluation. It is calculated by taking the mean of the inverse ranks for each recommendation. The rank is the position of the target item in the sorted scores. So if the target is ranked second, its Reciprocal Rank is $1/2 = \frac{1}{2}$. Like for recall, the k parameter denotes how many of the top scores we consider. If the target item is not in top k, its Reciprocal Rank is 0. This metric provides a measure of the performance, beyond simply checking if the target is among the top k, by weighting better ranks higher. An identical scheme for scoring individual individual positions like the one described in 6.1.1 is applied here as well. k-values of 5, 10 and 20 are used.

6.1.3 MAE

Mean Absolute Error is used for the time prediction. A global score here could be quite miss-leading. An absolute error of 4 days could be quite reasonable for a target of 40 days, but is most likely not good if the target was 1 day. This in mind, the targets are sorted into different buckets based on their magnitude, and MAE scores are calculated for each of these individually. The two smallest bucket intervals contains time-gaps in the range 0-2 hours and 2-12 hours respectivly. The The following buckets are 1 day wide, e.g. 2.5-3.5 days, all the way up to the 11.5-12.5 days interval, after which they are increased to be 2 days wide. MAE was chosen over RMSE(Root Mean Squared Error) mostly since it is easier to interpret. A MAE of 2 days means that the error is on average 2 days, a RMSE of 2 days is difficult to say anything about since this measure penalizes greater errors more than smaller ones. For the most part, we consider being off by e.g. 2 days to be roughly twice as bad as being 1 day off, so there is no real reason for penalizing greater errors more than smaller ones. Furthermore, testing was not the bottleneck of the model, so even if the absolute function is computational heavy, it did not affect the total training time of the model in any significant way.

6.2 Results and discussion introduction

In the following sections, all results of the different experiments will be presented. For the most part this will be in the form of tables and plots, but all additional information that is not covered in the experimental setup 5 will be noted in the text. In the results our proposed model will be denoted *THRNN* for Temporal *HRNN*. All presented results are the average value of using at least 5 different random seeds, results from fewer seeds might be mentioned in the text but then it will be explicitly stated that these are preliminary. For structuring purposes, it

was decided to combine the results and discussion, such that the results are first presented and then discussed immediately afterwards. The alternative is to have a pure results part and then backtrack in the discussion part, which would most likely make it difficult for readers to retain overview of the different experiments.

6.3 Main setup

This subsection contains the main results gotten from the default setup of the model. These are time prediction results, intra-session recommendation and initial recommendation. We will also take a closer look at how the joint training affects the intra-session recommendation.

6.3.1 Time results

LastFM

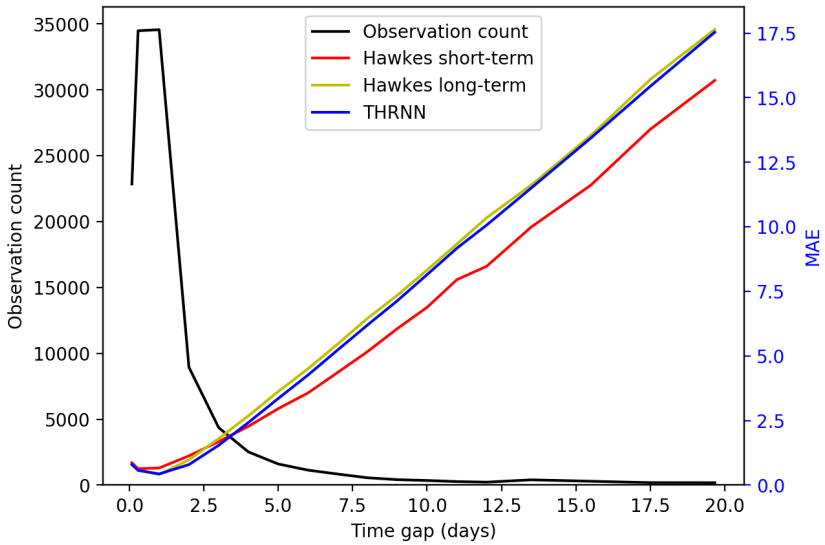

Figure 6.1: Full plot of the time prediction results along with observation counts for time-gaps of different durations. LastFM

Figure 6.1 is the full plot of the time prediction MAE on the LastFM dataset. As shown by the observation count plot in the same figure, most time-gaps in

this dataset are less than 4-5 days. For gaps greater than this, it is very clear that the baseline with short-term fitting outperforms the other two significantly. The relative difference between *THRNN* and the best baseline is the most at the interval 7.5-8.5 days where it is 0.97 days which is a difference of 19.9%. Towards the end, the MAE of all models appear to increase at the same linear rate.

Figure 6.2 focuses on time-gaps that are less than 5 days in order to better

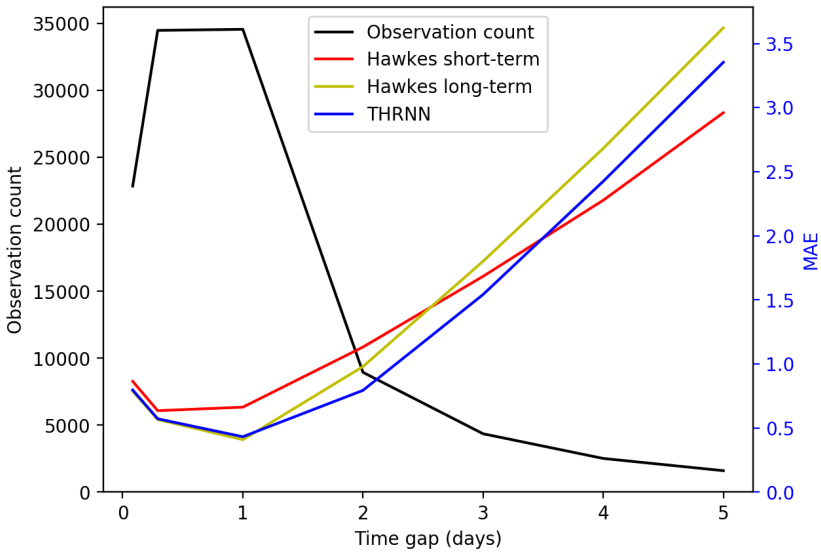

Figure 6.2: Focused plot of the time prediction results. LastFM

visualize the differences for the most frequent observations. One can observe that *THRNN* is quite a bit better than the short-term fitted baseline for time-gaps less than 3.5 days. It is also mostly better than the long-term fitted baseline overall, with the exception of the 0.5-1.5 days interval, where the long-term fitted baseline is ever so slightly better. For the 0.5-1.5 interval, *THRNN* has a MAE of 10.32 hours while the MAE of the short-term fitted Hawkes-process is 15.84 hours. This is a difference of 5.52 hours or a 34.8% improvement over the baseline. The difference is even greater for the 1.5-2.5 days interval, at 8.16 hours. For this interval, the performance of the long-term fitted baseline is also considerably worse than that of *THRNN*.

Even though the short-term baseline is thoroughly outperforming *THRNN* for most time-gap durations, *THRNN* is considerably better on time-gaps that are more frequent in the dataset. Given their frequency, it makes a lot of sense to optimize for them as well. We can also observe that after a certain time, the performance decrease similarly for all models, which is typically observed in prediction data for predictions significantly far into the future. The long-term fitted Hawkes-process is, for the most part, outperformed by *THRNN* in intervals with many observations, but it is better than the short-term fitted Hawkes-process on shorter time-gaps. This is not all that surprising since long-term optimization usually make the model excel at the most common observations. The short-term fitted Hawkes-process outperforms the other models for longer and less common time-gap durations, which is likely a product of it being more consistent in its use of short-term behaviour. *THRNN* is capable of considering short-term behaviour, but it may end up disregarding recent trends in favour of long-term trend, if the more recent behaviour is sufficiently uncommon. The short-term fitted baseline only observe the most recent data, so for periods of great inactivity, it will usually predict the continuation of similar inactivity.

The time-modeling is very stable, resulting in very small differences between different runs. The largest standard deviations are in the scale 0.02 days, which is approximately 30 minutes. This means that any error bars will usually be less wide than the default line thickness of the plots. Hence, error-plots will not be provided for any time-prediction results presented in this chapter.

Reddit

Figure 6.3 is the full plot of the time prediction MAE on the Reddit dataset. Like in the LastFM dataset, shorter time-gap durations are more frequent, but the observation count fall-off, when moving in the direction of longer time-gaps, is significantly less for this dataset. No intervals have less than 1500 observations in the Reddit dataset, while half of the intervals in the LastFM dataset have less than 1000 observations each. *THRNN* outperforms the other models on time-gap durations between 1 and 10 days, but for all other time-gap intervals, the short-term fitted Hawkes-process is the best model. The long-term fitted baseline starts off worst, and is never really excelling at any time-gap intervals.

Figure 6.4 is a focused plot on the time prediction results for targets that are 10 days or less, on the Reddit dataset. One can observe that no model have a MAE of less than 1.32 days for any interval. This means that all models are significantly off for the smaller time-gaps, where the errors are on average much greater than the time-gaps themselves. However, every model exhibits a slow increase in MAE for the smaller time-gap intervals. Especially *THRNN*, for which

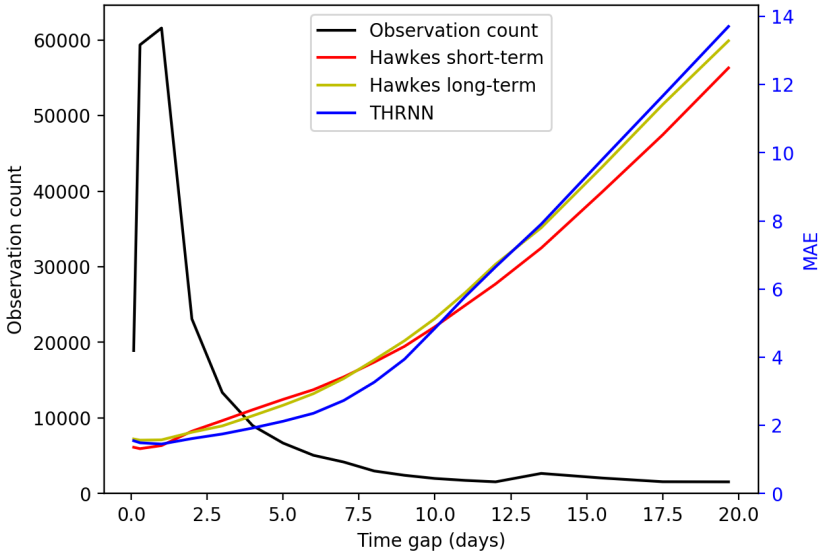

Figure 6.3: Full plot of the time prediction results along with observation counts for time-gaps of different durations. Reddit

the the MAE only increases by 0.91 days from the 0.5-1.5 days interval to the 5.5-6.5 days interval. So the prediction targets are on average 5 days greater, but the MAE is less than a day greater. For the 4.5-5.5 days interval, THRRN achieve a MAE that is 15.4 hours less than that of the short-term baseline, which is a difference of 23.2% relative to the score of the short-term fitted Hawkes-process.

It is apparent that the long-term fitted baseline struggles with this dataset. It achieves slightly better results than the short-term fitted baseline for time-gaps in the range 2-7 days, but is in the same interval significantly outperformed by *THRNN*. For the greatest time-gaps it achieves slightly better results than *THRNN*, but as both MAE's are mostly greater than 9 days for these intervals, it is difficult to identify a scenario where this difference will matter. The short-term fitted baseline is for the most part better than *THRNN* for time-gaps of 1 day or less. However, neither can be claimed to be accurate for the smallest time-gaps. Data analysis of the Reddit dataset reveals that there was an explosion in activity in the last year of the 9-10 recorded years of data. Only in the last 6 month, one can observe a 4-fold increase in number of sessions. It would not be

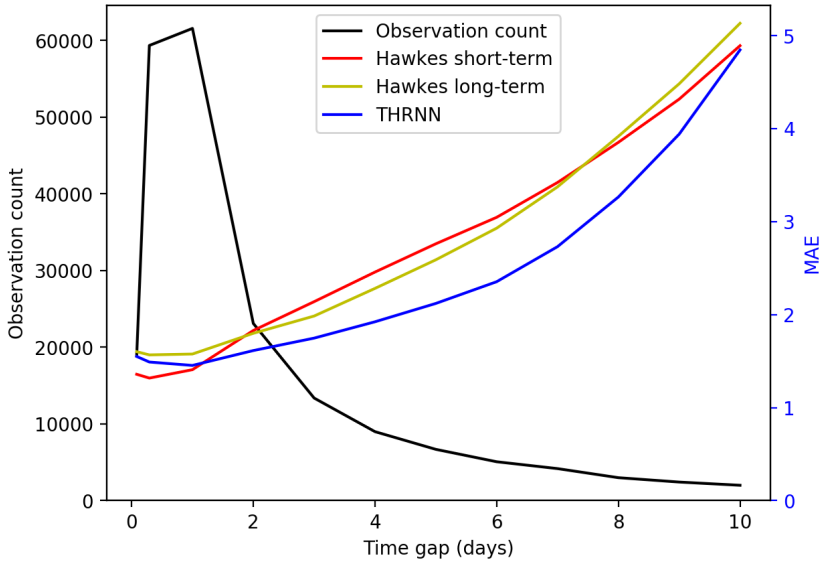

Figure 6.4: Focused plot of the time prediction results. Reddit

surprising if the large increase in activity will cause, directly or indirectly, changes in the overall user dynamics. While *THRNN* and the long-term fitted baseline both will train/fit on user data from when the service was much less active, the short-term fitted baseline use the most recent user data available. We can see that the short-term fitted baseline is strongly outperforming *THRNN* for the smallest time-gaps, but is then strongly outperformed for many middle-long time-gap intervals. Interesting enough, further analysis shows that the time-gap distribution is near identical in testing and training data, which does not necessarily mean that the dynamics is entirely the same, but at least that the users have more or less the same average time-gap durations in both training and testing data. So it is not entirely apparent why the short-term fitted baseline is so dominating for the smallest time-gaps. The long-term considering models might struggle more with this dataset because the time-gap distribution is not as centered on the smallest time-gaps as what can be observed in the LastFM dataset. Since there are non-negligible numbers of observations for even the largest time-gaps, these can affect the modeling by shifting the focus towards larger time-gaps. This is the leading theory, and is also one of the key motivations behind the introduction of the κ tuning parameter, which is tested in 6.5.

6.3.2 Intra-session recommendation results

LastFM

	R@5	R@10	R@20	MRR@5	MRR@10	MRR@20
HRNN	0.1415 ± 0.0005	0.1993 ± 0.0007	0.2751 ± 0.0006	0.0876 ± 0.0004	0.0952 ± 0.0004	0.1004 ± 0.0004
THRNN	0.1437 ± 0.0004 (+1.6%)	0.2026 ± 0.0006 (+1.7%)	0.2795 ± 0.0006 (+1.6%)	0.0889 ± 0.0002 (+1.6%)	0.0967 ± 0.0003 (+1.6%)	0.102 ± 0.0003 (+1.6%)

Table 6.1: Table with intra-session recommendation results compared with *HRNN*. LastFM

Table 6.1 contains the aggregated intra-session recommendation results on the LastFM dataset. Aggregated in the sense that it includes scores from all positions in the sessions and averaged over the total number of scores as described in 6.1.1. As shown by the relative differences, *THRNN* achieves a statistically significant improvement of 1.6% over the baseline in all measures.

While the improvements cannot be claimed to be revolutionary, they are not insignificant either. They indicate that the extended model does manage to improve the session modeling somewhat, and provide a better initial hidden state to the intra-session RNN than the one provided in the baseline. Since the difference between the model is not just the joint training, but also some added contexts, these results cannot conclude that the model benefits from the joint modeling. It is possible that the improvements origin in the added contexts alone. Thus, we test this claim by comparing with a version of the model that apply the contexts but does not predict time nor recommend an initial item in section 6.3.4.

Figure 6.2 contains the same results as the previous table but relative scores are

	R@5	R@10	R@20	MRR@5	MRR@10	MRR@20
RNN	0.1349 ± 0.0004	0.184 ± 0.0002	0.2474 ± 0.0002	0.086 ± 0.0003	0.0925 ± 0.0002	0.0969 ± 0.0002
HRNN	0.1415 ± 0.0005 (+4.9%)	0.1993 ± 0.0007 (+8.3%)	0.2751 ± 0.0006 (+11.2%)	0.0876 ± 0.0004 (+1.8%)	0.0952 ± 0.0004 (+2.9%)	0.1004 ± 0.0004 (+3.7%)
THRNN	0.1437 ± 0.0004 (+6.6%)	0.2026 ± 0.0006 (+10.1%)	0.2795 ± 0.0006 (+13.0%)	0.0889 ± 0.0002 (+3.4%)	0.0967 ± 0.0003 (+4.5%)	0.102 ± 0.0003 (+5.3%)

Table 6.2: Table with intra-session recommendation results compared with *RNN*. LastFM

now from comparing with the results from using *RNN*.

Reddit

	R@5	R@10	R@20	MRR@5	MRR@10	MRR@20
HRNN	0.4432 ± 0.0013	0.5316 ± 0.0009	0.616 ± 0.0012	0.317 ± 0.0016	0.3288 ± 0.0016	0.3347 ± 0.0015
THRNN	0.4468 ± 0.0013 (+0.8%)	0.5366 ± 0.001 (+1.0%)	0.6228 ± 0.0009 (+1.1%)	0.3191 ± 0.0015 (+0.7%)	0.3311 ± 0.0014 (+0.7%)	0.3371 ± 0.0014 (+0.7%)

Table 6.3: Table with intra-session recommendation results compared with *HRNN*. Reddit

Table 6.3 contains the aggregated intra-session recommendation results on the Reddit dataset. The same aggregation was applied here as on the LastFM dataset, so the scores are weighted over all outputted data, regardless of the position they appeared in in the sessions. We can also observe improvements over all measures for this dataset, but they are not in the same scale as those observed on the LastFM dataset. There is also a difference between the improvements on Recall measures and MRR scores. Because MRR does not scale linearly, it is difficult to say anything about how much the improvements in the Recall would look in the MRR score. x% increase in Recall@k indicates that the model successfully recalls x% more recommendations, but the same improvements in MRR@k can be caused by better Recall, better ranking, positive fluctuations in rankings or a combination of these. In any case, the % improvement over the baseline is less for the MRR score than for the Recall score. Ruocco et al. [2017] (*HRNN*) observed a greater improvement in MRR scores than in Recall scores on the Reddit dataset, but the opposite was observed on the LastFM dataset. So it is not unlikely that our observation is simply a product of the non-linear nature of MRR in relation to Recall.

Figure 6.4 contains the same results as the previous table, but relative scores

	R@5	R@10	R@20	MRR@5	MRR@10	MRR@20
RNN	0.3208 ± 0.0004	0.3959 ± 0.0005	0.475 ± 0.0003	0.2346 ± 0.0006	0.2445 ± 0.0006	0.25 ± 0.0006
HRNN	0.4432 ± 0.0013 (+38.1%)	0.5316 ± 0.0009 (+34.3%)	0.616 ± 0.0012 (+29.7%)	0.317 ± 0.0016 (+35.1%)	0.3288 ± 0.0016 (+34.5%)	0.3347 ± 0.0015 (+33.9%)
THRNN	0.4468 ± 0.0013 (+39.3%)	0.5366 ± 0.001 (+35.6%)	0.6228 ± 0.0009 (+31.1%)	0.3191 ± 0.0015 (+36.0%)	0.3311 ± 0.0014 (+35.4%)	0.3371 ± 0.0014 (+34.8%)

Table 6.4: Table with intra-session recommendation results compared with *RNN*. Reddit

are now from comparing with the results of *RNN*.

Step	R@5	R@10	R@20	MRR@5	MRR@10	MRR@20
initial	0.1275 ± 0.0007	0.1821 ± 0.0011	0.2521 ± 0.0013	0.0776 ± 0.0006	0.0848 ± 0.0006	0.0896 ± 0.0006
2	0.1296 ± 0.0006	0.1836 ± 0.0011	0.2552 ± 0.0011	0.0797 ± 0.0004	0.0868 ± 0.0004	0.0917 ± 0.0004
3	0.1493 ± 0.0005	0.2067 ± 0.0006	0.2808 ± 0.0012	0.0948 ± 0.0005	0.1024 ± 0.0005	0.1075 ± 0.0005

Table 6.5: Table with Recall and MRR scores from the LastFM dataset. The initial recommendation is presented with the intra-sessions recommendation results evaluated on the two first selections.

6.3.3 Initial recommendation results

LastFM

Table 6.5 contains the initial recommendation results on the LastFM dataset, as well as the individual scores on early items in the intra-session recommendation. Overall, the results are slightly worse than the first intra-session recommendation, i.e. when the initial selection of the new session has been provided to the model to be used in the recommendation of the "second" item. This indicates that the model is able to use the inter-session modeling to return a decent recommendation of the initial item, but that when the first item of the new session is provided as well, the recommendation is improved beyond this. From analysis of both dataset, it has been noted that the first and second items should be similarly difficult to predict using a simple frequency-based recommendation model. Furthermore, observations show that in general, the model is able to provide better recommendations longer into sessions, which indicates strong intra-session relations. Thus, it is not surprising that the model is able to provide better recommendation for the "second" item, since it at this point can apply both inter- and intra-session modeling.

Reddit

Step	R@5	R@10	R@20	MRR@5	MRR@10	MRR@20
initial	0.4564 ± 0.0012	0.5415 ± 0.0014	0.6218 ± 0.0014	0.3108 ± 0.0006	0.3222 ± 0.0006	0.3278 ± 0.0006
2	0.4144 ± 0.0011	0.5031 ± 0.001	0.5896 ± 0.001	0.2884 ± 0.0015	0.3003 ± 0.0014	0.3063 ± 0.0014
3	0.482 ± 0.0017	0.5677 ± 0.0013	0.6501 ± 0.0012	0.3591 ± 0.0016	0.3706 ± 0.0015	0.3763 ± 0.0015

Table 6.6: Table with Recall and MRR scores from the Reddit dataset. The initial recommendation is presented with the with the intra-sessions recommendation results evaluated on the two first selections.

Table 6.6 contains the initial recommendation results from the Reddit dataset and some scores of the early selections in the intra-session recommendation. In contrast to the results from the LastFM dataset, the performance on the initial item is actually better than that of the first intra-session recommendation. This might mean that the inter-session modeling can say more about the next sessions

in this dataset, which is supported by the observation that *HRNN*'s improvement over *RNN* is much greater on the Reddit dataset than on the LastFM dataset. However, it also means that even though the second item should have both good inter-session information and intra-session information, it is outperformed by a recommendation that is purely based on inter-session modeling. First and foremost, the initial recommendation train a single set of weights for the sole purpose of recommending the initial item, while when the hidden state is passed to the intra-session RNN, all the weights affecting its propagation is also trained for the more general intra-session recommendation. Secondly, the initial hidden state is the only propagated hidden state which is not propagated from the intra-session RNN itself, so its structure and form might not be equally optimized for the weights as those that do. Thus, even if the inter-session information is of high quality, part of this may be lost or disregarded during propagation. This may indicate that there is a potential for improvement by looking into how the last hidden state of the inter-session RNN is propagated to the intra-session RNN. When looking at the third item, we can see that these scores are considerably better than the initial recommendation, so it is clear that at this point, the combined forces of intra- and inter-session modeling outperforms inter-session modeling alone.

6.3.4 Effect of training joint model

Du et al. [2016] and Jing and Smola [2017] both observed an improved recommendation when training their temporal models with time prediction loss. In 6.3.2 we show that our model outperforms the baseline in recommendation, but our model has also extended the baseline with contexts in addition to the temporal aspects. So this section aims to properly test whether or not the intra-session recommendation of our model benefits from jointly training on the three different tasks.

LastFM

	R@5	**R@10**	**R@20**	**MRR@5**	**MRR@10**	**MRR@20**
CHRNN	0.1443 ± 0.0003	0.2038 ± 0.0007	0.2817 ± 0.0005	0.0889 ± 0.0001	0.0967 ± 0.0002	0.1021 ± 0.0002
THRNN	0.1437 ± 0.0004 (-0.4%)	0.2026 ± 0.0006 (-0.6%)	0.2795 ± 0.0006 (-0.8%)	0.0889 ± 0.0002 (+0.1%)	0.0967 ± 0.0003 (-0.0%)	0.102 ± 0.0003 (-0.1%)

Table 6.7: Table with Recall and MRR scores from the LastFM dataset. *CHRNN* is *HRNN* extended with the same contextual information used in *THRNN*.

Table 6.7 compares the intra-session results of *THRNN* with and without considering time prediction and initial recommendation loss during training, on the LastFM dataset. The difference in the MRR scores are statistically insignificant.

For the Recalls, one can actually observe a small statistical significant difference in favour of the single task model, which indicate that the joint training is slightly deteriorating the models intra-session recommendation performance.

Reddit

	R@5	R@10	R@20	MRR@5	MRR@10	MRR@20
CHRNN	0.4471 ± 0.0009	0.5365 ± 0.0009	0.622 ± 0.0007	0.3197 ± 0.0012	0.3316 ± 0.0011	0.3376 ± 0.0011
THRNN	0.4468 ± 0.0013 (-0.1%)	0.5366 ± 0.001 (+0.0%)	0.6228 ± 0.0009 (+0.1%)	0.3191 ± 0.0015 (-0.2%)	0.3311 ± 0.0014 (-0.2%)	0.3371 ± 0.0014 (-0.2%)

Table 6.8: Table with Recall and MRR scores from the Reddit dataset. *CHRNN* is *HRNN* extended with the same contextual information used in *THRNN*.

Table 6.8 compares the intra-session results of *THRNN* with and without considering time prediction and initial recommendation loss during training, on the Reddit dataset. As can be seen from the standard deviations, all differences are covered by a single standard deviation. So the results does not support any statistical difference between training with or without time prediction loss on the Reddit dataset.

Effect of joint training discussion

Du et al. [2016] and Jing and Smola [2017] both observed improved recommendation when including time modeling. However, this can not easily be compared with our results as they both used single-level RNNs for making a single recommendation with an adhering time-prediction. While Jing and Smola [2017] do model inter-session time-gaps, like our model, the recommendation and time prediction are more tightly connected in their model. Since our model is hierarchical, the only way time prediction can improve the intra-session recommendation is by allowing for better initial hidden states to be propagated to the intra-session RNN. Both Ruocco et al. [2017] and Quadrana et al. [2017] showed that by going from a default initial hidden state to one provided by a hierarchical RNN level, based on previous user history, it is possile to improve recommendation significantly. However, it was also shown that the vast majority of the improvements could be observed in the first few recommendations of the sessions. Thus, decreasing the difference in performance between when the model has seen few selections in the session and when it has seen many. This could indicate that there is a practical limit to how much simpler RNN-based recommendation can be improved by better initial hidden states alone. Furthermore, since the hidden state is used in the two other tasks of our model as well, there might arise conflicts from the three different gradient contributions. Thus, even if the time prediction training allows for better general time-modeling, this might come at

the cost of inter-session modeling trained by the intra-session recommendation. The overall effect may therefore be equal or worse than not considering the time prediction loss at all.

The initial recommendation loss comes from the same type of loss function that is used for intra-session recommendation, but is both weighted less and has a single contribution for each training case as opposed to the session length. It is therefore not likely that it affects the model very much, with the exception of its exclusive linear layer. It is therefore largely ignored in this discussion.

It is not very straight-forward to devise a plan for testing with certainty whether or not the recommendation is considering any time-modeling. If it does in fact do so, but the improvements are hidden through worse general recommendation modeling, it might be better for the joint model to use higher dimensionality tensors in the inter-session RNN. The intuition being that more weights can more easily be distributed between conflicting "opinions". However, by doing so, observed changes might stem from the increased dimensionality and not the improved time modeling. Considering this, we could compare both joint- and pure recommendation setups using two different dimensionalities. If we only want to increase the dimensionality in the inter-session RNN, we need to introduce a linear layer for reducing the dimensionality of the initial hidden-state to the intra-session RNN, which is new potential source of noise. The best option is probably to increase the dimensionality of both the inter- and intra-session RNN, if the greater dimensionality shows improvements when the joint model is used, it should be an indication that the time-modeling can in fact help the recommendation, also in a hierarchical model.

Tables 6.9 and 6.10 contains the intra-session recommendation results on both

	R@5	**R@10**	**R@20**	**MRR@5**	**MRR@10**	**MRR@20**
CHRNN*	0.1458 ± 0.0005	0.2054 ± 0.0006	0.2835 ± 0.0004	0.0898 ± 0.0001	0.0977 ± 0.0001	0.103 ± 0.0001
THRNN*	0.145 ± 0.0 (-0.6%)	0.2042 ± 0.0002 (-0.6%)	0.2817 ± 0.0006 (-0.7%)	0.0898 ± 0.0002 (-0.0%)	0.0976 ± 0.0002 (-0.1%)	0.1029 ± 0.0002 (-0.1%)

Table 6.9: Table with Recall and MRR scores from the LastFM dataset. * is added to denote that the dimensionality of the hidden state has been increased in these models. Increased by 20 from 100 to 120.

datasets when the dimensionality of the hidden states have been increased by 20. Results from LastFM are near identical to those from using the regular dimensionalities. The new Reddit results show some improvement in the MRR scores, however, the results are far from being statistically significant. Thus, the conclusion remains that it is not fully clear whether or not the model can achieve

	R@5	R@10	R@20	MRR@5	MRR@10	MRR@20
CHRNN*	0.4632 ± 0.0006	0.5523 ± 0.0003	0.6365 ± 0.0007	0.3318 ± 0.0017	0.3437 ± 0.0016	0.3496 ± 0.0015
THRNN*	0.4631 ± 0.0005 (-0.0%)	0.5525 ± 0.0004 (+0.0%)	0.6366 ± 0.0004 (+0.0%)	0.3327 ± 0.0011 (+0.3%)	0.3447 ± 0.001 (+0.3%)	0.3505 ± 0.001 (+0.3%)

Table 6.10: Table with Recall and MRR scores from the LastFM dataset. * is added to denote that the dimensionality of the hidden state has been increased in these models. Increased by 20 from 50 to 70.

improved recommendation by training jointly on all three tasks.

It might be possible to achieve better results by devising a different training scheme for training the different tasks and the joint model. Some experiments was performed where the model was first trained on different tasks in turn, then in pairs and finally all three. None of the setups we tried performed better than jointly training on all tasks throughout the training, however, this does not fully exclude the possibility that a more complex scheme might work.

6.4 Synthetic time-gaps with real sessions

In order to test the robustness of the model, we created some datasets where the time-gaps between the sessions are switched out with synthetically generated ones. By doing so, the relation between the content of the sessions and the inter-session time-gaps disappears/is altered, but it will both test the models capability of fitting different parameterizations and at the same time perform the tasks of recommendation. All synthetic time-gaps are in some way related to the user they were generated for, so the content of the sessions and the synthetic time-gaps are not completely independent. This will provide some long-term user behaviour, allows the experiments to also test the models personalization capabilities.

6.4.1 Normal distributed time-gaps

LastFM

Figure 6.5 is the time prediction plot of the LastFM dataset with normal distributed time-gaps. The reason all models show a dip in MAE for some of the smaller time-gaps is because the time-gaps cannot be smaller than the smallest allowed time-gap, which for the LastFM dataset is half an hour. Thus, all drawn time-gaps that were less than half an hour, was set to half an hour. This effectively reduces the variance for users with a mean close to this value, which one can see from the observation count plot, is most users. We can see that all models

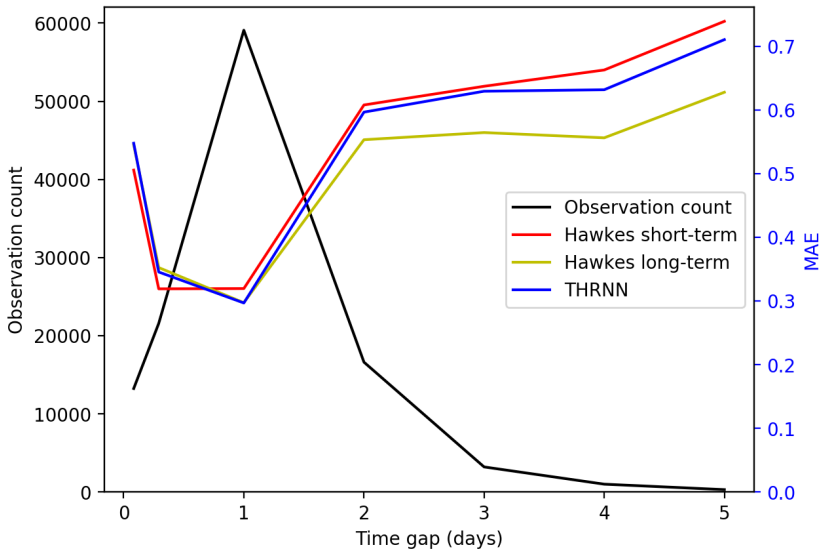

Figure 6.5: Time prediction results on the LastFM dataset using synthetically generated time-gaps drawn from normal distributions with mean equal to the users' average time-gaps, and standard deviations of 0.5 days.

are pretty close, but that the long-term fitted baseline outperforms the other two later on, while the short-term baseline trails behind.

Since the distributions was set for each user based on their personal time-gap mean, it is no surprise that the long-term fitted baseline is the best at accurately identifying this mean. That *THRNN* performs better than the short-term fitted baseline supports the notion that it is capable of considering both long- and short-term dynamics better than the short-term Hawkes-process. After the initial intervals with smaller variance, all models have an MAE that is slightly greater than the standard deviation. This makes it likely that they have all adapted strategies that essentially is to consistently predicting an estimated mean.

Reddit

Figure 6.6 is the time prediction plot of the Reddit dataset with normal distributed time-gaps. For both baselines we observe the same dip in MAE for the

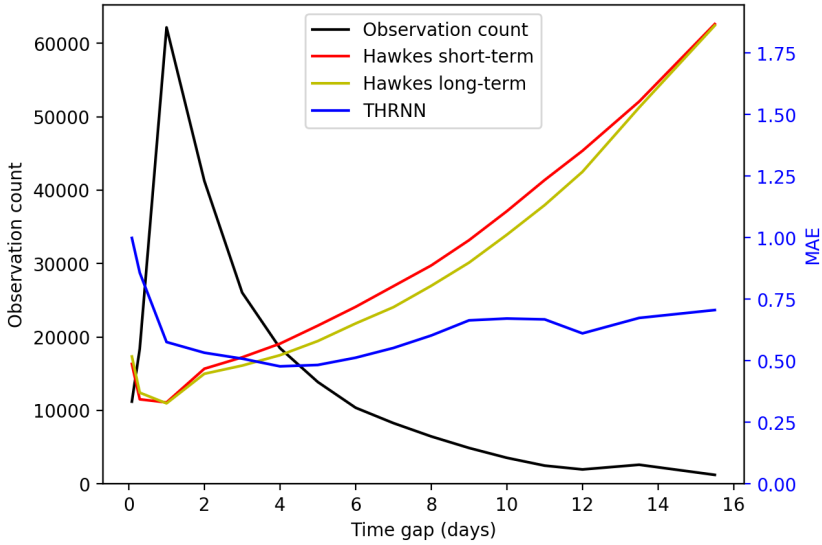

Figure 6.6: Time prediction results on the Reddit dataset using synthetically generated time-gaps drawn from normal distributions with mean equal to the users' average time-gaps, and standard deviations of 0.5 days.

smallest time gaps, however *THRNN* starts off with a much greater MAE. We can also observe that there clearly are a larger fraction of the Reddit users with higher mean time-gaps than those observed in the LastFM dataset. For longer time-gaps we can see the MAE of both baselines starting to diverge from the standard deviation. This indicate that both baselines struggle with many of the users with greater time-gap means.

The performance of the long-term baseline only affected by the training data, while the short-term baseline is more concerned with the testing data(as long as the training data is much larger than the length of the observed history). The fact that neither of these are able to capture the user means could indicate that many of the relevant users have very little data, making it difficult to fit the processes. This is also supported through analysis of the data. The fact that *THRNN* still manages to perform well for these may indicate that it is able to apply knowledge learned from similar users to make up for the lack of data for the less active users. What is peculiar about this is that it appears to be able

to infer a lot from users with little data, but struggles with many of the users with shorter time-gap means, despite the fact that the shorter time-gaps appear much more frequently. An explanation for this may be the skewed relationship between number of user with certain means, and the number of sessions the users have, combined with the usage of user embeddings. Perhaps numerous low activity users, with longer time-gap means, "hijacks" user embeddings through their sheer number. This could end up leaving little room for the fewer, but more active users. Since the number of users in the Reddit datset is very large, it becomes improbably to give each user a highly distinct embedding, thus the model might learn broader "general" users instead. If this is the case, the model might be tuned for having general users who all behave pretty similarly, and therefore struggles with the few "specialized" users.

6.4.2 Hawkes distributed time-gaps

LastFM

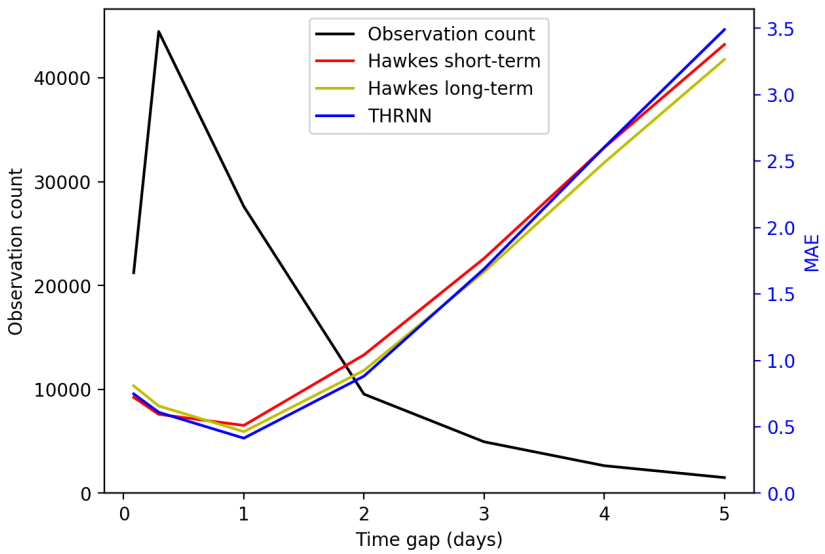

Figure 6.7: Time prediction results on the LastFM dataset using synthetically generated gaps drawn from Hawkes-processes fitted on individual users' time-gaps.

Figure 6.7 is the time prediction plot of the LastFM dataset with Hawkes distributed time-gaps. All in all, there does not appear to be a great difference between the models. The short-term fitted baseline is best on the two smallest time-gap intervals, *THRNN* is best on the following two and the long-term fitted Hawkes is best for all the remaining greater time-gap intervals.

Since the time-gaps are drawn from a Hawkes process, both baselines are expected to perform well in this experiment. The long-term fitted baseline will most likely, on average, fit parameters that are closer to that of the generating process than the parameters fitted by the short-term fitted baseline. So it is not surprising that the long-term fitted baseline performs better for most time-gap intervals. The reason the short-term fitted baseline is better for the smallest time-gaps is most likely tied with the fact that the long-term fitted version gets to consider all the history, and while the shortest time-gaps are the most frequent ones, there is also a significant amount of larger ones also present in the training set. Due to the nature of the Hawkes process, we can get periods with very high intensity caused by many short time-gaps being drawn. While the long-term fitted baseline clearly is able to capture this somewhat, the short-term fitted baseline might be able to fit specialized Hawkes processes that are good at predicting in such intensive periods. The fitted parameters might not be very similar to those of the generating process, and would most likely perform bad when used on all testing data, as opposed to just the current prediction. *THRNN* is clearly able to keep up with the Hawkes-based baselines, thus proving that it is robust in the sense of being able to model different distributions. We also see that it is the best performing model for some of the time-gap intervals it performed well on in the real dataset. This can most likely be explained by it being able to use general user-behaviour learned from having observed all users, and since the loss function appears to optimize for certain time-gap durations depending on the distribution of observations(see 6.5).

Reddit

Figure 6.8 is the time prediction plot of the Reddit dataset with Hawkes distributed time-gaps. Many of the same observations and remarks done on the results from the same experiment, but on the LastFM dataset, can be applied on these results as well. Like for LastFM, one can observe that *THRNN* outperforms the other models on time-gap intervals it performed well on in the real time-gap dataset. The difference is greater than what it was in the LastFM dataset, but that was also the case for the datasets with real time-gaps. Looking at the observation count plots, we can see that the time-gap distributions of the synthetic and real time-gaps are comparable, with Reddit having a greater and more significant tail. As the generating Hawkes processes are fitted on the real

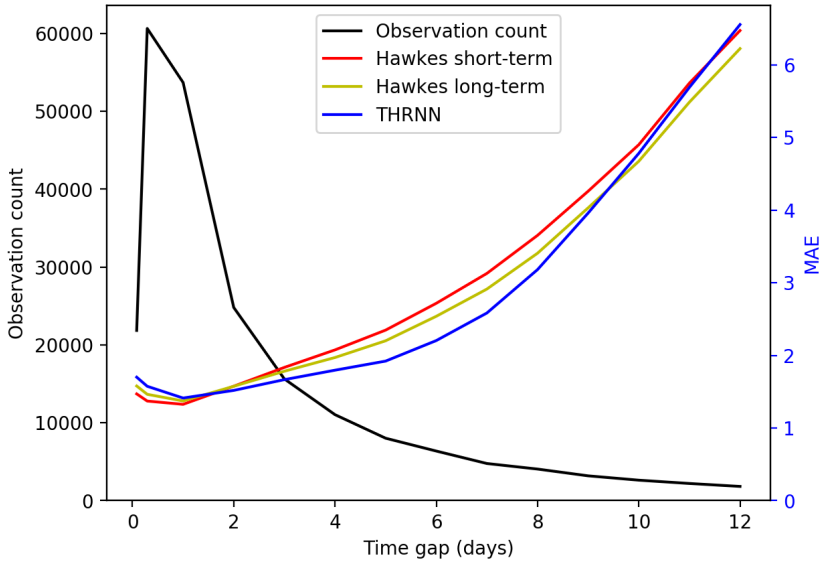

Figure 6.8: Time prediction results on the Reddit dataset using synthetically generated gaps drawn from Hawkes-processes fitted on individual users' time-gaps.

data, this is no surprise.

6.5 κ hyperparameter tests

In this section we will look at time-prediction results from different settings of the κ hyperparameter described in section 4.7.2.

LastFM

Figure 6.9 contains the time prediction results on the LastFM dataset for different initializations of the κ hyperparameter. All tested values are 1.0 or less which intuitively will make the model weight the larger time-gaps less relative to the smaller ones. This is fully supported by the observations represented in the plot. When κ is set close to 1.0, we get better performance on longer time-gaps but worse for smaller ones. The performance of all initializations seems to meet at 1.5 days time-gaps which is on the middle point between the 0.5-1.5 days interval

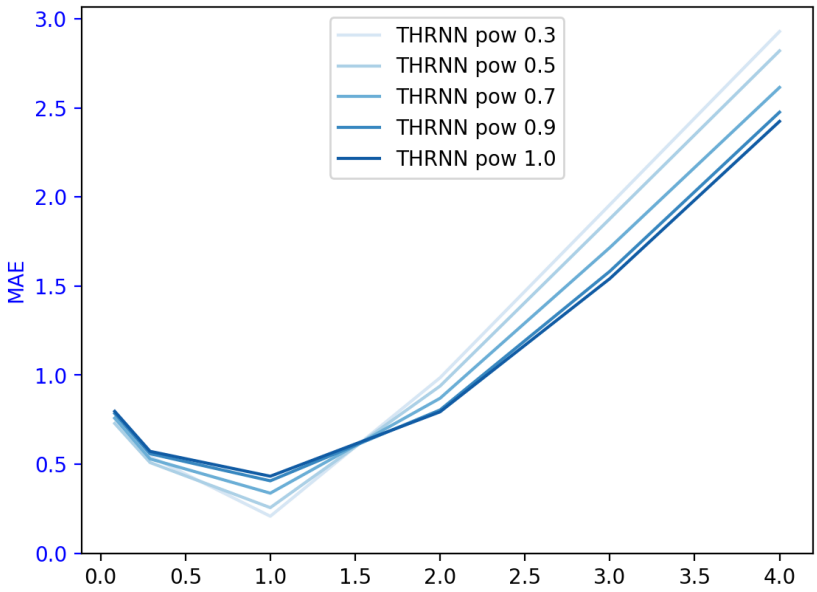

Figure 6.9: Comparing time prediction MAE from different values of the κ hyperparameter evaluated on the LastFM dataset. Runs from 0.1 are not included as time-specific gradients diverged for this initialization.

and the 1.5-2.5 days interval. For time-gaps greater than 3 days, the performance of all settings appear to deteriorate at the same rate. It is also worth noting that the plots looks more and more linear the smaller κ is set. For $\kappa = 0.3$ the plot looks very similar to one produced by predicting an averaged time-gap for every single prediction. Since the relative loss contribution of longer time-gaps gets smaller as κ is set smaller, it is not surprising for the model to end up returning predictions in a small interval centered close to the average time-gap length. The plot is a good illustration of the ensuing trade-off between prediction error for the most frequent time-gaps and the dynamic prediction range of the model.

Figure 6.10 compares the time prediction MAE of the model when $\kappa = 0.7$, with the Hawkes baselines.

By comparing with figure 6.2, we can see that by tuning κ, our model goes from being outperformed by the long-term fitted Hawkes process on some of the

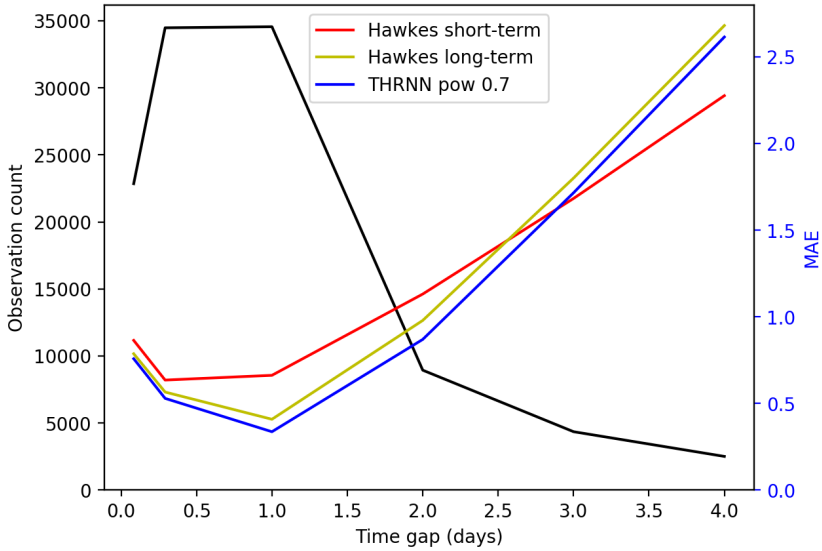

Figure 6.10: Time prediction MAE compared with Hawkes baselines when initalizing κ to 0.7. LastFM

smaller time-gaps, to significantly outperforming it for all time-gaps that are less than 4 days. This did come at the cost of being outperformed by the short-term fitted Hawkes for all time-gaps greater than 3 days as opposed to those greater than 3.5 days which was the case for $\kappa = 1.0$. Since 90% of the time-gaps are less than 3 days and only 2% of the time-gaps are between 3 and 3.5 days, the improved performance on smaller time-gaps might be well worth the decreased performance on infrequent time-gaps and the narrower dynamic prediction range.

Reddit

Figure 6.11 contains the time prediction results evaluated on the Reddit dataset for different initialization of the κ hyperparameter. This is quite different from the similar analysis on the LastFM dataset, as the MAE starts off very high for greater κ, but deteriorates very slowly. This indicates that the model has a much greater prediction range when trained on the Reddit dataset compared to when trained on the LastFM dataset. However, we see that by reducing κ slightly, we can shift the focus to smaller time-gap intervals. This can significantly improve the model's performance on these, while still keeping a greater prediction range

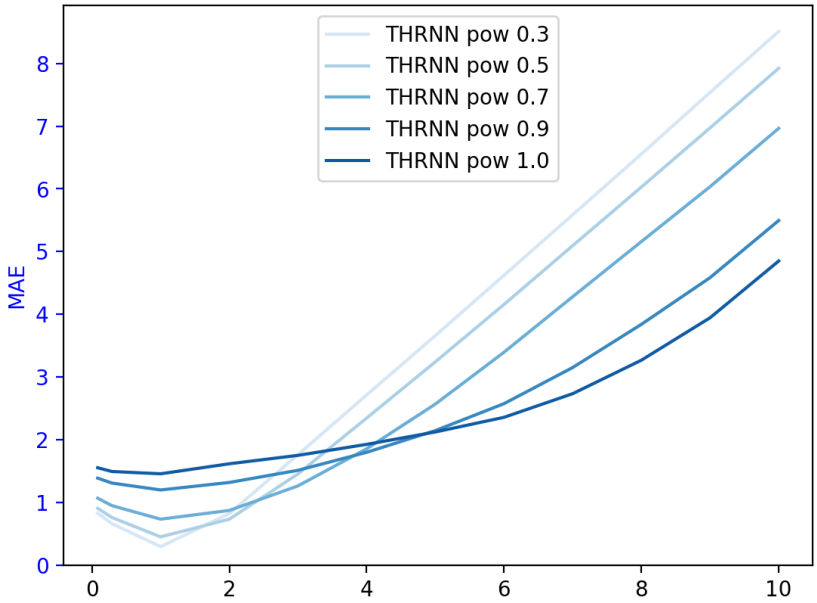

Figure 6.11: Comparing time prediction MAE from different values of the κ hyperparameter evaluated on the Reddit dataset. Runs from 0.1 are not included as time-specific gradients diverged for this initialization.

than what observed on the LastFM dataset. For $\kappa = 0.3$ we can observe a prediction pattern which is very similar to the most limited LastFM setups, so the predictive range is clearly deteriorating fast for small κ. One can also see that when going from κ of 0.9 to 0.7, there is a much greater deterioration of performance on long time gaps than those seen/indicated when looking at 1.0 to 0.9 and 0.7 to 0.5. Similarly, it can be observed that for smaller-middle time-gaps, going from κ of 0.9 to 0.7 improves the performance more than any other transitions. This could indicate that there is a good candidate κ somewhere between 0.7 and 0.9 which has similar performance of 0.7 on smaller-middle time-gaps but closer to 0.9 performance on longer time-gaps.

In figure 6.12 one can see the time prediction evaluated on the Reddit dataset, when $\kappa = 0.7$, compared with the Hawkes baselines.

The difference between figure 6.3 where $\kappa = 1.0$ and the new plot is striking.

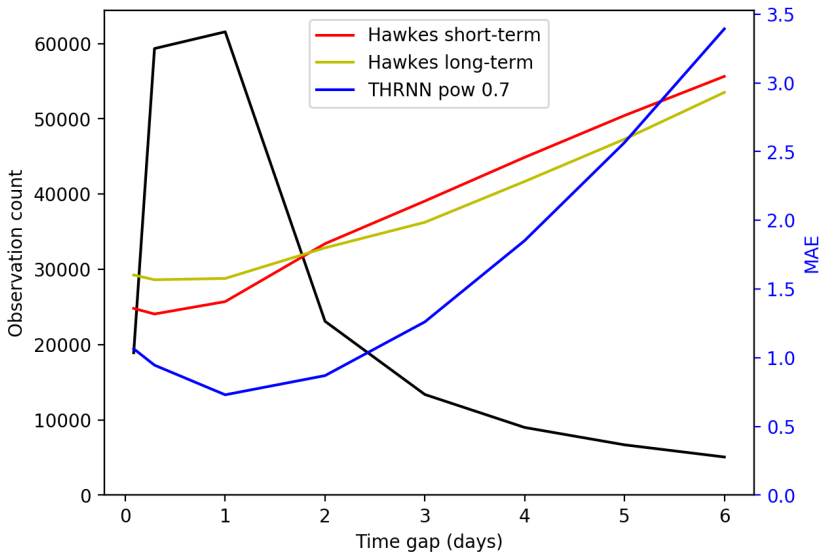

Figure 6.12: Time prediction MAE compared with Hawkes baselines when initializing κ to 0.7. Reddit

For $\kappa = 1.0$ *THRNN* was outperformed on the three smallest time-gap intervals by the short-term fitted Hawkes. When $\kappa = 0.7$, the MAE of the most frequent time-gap interval is approximately half of that of the best performing baseline. The difference in performance is highly significant for time-gaps that are less than 4 days, but decreases rapidly for larger time-gaps. *THRNN* is outperformed on time-gaps that are greater than 5 days by both baselines. For $\kappa = 1.0$, *THRNN* is outperformed on time-gaps greater than 10 days, meaning that the performance on time-gaps between 5 and 10 days has deteriorated significantly. However, as 85% of the time-gaps are less than 5 days, this might very well be a favorable trade-off in many scenarios. Like previously mentioned, the plots could indicate a "better" value of κ between 0.7 and 0.9 which balances short and long time-gaps well. That being said, what one defines as well is highly dependent on the domain/environment the model is used in.

6.5.1 Summarized plot

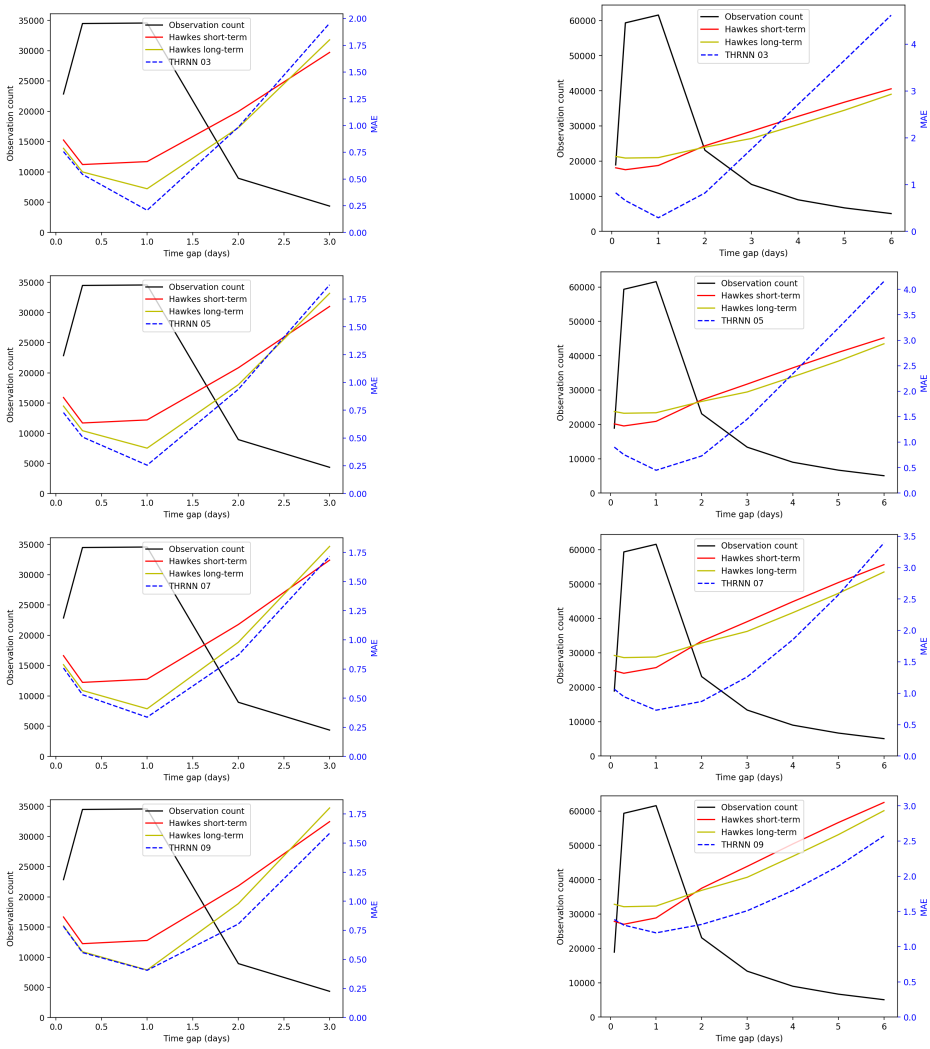

Figure 6.13: Left column: LastFM. Right column: Reddit. From top to bottom, κ settings: 0.3, 0.5, 0.7 and 0.9

Figure 6.13 contains plots of the different κ settings compared with the Hawkes baselines, on both datasets.

Chapter 7

Conclusion

Being able to model time in relation to recommendation is a highly relevant ability that have a lot of different useful use-cases. In this thesis we have presented our joint personalized model for session-based recommendation and time modeling. By basing the model on a hierarchical RNN architecture, we were able to use a low-cost user session-history for personalization of the recommendation and for providing time prediction for future sessions. The time prediction was shown to be highly competitive when combined with state-of-the-art time-prediction models, even without altering the loss function. We further introduced a tuning parameter to the loss function, which was shown to be capable of adapting the time-model's short-/long-term consideration. For both evaluated datasets, we found parameter settings which made the model outperform all baselines significantly on more than 85% of the testing data, only performing worse on the <15% of the longest and most infrequent time-gap durations. We also showed that the time modeling was robust, and able to adapt different time-gap distributions, by testing on synthetic generated time-gaps from different distributions. By loosely basing the generating distributions on the individual users, we got results which indicates that the model is able provide more personalized time-predictions through utilization of explicit user-modeling.

The model was also shown to achieve state-of-the-art intra-session recommendation based on comparisons with strong baselines. We observed that a non-joint setup of the model, where only intra-session recommendation was considered, preformed as good or better than the joint setup. This indicates that the intra-session recommendation of the model itself did not benefit from the joint training. However, our results does not exclude the possibility that it got better at modeling time in relation to recommendation, only that the final results were equal

or worse. We also showed that the model is capable of providing initial recommendations, based on inter-session modeling, which are comparable to the first intra-session recommendation(based on both inter- and intra-session modeling). However, we also uncovered that there is most likely room for improvement in the way inter-session modeling knowledge is propagated to the part of the model concerned with intra-session modeling and intra-session recommendation.

Overall, the goals from the research questions have been fulfilled, with the caveat that we did not see any improvement in the recommendation stemming from the time-modeling. As stated in the discussion, this might be due to conflicting gradient from the different tasks, where potential improvements in time modeling might come at the cost of general recommendation modeling. Furthermore, our model models time between abstract session representation, which is not tightly connected with the intra-session recommendations. Models that have observed significant improvements in recommendation through time modeling, usually have a more direct connection between the main recommendation and the time predictions. That being said, our model architecture allows it to jointly perform many tasks that are not usually combined. Furthermore, we have shown that the joint model performs all tasks well, so while the tasks might not be mutually beneficial for the model, they are clearly not incompatible either.

One area of interest for further work could be to add modeling of intra-session temporal aspects. This would most likely not be useful in domains where the intra-session time-gaps mostly are pre-defined action durations, e.g. song duration in music services. But for domains that involves idle time/periods of unobservable activity, it could be quite useful to model such inter-action time-gaps in addition to the inter-session time-gaps. If such modeling is applied, the new time-modeling will be more directly connected with the individual recommendations, which might prove to be mutually beneficial. There might also be insight to be gained from looking into other ways of incorporation the inter-session modeling into the intra-session RNN. This could for instance be a specialized RNN-cell architecture for temporal consideration. A third direction could obviously be to look into other time modeling parameterizations, which might reveal better options capable of more dynamic time predictions or better balancing of long-/short-term time prediction performance.

Bibliography

Bertin-mahieux, T., Ellis, D. P. W., Whitman, B., and Lamere, P. (2011). The million song dataset. In *In Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR.*

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Du, N., Dai, H., Trivedi, R., Upadhyay, U., Gomez-Rodriguez, M., and Song, L. (2016). Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1555–1564, New York, NY, USA. ACM.

Elman, J. L. (1990). Finding structure in time. *COGNITIVE SCIENCE*, 14(2):179–211.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852.

Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. (2015). Session-based recommendations with recurrent neural networks. *CoRR*, abs/1511.06939.

Hidasi, B., Quadrana, M., Karatzoglou, A., and Tikk, D. (2016). Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 241–248, New York, NY, USA. ACM.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Jannach, D. and Ludewig, M. (2017). When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 306–310, New York, NY, USA. ACM.

Jing, H. and Smola, A. J. (2017). Neural survival recommender. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 515–524, New York, NY, USA. ACM.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Liu, Q., Wu, S., Wang, D., Li, Z., and Wang, L. (2016). Context-aware sequential recommendation. *CoRR*, abs/1609.05787.

Liu, T., Moore, A. W., and Gray, A. (2006). New algorithms for efficient high-dimensional nonparametric classification. *J. Mach. Learn. Res.*, 7:1135–1158.

Ludewig, M. and Jannach, D. (2018). Evaluation of session-based recommendation algorithms. *CoRR*, abs/1803.09587.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Morse, S. and Chodrow, P. (2016). Parameter estimation for persistent communication cascades. https://stmorse.github.io/docs/6-867-final-writeup.pdf.

Muja, M. and Lowe, D. G. (2014). Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240.

Ogata, Y. (1981). On lewis' simulation method for point processes. *IEEE Transactions on Information Theory*, 27(1):23–31.

Quadrana, M., Karatzoglou, A., Hidasi, B., and Cremonesi, P. (2017). Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 130–137, New York, NY, USA. ACM.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2012). BPR: bayesian personalized ranking from implicit feedback. *CoRR*, abs/1205.2618.

Ruocco, M., Skrede, O. S. L., and Langseth, H. (2017). Inter-session modeling for session-based recommendation. *CoRR*, abs/1706.07506.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA. ACM.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Tan, Y. K., Xu, X., and Liu, Y. (2016). Improved recurrent neural networks for session-based recommendations. *CoRR*, abs/1606.08117.

Zhu, Y., Li, H., Liao, Y., Wang, B., Guan, Z., Liu, H., and Cai, D. (2017). What to do next: Modeling user behaviors by time-lstm. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pages 3602–3608. AAAI Press.