UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Exploiting Inter-session Dynamics for Long Intra-Session Sequences of Interactions with Deep Reinforcement Learning for Session-Aware Recommendation

Gustavo Junior Escobedo Ticona

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP SÃO CARLOS

Gustavo Junior Escobedo Ticona

Exploiting Inter-session Dynamics for Long Intra-Session Sequences of Interactions with Deep Reinforcement Learning for Session-Aware Recommendation

Dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Master in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Marcelo Garcia Manzato

USP – São Carlos
May 2021

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

E74e

Escobedo Ticona, Gustavo Junior
Exploiting Inter-session Dynamics for Long Intra-Session Sequences of Interactions with Deep Reinforcement Learning for Session-Aware Recommendation / Gustavo Junior Escobedo Ticona; orientador Marcelo Garcia Manzato. -- São Carlos, 2021.
86 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2021.

1. Sesssion-Aware Recommendation. 2. Deep Reinforcement Learning. 3. Hierarchical Recurrent Neural Networks. 4. Recommender Systems. I. Manzato, Marcelo Garcia, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Gustavo Junior Escobedo Ticona

Explorando dinâmicas entre sessões para sequências longas de interações de sessão com Aprendizado por Reforço Profundo para Recomendação Ciente de Sessão

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Marcelo Garcia Manzato

USP – São Carlos
Maio de 2021

To my beloved parents
for their unconditional support and love.

ACKNOWLEDGEMENTS

There are many groups of people that helped me through my formation that I would like to acknowledge.

To the University of São Paulo and the Institute of Mathematics and Computer Science(ICMC) who had given me the opportunity to complete my Masters studies offering high quality courses and an optimal environment for developing research.

I also would like to thank to all the members of the Intermídia Laboratory for their openness and kindness, specially to my supervisor Prof. Phd. Marcelo Manzato for having guided me through the development of this research. Also, to the RecSys team for sharing their points of view and suggestions.

To my parents for showing me that without hard work nothing can be accomplished.

To my friends and rest of family who always showed their appreciation and support.

"The true delight is in the finding out rather than in the knowing"
(Isaac Asimov)

RESUMO

ESCOBEDO T. G. J. **Explorando dinâmicas entre sessões para sequências longas de interações de sessão com Aprendizado por Reforço Profundo para Recomendação Ciente de Sessão** . 2021. 86 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2021.

Sistemas de recomendação são ferramentas cujo objetivo é filtrar o conteúdo relevante para os usuários de acordo com suas preferências. Recentemente, devido às novas demandas de negócios eletrônicos em que a maioria dos usuários não estão autenticados, surgiram os sistemas de recomendação baseados em sessão. Esta abordagem modela dados da sessão (por exemplo, sequências de interações, metadados de itens) para predizer quais itens serão relevantes para o usuário durante a sessão atual. As abordagens cientes de sessão incluem representações de sessões anteriores de usuários para melhorar o desempenho em novas sessões. No entanto, eles usam apenas essas representações no início da sessão, sendo que em uma longa sequência de interações não aproveita as possíveis mudanças de interesse durante a propria sessão. Os modelos atuais pressupõem que essas mudanças ocorrem apenas no início de uma nova sessão, conseqüentemente neste trabalho de pesquisa exploramos a possibilidade de usar essas representações entre sessões para beneficiar o desempenho das recomendações durante sessões longas. Propusemos uma adaptação do algoritmo Deep Deterministic Policy Gradient em um modelo de recomendação ciente de sessão para treinar uma política que lida com a interação entre o estado intra-sessão atual e as representações inter-sessão. Realizamos experimentos em dois conjuntos de dados de diferentes domínios, encontrando os principais fatores que afetam o desempenho dos modelos cientes de sessão. No entanto, não pudemos encontrar evidências fortes para afirmar que as dinâmicas entre as sessões podem melhorar o desempenho durante longas sequências de interações entre as sessões.

Palavras-chave: Recomendação Ciente de Sessão, Aprendizado por Reforço Profundo, Redes Neurais Recorrentes Hierarquicas, Sistemas de Recomendação.

ABSTRACT

ESCOBEDO T. G. J. **Exploiting Inter-session Dynamics for Long Intra-Session Sequences of Interactions with Deep Reinforcement Learning for Session-Aware Recommendation**. 2021. 86 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2021.

Recommender systems are tools whose objective is to filter relevant content to users according to their preferences. Recently, due to the new demands of electronic business where most of users are not authenticated, Session-based recommender systems emerged. This approach models session data (e.g. sequences of interactions, item metadata) to predict which items will be relevant for the user during the current session. Session-aware approaches include representations from users' past sessions to improve performance on fresh new sessions. However, current approaches only exploit these representations at the beginning of the session which in a long sequence of interactions does not take advantage of possible changes of interest during the same session. Consequently, in this research work, we explore the possibility of exploiting inter-session representations to improve recommendation performance. We proposed an adaptation of the Deep Deterministic Policy Gradient algorithm on a session-aware recommender model to train a policy that handles the interaction between the current intra-session state and inter-session representations. We performed several experiments on two datasets from different domains finding key factors that affect session-aware models' performance. However, we could not find strong evidence to claim that inter-session dynamics can improve performance during long sequences of intra-session interactions.

Keywords: Session-Aware Recommendation, Deep Reinforcement Learning, Hierarchical Recurrent Neural Networks, Recommender Systems.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

CBF	Content Based Filtering
CF	Collaborative Filtering
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DRL	Deep Reinforcement Learning
FN	False Negative
GRU	Gated Recurrent Unit
IR	Information Retrieval
LSTM	Long-short Term Memory
MRR	**Mean Reciprocal Rank**
NB	Neighborhood Based
NLP	Natural Language Processing
NNs	Neural Networks
R	**Recall**
RL	Reinforcement Learning
RL	Reinforcement Learning
RNNs	Recurrent Neural Networks
RS	Recommender Systems
SARS	Sequence-Aware Recommender Systems
SBRS	Session Based Recommender Systems
TP	True Positive

CONTENTS

CHAPTER

1

INTRODUCTION

1.1 Motivation

In the last two decades, the outstanding success of the World Wide Web has set out several challenges to the Information Retrieval (IR) area (BAEZA-YATES; RIBEIRO-NETO, 2008). The huge volume of data generated every day pose the main challenges such as managing, representing, retrieving and filtering. In order to tackle these problems, efficient methods were introduced. An example of these methods is called Recommender Systems (RS) whose main objective is to deliver relevant personalized content to the user. These tools have been widely applied in Web applications from many domains such as electronic businesses and media consuming applications making them more profitable. To fulfill their objective, these systems rely on modeling techniques and data from many sources (i.e. users' profiles, content structure). All these data are computationally represented, processed and filtered to be delivered to users (AGGARWAL, 2016).

In order to provide relevant recommendations, many approaches have been proposed, ranging from nearest neighbors methods, one of the earliest approaches, to latent factor models that are considered state-of-art techniques. Traditional approaches only deal with two types of entities, users and items, without including contextual information such as time or place. Moreover, including contextual information to the model may help refining recommendations under certain circumstances (ADOMAVICIUS; TUZHILIN, 2011). Consequently, Context-aware recommender systems were introduced to improve performance of traditional models. Also with this addition, new challenges for modeling and interpreting context were set out.

Most of the RS traditional techniques rely on matrix based structures which are designed to represent static users' preferences, which by nature are dynamic. In an effort to model dynamic aspects, works like (KOREN, 2010) adapted the traditional methods

to include temporal information as contextual signal, in which long-term and short-term user interests were analyzed obtaining new significant improvements. With these new findings, modeling evolving users' interests for recommender systems gained a lot of attention, which originated a new group of techniques that model the sequential nature of user interactions through time, called Sequence-Aware Recommender Systems (SARS). These new techniques adopt sequential pattern mining techniques on time-ordered users' interactions logs to find common patterns of user behavior (QUADRANA; CREMONESI; JANNACH, 2018).

Recently, Deep Learning (DL) techniques had great success in many areas of Computer Science such as Image Processing, Speech Recognition, Handwritten Text Generation and Sequence Pattern Mining. These new advances influenced the RS community to focus research efforts into the usage of DL techniques to leverage their main advantages to improve performance (HIDASI *et al.*, 2017). This and the necessity of modeling user temporal dynamics, motivated the usage of Recurrent Neural Networks for modeling sequential user interactions due to their proven capacity at mapping sequential patterns mostly for Natural Language Processing (NLP).

Many efforts have been developed in the last few years to introduce deep learning sequence modeling techniques into recommendation tasks. One solid example of application is session-based recommendations, where the main objective is to predict which is the next item to be consumed by the user, based on current session data. Unlike traditional RS techniques session-based approaches have a clear advantage when there is no previous user data available. Under this category of techniques, session-aware approaches emerged with the aim of using users' past sessions information to improve predictions on fresh new sessions of returning users (QUADRANA *et al.*, 2017). These approaches are mainly composed of an inter-session component that models session-to-session dynamics(long-term) and an intra-session component(short-term) that models item-to-item dynamics. In several works, these two components are modeled with Hierarchical Recurrent Neural Networks, which consists in two networks, the inter-session network models the transitions between the resulting states of the intra-session network when a session ends and the intra-session network models transitions between interactions within the same session (QUADRANA *et al.*, 2017)(VASSØY *et al.*, 2019)(RUOCCO; SKREDE; LANGSETH, 2017). These approaches use the resulting internal state of the inter-session component to initialize the internal state of the intra-session component when a fresh new session starts to introduce users' past sessions information. Moreover, the training is performed using time-ordered sequences of interactions grouped in time-ordered sessions.

Other learning paradigms were also introduced to the RS domain such as Reinforcement Learning (RL). These techniques consist on the interaction between an agent, that holds a policy, and an environment. The learning objective is to optimize the pol-

icy, held by the agent, that decides which is the optimal sequence of actions, based on a reward value, to take according to the current state of the environment to fulfill a given goal. Most of recent advances in this area owe their popularity for their success in complex tasks such as Robotic Control or computer programs that can defeat humans in board games such as AlphaGo[1]. Furthermore, the advances of DL techniques and their application to this area originated a sub-area called Deep Reinforcement Learning (DRL), in which Deep Neural Networks are adopted as main modeling techniques. Some key features of these approaches are their long-term goal orientation and ability to learn policies on very complex environments based the interaction of the agent and environment through time. There are two main groups of techniques for model-free environments [2], policy-based and value-based, which combination gave origin to a powerful group of techniques called Actor-Critic methods which have two main components, an Actor that predicts the next action to be taken based on the current state of the environment(policy optimization) and a Critic that will evaluate the goodness of the action taken(value optimization). The main advantages of this approach is that it can learn policies without knowing about the mechanics of the environment. Additionally, with the critic network it is possible to have a robust estimator of the goodness of the current state.

In the RS field and especially for the e-commerce domain, several approaches based on Actor-Critic methods were explored for sequential recommendation obtaining competitive and better results than previous sequential RS techniques(ZHAO *et al.*, 2018b)(ZOU *et al.*, 2019). However, these new approaches pose several new challenges for the RS domain such as high variance and constantly changing data distribution.

1.2 Justification

Most session-aware approaches use representations of past sessions to initialize fresh new sessions towards improving their predictions, however, the longer the sessions the less useful this initialization becomes because users tend to change their short-term interests during a long sequence of interactions. This is the case for instance when a user is browsing for an item of a certain category and finds it, if the session continues, he is very likely to search for other items that a re not related to his first intent. The same can be applied to other domains like music when a user is listening to a certain genre or artist and then shifts to a different one due to a "mood change" during the same listening session. Previous works do not address the detection of these changes within the same session (QUADRANA *et al.*, 2017; VASSØY *et al.*, 2019; RUOCCO; SKREDE; LANGSETH, 2017; HIDASI *et al.*, 2015), which could be beneficial in a long sequence of interactions scenario (LUDEWIG; JANNACH, 2018). Detecting these changes does not

[1] <https://deepmind.com/research/case-studies/alphago-the-story-so-far>
[2] see Section 2.5.2

have a trivial solution due to the chaotic nature of user behavior during sessions. Therefore, there is no method split a session based on user interest changes. However, these changes are more likely to happen from session to session(inter-session dynamics). Additionally, in recent session-aware approaches the inter-session and intra-session components are represented by dense vectors(internal states of networks) in latent space, which makes detecting changes even a harder problem.

Given the problematics previously exposed, in this research work, instead of detecting changes during long sequences of interactions, we explore the possibility of exploiting inter-session dynamics between contiguous sub-sessions to introduce the topic-changing nature of different sessions by combining both inter and intra session states to adjust the current intra-session. To accomplish this, we need a method to select which features of the inter-session state to combine with the intra-session state, which is unsupervised data problem. Therefore, we apply Deep RL techniques to find a policy that can analyze the current states of both levels to select which inter-session state features should be combined with the current intra-session state based on resulting performance.

1.3 Hypothesis

The main hypothesis of this research is that exploiting inter-session dynamics according to the current context of a session can improve performance for long sequences of interactions within the same session in a hierarchical recurrent neural network for session-aware recommendation. Consequently we defined the following research questions:

- What strategies should our proposed agent adopt to perform its actions?
- How should we build the observations that we use to train our agent?
- How well does our proposal performs only for the long sessions scenario?
- How well does our proposal performs in the overall recommendation scenario?

1.4 Objective

In this research work we have as main objective to use deep reinforcement learning techniques to train a policy that exploits inter-session dynamics information to improve the current state of intra-session predictions for long sequences of interactions within the same session in a hierarchical recurrent neural network for session-aware recommendation. Additionally to our main objective we defined the following specific objectives:

- Adapt the Deep Deterministic Policy Gradient algorithm for the session-aware recommendation scenario to train a policy that can deal with the interaction of inter-intra session representations
- Develop a method to train our policy with sufficient data
- Explore how well our proposal performs for the global and long sessions recommendation scenario
- Explore how well our proposal performs alongside its baselines according to different factors such as history length, session average length, and average recurrent session length

1.5 Document Organization

This text is organized as follows: In Chapter 2, basic concepts of recommender systems are described. Following, in Chapter 3 there is a review of recent works relevant to this research. Further, in Chapter 4, a detailed description of the proposal, methodological aspects and needed tools, are discussed. Moreover, in Chapter 5 we describe our experiments' design and results. Finally in Chapter 6 we present a summary of this research work, main contributions, conclusions and future works.

CHAPTER

2

THEORETICAL BACKGROUND

In this Chapter, all the relevant related concepts within this research will be addressed. This Chapter is organized as follows: Section 2.1 describes the basic concepts and paradigms of Classical Recommender Systems. Next, in Section 2.3 we review the elements of Session-Based Recommender Systems. Following in Section 2.4 we make a brief review of Neural Recommendation techniques. Then, in Section 2.5 we review the fundamentals of Reinforcement Learning. Finally, in Section 2.6 we present our final remarks for this Chapter.

2.1 Classical Recommendation Techniques

Recommender Systems (RS) are tools designed for filtering relevant information from overwhelming amounts of data. These tools have gained great importance over the last decades, due to their widely application on several domains such as e-commerce and media services applications making them more profitable. To accomplish such objectives, several sources of information are used, for instance users' preferences records, content features, demographic information, spatio-temporal information, etc., along with many modeling techniques. In the following sections, we will review the main approaches for recommenders including Collaborative Filtering, Content Based Filtering, Knowledge Based Filtering and the widely used Context-Aware Recommender Systems.

2.1.1 Collaborative Filtering

Being one of the most popular approaches in literature, collaborative filtering is defined as the set of techniques that uses users' ratings to estimate the relevance of unseen items for other users in the community. It is based on the idea that ratings are highly correlated among users, being its main challenge the underlying sparsity of ratings matri-

ces (AGGARWAL, 2016). Collaborative Filtering (CF) approaches can be subdivided in *memory-based* and *model-based* approaches, as described next.

2.1.1.1 Memory-based CF

This approach is also called Neighborhood Based (NB), so information filtering is made according to their proximal elements (users or items). It uses the *user*×*item* matrix and then, using a similarity function (e.g. Euclidean, Jaccard, Cosine), it determines the similarities between rows or columns. For example, if the users A and B had rated almost the same movies with similar values then we can say that A and B are similar, and depending on how similar they are we can obtain a ranking of similar users. There is a similar scenario for items but we use columns instead of rows (AGGARWAL, 2016).

The main advantage of this approach is its simplicity of implementation and the generation of explainable recommendations. However, it does not perform very well when dealing with sparse matrices, for instance when there are not sufficiently similar users to calculate rating for a given user, which causes lack of full *coverage* of ratings predictions. Also, it requires a lot of resources (time and space) during its offline preprocessing stage when dealing with large volumes of data (AGGARWAL, 2016).

2.1.1.2 Model-based CF

The main feature of this approach is the usage of models brought from other areas of science in order to predict ratings based on many sources of information. These models are capable to manage with the complexity, sparsity and constant information drift. Most of the techniques used for this kind of RS come from vastly studied machine learning algorithms, because the matrix completion process is a special case of traditional classification and regression problems. Some clear advantages over memory-based methods are listed below (AGGARWAL, 2016):

- *Space efficiency:* Typically, trained models are smaller representations of original rating matrices.
- *Training speed and prediction speed:* These models are often faster in the data preprocessing stage. And despite their compact and summarized representations, predictions are efficiently calculated, in most cases.
- *Overfitting Avoidance: Overfitting* is a serious problem in machine learning techniques, in which prediction are overly influenced by random artifacts in the data. In model-based approaches *regularization* techniques can be used to alleviate this problem.

2.1.2 Content-based Filtering

For Content Based Filtering (CBF), the intrinsic structure of items is analyzed to find better similarity between items. In the field of recommendations the common schema is to gather item metadata to generate better representations. These data range from tags, reviews, item textual description to audio spectrum, video features, etc. This kind of recommender depends on two sources of data:the user profile and description of items. For this approach, we have three basic components (AGGARWAL, 2016):

- *Preprocessing and feature extraction:* In this stage, the intrinsic content features are processed to create vector representations that are domain specific, for instance, keyword-based vector-space for news representation. The selection of features to represent items is a key process, which affects the performance in any content-based recommender.

- *Content based learning of user profiles:* The objective of this stage is to create a user specific representation based on historical data, in order to predict user interest in items. For this purpose, explicit and implicit feedback are combined with attributes of items to create the training data. The resultant *user profile* relates user interests to item attributes.

- *Filtering and recommendation:* In this stage, predictions for specific users are performed.

This approach has good performance when the item is new and there are few ratings for that item. Also, it produces explainable recommendations, which not always can be accomplished with CF approaches. Content-based methods can use *Off-the-shelf* text classifiers with relatively little engineering effort. On the other hand, this approach can not deal with new users because of its dependency on user profiles. Additionally, it tends to fall into the *overspecialization* problem showing just similar items to the ones that the user has already seen.

2.2 Evaluation of Recommender Systems

Evaluating RS is not a trivial task but follows common guidelines. In this section, we will review the evaluation protocols for RS adopted in this work.

2.2.1 Offline evaluation

Offline evaluation is the most common schema in RS because standardized evaluation frameworks have been developed for such cases. The main disadvantage of this

evaluation method is that it does not measure the actual propensity of the user to react to the recommender system in the future. However, the quantifications that this method provides are statistically robust and easily understandable (AGGARWAL, 2016). In following paragraphs we describe the metrics used throughout this research work.

Recall (R) is an accuracy based metric and is defined as the fraction of relevant documents that are retrieved (Equation 2.1). We also can consider it as the fraction of True Positive (TP) samples and the sum of the TP and False Negative (FN) samples (Equation 2.2) (MANNING; RAGHAVAN; SCHüTZE, 2008).

$$Recall = \frac{\#\ relevant\ items\ retrieved}{\#\ relevant\ items} \tag{2.1}$$

$$R = \frac{TP}{TP + FN} \tag{2.2}$$

Mean Reciprocal Rank (MRR) is ranking based metrics and is useful for evaluating how early in the list the first relevant recommended item appears. Reciprocal Rank is the inverse of the position of the first relevant item (BAEZA-YATES; RIBEIRO-NETO, 2008). Let *rank* be the predicted rank position of an item i and K the threshold for ranking position,

$$RR@K(i) = \begin{cases} \frac{1}{rank_i}\ ,\ if\ rank_i \leq K \\ 0\ ,\ otherwise \end{cases} \tag{2.3}$$

The mean reciprocal rank is given by the mean of RR for all the predicted items I:

$$MRR@K(I) = \sum_{i}^{I} RR(rank_i) \tag{2.4}$$

2.2.2 Protocols for Evaluation

In order to measure the performance of our trained models on unseen data, there are several protocols. These protocols follow common guidelines brought from machine learning evaluation methods as (AGGARWAL, 2016):

- *Holdout:* The dataset is divided in two groups training and test, and the performance of the algorithm is measured on the predicted values for the test partition after having trained the model with the training partition.

- *K-fold cross-validation:* This method divides the dataset in k partitions to then perform training with $k-1$ partitions (folds) and testing with the remaining one. This process is repeated k times, one for each fold.

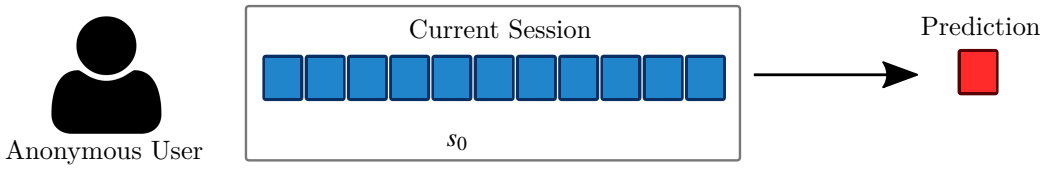


Figure 1 – Session-based recommendation scenario

2.3 Session-Based Recommender Systems

Unlike traditional recommendation techniques, Session Based Recommender Systems (SBRS) do not rely on user's long-term historical data to make their predictions. Instead, they gather data generated from interactions performed within the current session with the main goal of adapting predictions to match the user's short-term interest (QUADRANA; CREMONESI; JANNACH, 2018).

In order to perform training, this type of RS requires to split long user's activity logs into sessions. A session can be defined as a set of interactions performed by the user during a period of time that is heuristically limited and domain dependent, for example, we could establish the end of a session and the start of a new one whenever we have an idle time of 30 minutes between two interactions (LUDEWIG; JANNACH, 2018).

In recent years most approaches treat this problem a sequence pattern mining task with the objective of predicting the next action that the user will perform. This group of techniques is known as **sequence-aware** RS which implies several challenges such as context-adaptation and trend-detection. According to the interaction between long-term and short-term user's interests, session based recommendation has two main scenarios (QUADRANA; CREMONESI; JANNACH, 2018):

- **Session Based Recommender Systems** As we defined before it bases its predictions on user's short-term interests (Figure 1).

- **Session-aware Recommender Systems** This kind of recommender makes a combination of long-term and short-term interests to improve accuracy. Therefore, these RSs use past sessions' information alongside the current session (QUADRANA; CREMONESI; JANNACH, 2018) (Figure 2).

In this work, our focus is on session-aware recommender systems for which we introduce a method to exploit user long-term interests representations to address long sequences of session interactions where short-term interest might change.

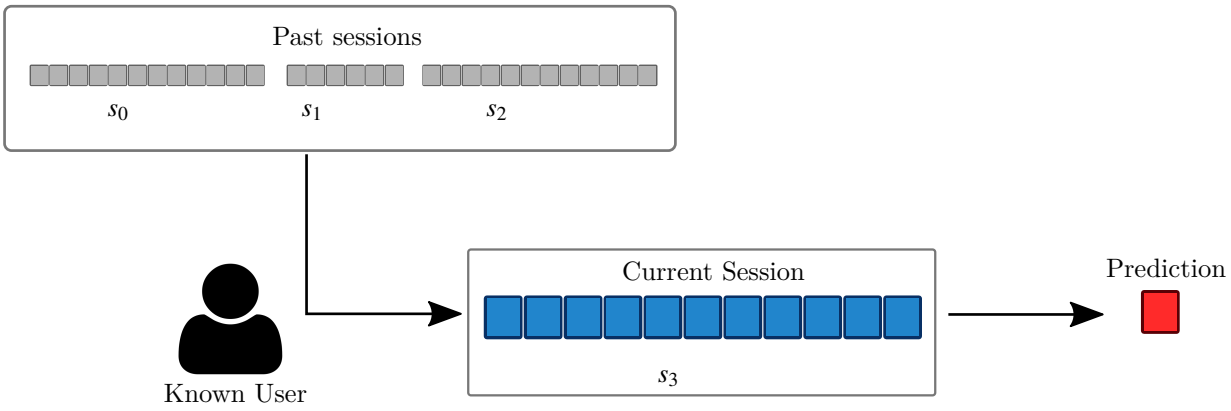


Figure 2 – Session-aware recommendation scenario

2.4 Neural Recommendation

Neural recommendation is a sub-area of RS in which modeling and representation techniques are inspired by Neural Networks (NNs). Common neural inspired techniques for sequential modeling are depicted in the following sections.

2.4.1 Recurrent Neural Networks

Modeling sequences for the recommendation task has gained importance because matrix based approaches aren't specifically designed to support the constant changes in preferences of users and content. There have been many efforts to exploit patterns from series of continuous interactions of users and web applications. Consequently, with the rise of Deep Learning techniques, Recurrent Neural Networks (RNNs), which is a family of neural networks for processing sequences of data, were introduced to Recommender Systems to address the dynamic nature of users and items (WU *et al.*, 2017). They are based in one of the earliest machine learning concepts, parameter sharing across different parts of the network (GOODFELLOW; BENGIO; COURVILLE, 2016), which allows the model to generalize across inputs of different forms (variable length). A brief description of the RNN internals is presented below.

Recurrent neural networks represent the position of values in sequences with the index t and consider the hidden state h_t as the current system status, which is updated by feeding the model with data representational vectors x_t and a non-linear function f from time to time.

$$h_t = f(h_{t-1}, x_t) \tag{2.5}$$

Usually, this function f is a linear transformation plus a non-linear activation function for instance in:

$$h_t = \tanh(W[h_{t-1}, x_t] + b) \tag{2.6}$$

where the parameters related to h_{t-1} and x_t are combined into the matrix W, and b is a bias term. Also, the activation function, for this case **tanh**, is applied to every element of its input. The task of an RNN is to learn the parameters W and the bias term b (ZHOU *et al.*, 2016).

There is a specialized type of architecture that uses several internal mechanisms (gates) for controlling signals called gated RNN. The most well known ones are *Long-Short Term Memory* and *Gated Recurrent Unit*, that in practical situations have similar behavior (DONKERS; LOEPP; ZIEGLER, 2017). They are reviewed below.

2.4.2 Long-short Term Memory

Long-short Term Memory (LSTM) is one of the most successful architectures of gated RNNs, it consists in adding several control gates inside each cell and self-loop paths where the gradients can flow for longer durations (HOCHREITER; SCHMIDHUBER, 1997). The addition of these control gates solves the vanishing/exploding gradient problem of regular RNNs. The formal definitions of these control gates are given by the following equations:

$$\begin{aligned} i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\ f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\ o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ \tilde{c_t} &= \tanh(W_c[h_{t-1}, x_t] + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c_t} \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{2.7}$$

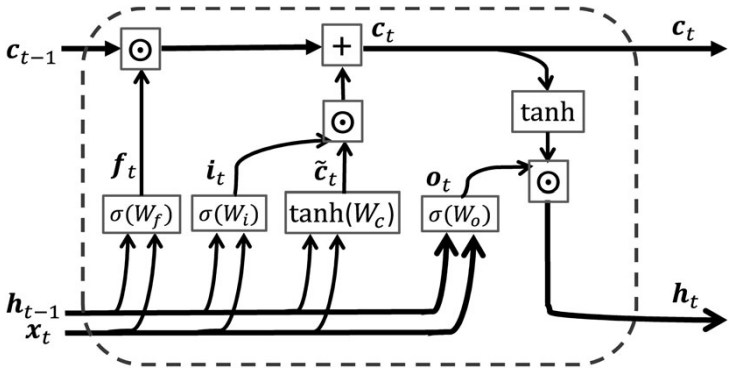
where i_t, f_t, o_t are equations for the *input*, *forget* and *output* gates, respectively, $\odot$ is the component-wise product between two vectors and σ represents the logistic sigmoid function (applied to every component of the input) and is defined by the equation:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

All these inner components and their interactions are depicted in Figure 3 that shows the architecture of LSTM.

The model above is described as follows (ZHOU *et al.*, 2016):

- There is one additional hidden state c_t that, in addition to h_t, helps maintaining long-term memories.
- The forget gate f_t determines the portion of c_{t-1} to be remembered, calculated by the parameters W_f and b_f.



Source – (ZHOU *et al.*, 2016)

Figure 3 – Long-short term memory architecture

- The input gate i_t determines which portion of time t's new information is to be added to c_t with parameter W_i and b_i.
- The inputs are transformed as the update $\tilde{c}_t$ with parameters W_c and b_c. Then $\tilde{c}_t$ (weighted by i_t) and c_{t-1} (weighted by f_t) form the new cell state c_t.
- The output gate o_t is determined by parameters W_o and b_o, and controls which part of c_t is to be output as the hidden state h_t.

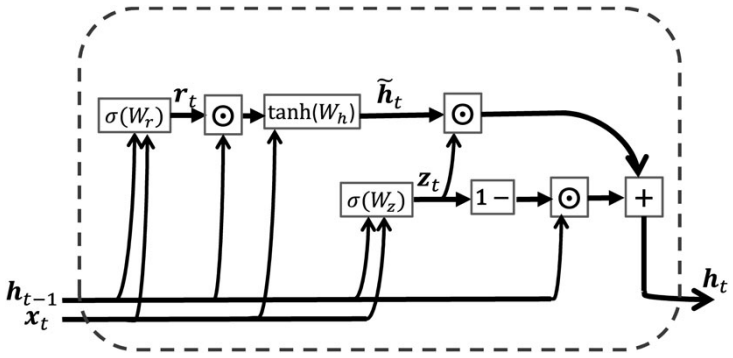
2.4.3 Gated Recurrent Unit

Gated Recurrent Unit (GRU) is another architecture that simplifies the LSTM-like units. The main difference is that a single gating unit simultaneously controls the forgetting factor and the decision to update the state unit. Each GRU contains an update gate z, whose role is similar to the LSTM forget gate, and a reset gate r, which mimics the role of the LSTM input gate. It is represented by the following equations:

$$\begin{aligned} r_t &= \sigma(W_r[h_{t-1}, x_t] + b_r) \\ z_t &= \sigma(W_z[h_{t-1}, x_t] + b_z) \\ \tilde{h_t} &= \tanh(W_h[r_t \odot h_{t-1}, x_t] + b_h) \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h_t} \end{aligned} \tag{2.9}$$

To illustrate the data flow, the architecture of the GRU is shown in Figure 4. It can be seen in the Figure 4 that the LSTM output gate was removed as well as the additional hidden state gate c, which means that GRU units have less computational complexity. Furthermore, GRU units have slightly better performance (CHUNG *et al.*, 2014).

In this section we have reviewed RNNs' internals and common RNN cell architectures pointing out their main capabilities. In this research, we will leverage the sequence modeling capabilities of RNNs to tackle session-aware recommendation.



Source – (ZHOU *et al.*, 2016)

Figure 4 – Gated Recurrent Unit architecture

2.4.4 Training Session-Based Recommender Systems

In this Section we are going to review some of the most common practices adopted for training session-based recommender systems including data split protocols and session creation.

2.4.4.1 Split Protocols

There are several dataset splitting protocols to try to emulate a real scenario of session-based recommendation that may vary according to the objective of the system. Additionally, to build sessions out from a user's interactions logs most systems adopted a heuristic methodology based on the idle time of the user (i.e. 30 minutes), others adopted for example one day of interactions (LUDEWIG; JANNACH, 2018). After building sessions we can split the data to perform training.

- *Temporal-Holdout:* when the dataset is split according to a defined timestamp or period (e.g. last day, last week, last month of interactions).
- *Session-Holdout:* when the dataset is split with defined proportion considering one session as an example.
- *Last-n-sessions-out:* when the last n user sessions are separated for holdout.
- *User-Holdout:* when a defined subset of users is separated for testing according to a defined proportion.

We adopted the *last-n-sessions-out* for n=*1* or *last-session-out* as in (QUADRANA *et al.*, 2017).

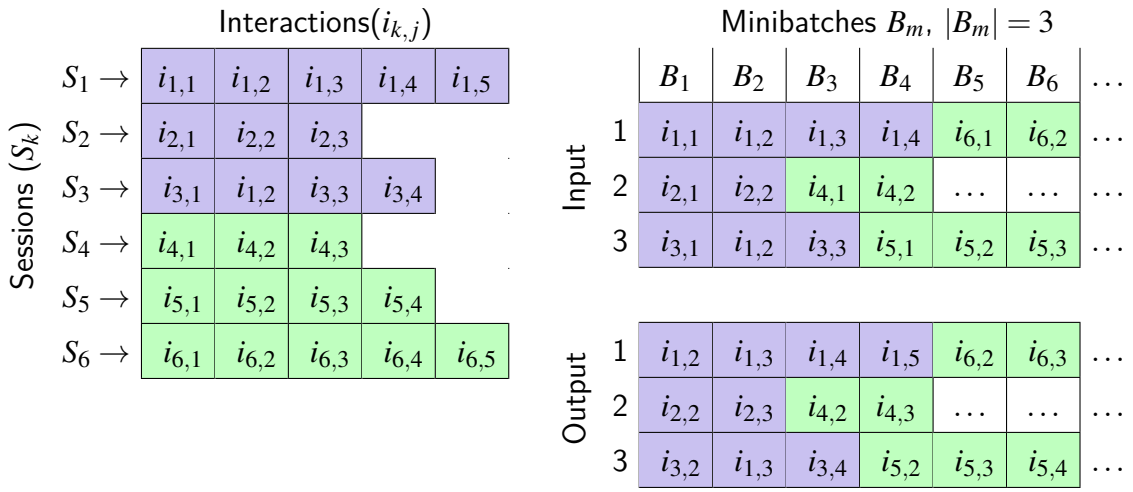


Figure 5 – Session-Parallel Mini batches adapted from (HIDASI *et al.*, 2015). As we can see for the batch size $|B_m| = 3$ as soon as the last interaction $i_{2,2}$ of the session S_2 is processed in B_2, the first interaction $i_{4,1}$ of the session S_4 is inserted in the position 2 of the batch B_3.

2.4.4.2 Training RNN-based SBRS

To train RNN based SBRS in (HIDASI *et al.*, 2015) the authors introduced Session-Parallel Minibatches, a novel method to train over anonymous sessions that consists in creating a batch that includes in each position a pair of sequential interactions i_k, i_{k+1} where the input i_k is mapped to the output i_{k+1} until the session finishes, to then fill in that position in the batch with the next session's first pair of interactions. Under this setting we consider the session as being independent from each other, we can see an example in Figure 5.

In (QUADRANA *et al.*, 2017) the authors made an adaptation for the case of non-anonymous sessions or user sessions called user session-parallel minibatches that consist in chronologically ordering a sequence of user sessions to then use each position of the batch to process all the sessions of the same user. After we finished processing one user's sessions, we replace that position in the batch with the next user's interactions, an example is depicted in Figure 6. Throughout this research we use the user session parallel mini batches with some adaptations to train our models.

2.5 Reinforcement Learning

Reinforcement Learning (RL) is a set of computational techniques that facilitate the understanding and automating of the learning process and decision making for a given task. This approach main feature is that the learning of the agent is carried out by interacting with its environment without the need for supervised data. The representation of these techniques follows the Markov Decision process framework due to its awareness

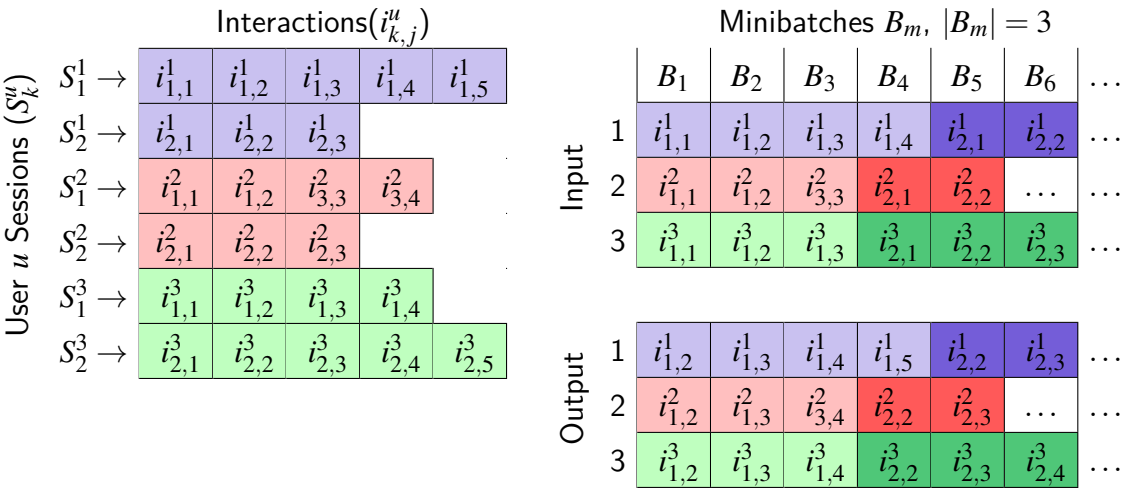


Figure 6 – User Session-Parallel Mini batches adapted from (QUADRANA *et al.*, 2017). As we can see for the batch size $|B_m| = 3$ as soon as the last interaction $i^2_{1,3}$ of the user session S^2_1 is processed in B_2, the first interaction $i^2_{2,1}$ of the same user's following session S^2_2 is inserted in the position 2 of the batch B_3

of cause-effect, non-determinism, and goal orientation (SUTTON; BARTO, 2018).

2.5.1 The RL Problem Setting

The basic setup of a RL scenario has two main components the **agent** and the **environment** that interact in a loop of constant feedback. On every step of this loop, the agent relies on a **policy** (π) to generate an optimal **action** A_t for the current **state** S_t and **reward signal** R_t, this action is performed on the environment which will return a new state S_{t+1} and reward signal R_{t+1} completing the loop (DING *et al.*, 2020) as shown in Figure 7. In following Sections we will describe the formulation of an RL problem.

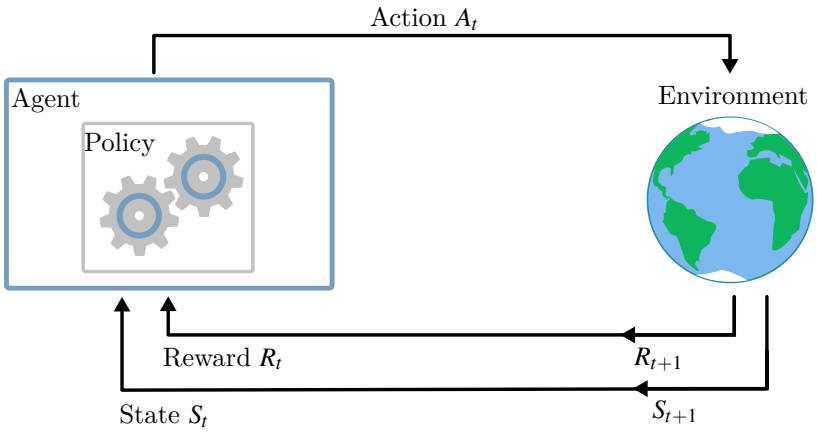


Figure 7 – The RL problem setting

2.5.1.1 Action Spaces

RL has diverse applications therefore environments might be different from one problem to another and actions. An **action-space** is defined as the set of valid actions that can be performed by the agent. In some cases we can have a **discrete** space of actions that is the case for a game like Chess, or **continuous** for tasks like deciding the next rotation of a robotic arm (DING *et al.*, 2020).

2.5.1.2 States and Observations

The state contains all the environment's information at a given moment, from which we gather observations **o**. An environment is considered **fully observable** or **partially observable** depending on whether the observations contain complete or partial state's information (DING *et al.*, 2020).

2.5.1.3 Trajectories

We can call a trajectory τ to the sequence of states (S_t) , actions (A_t) and rewards (R_t) that the interaction of the environment and the agent generates over time (DING *et al.*, 2020).

$$\tau = (S_0, A_0, R_0, S_1, A_1, R_1 \ldots) \tag{2.10}$$

2.5.1.4 Reward and Return

The reward signal R_t (Equation 2.11) serves as an indicator of the immediate performance of the agent on a single step. However, the objective of our agent is to find the optimal policy π^* that maximizes the future accumulated reward. Therefore we define **return** $R(\tau)$ (Equation 2.12) as the accumulated reward for a given trajectory τ (DING *et al.*, 2020).

$$R_t = R(S_t) \tag{2.11}$$

$$R(\tau) = \sum_{t=0}^{T} R_t \tag{2.12}$$

In a trajectory the closer the events the greater the impact on its accumulated reward, for this reason, a **return discount factor** $\gamma \in [0,1]$ is included . Therefore we define the **discounted return** (Equation 2.13) as a weighted sum of the reward of each step in the trajectory.

$$R(\tau) = \sum_{t=0}^{T} \gamma^t R_t \tag{2.13}$$

An agent has to estimate the expected return from a state $s \in S$ to decide which following state of the trajectory will have greater value, to this we use a **value function** $V(s)$

(Equation 2.14).

$$V(s) = \mathbb{E}[R_t|S_t = s] \tag{2.14}$$

2.5.1.5 Policies

An agent uses a policy π (Equation 2.15) to handle the state $s \in S$, gathered from the environment, and chooses the optimal action $a \in A$ from the probability distribution of transitions p that will maximize the expected return $J(\pi)$ (Equation 2.16). In order to do that, we optimize to maximize this policy to choose the best policy π^* for all trajectories τ, we can denote the optimal policy as in Equation 2.17 (DING *et al.*, 2020).

$$\pi(a|s) = p(A_t = a|S_t = s) \tag{2.15}$$

$$J(\pi) = \int_{\tau} p(\tau|\pi)R(\tau) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{2.16}$$

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{2.17}$$

Given the optimal policy π^*, the **value-function** V^{π} (Equation 2.18) is the expected return for the state s .

$$V^{\pi}(s) = \mathbb{E}_{\tau\sim\pi}\Big[R(\tau)|S_0 = s\Big] = \mathbb{E}_{A_t\sim\pi(.|S_t)}\left[\sum_{t=0}^{\infty}\gamma^t R(S_t, A_t)|S_0 = s\right] \tag{2.18}$$

An **on-policy value function** is the value function that depends on a policy π that we denote as $V^{\pi}(s)$ (Equation 2.19), that is optimal when it depends on π^* and is denoted as $V^*(s)$ (Equation 2.20), known as **optimal value function**.

$$V^{\pi}(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0 = s] \tag{2.19}$$

$$V^*(s) = \max_{\pi} \mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0 = s] \tag{2.20}$$

To estimate the expected return starting from a state s, an arbitrary action a and the current policy π, we use an **action-value function** $Q^{\pi}(s,a)$ (Equation 2.21) which once we have the optimal policy π^* we call **optimal action-value function** $Q^*(s,a)$ (Equation 2.22).

$$Q^{\pi}(s,a) = \mathbb{E}_{\tau\sim\pi}\Big[R(\tau)|S_0 = s, A_0 = a\Big] = \mathbb{E}_{A_t\sim\pi(.|S_t)}\left[\sum_{t=0}^{\infty}\gamma^t R(S_t, A_t)|S_0 = s, A_0 = a\right] \tag{2.21}$$

$$Q^*(s,a) = \max_{\pi} \mathbb{E}_{\tau\sim\pi}\Big[R(\tau)|S_0 = s, A_0 = a\Big] \tag{2.22}$$

2.5.2 Taxonomy of RL Algorithms

It is really hard to establish an adequate classification for each RL algorithm because of their diverse foundations, in this Section we will introduce the most relevant criteria that can help us to identify the basic features for each group of algorithms. Nonetheless, this classification might not be the completely accurate but reasonably useful (ACHIAM, 2018).

2.5.2.1 Model-based vs Model-Free RL

We can divide RL algorithms in two big groups: model-based and model-free based on whether they have access to the model of the environment or they have to learn it (ACHIAM, 2018).

For **model-based** approaches, a model can be defined as set of predefined rules that allows the agent to plan ahead, so it chooses among a set of possible trajectories. To learn its policy we can use supervised learning techniques on the observations (s,a,s',r) collected by performing actions on the environment. This approach main advantage is that we can have better results easily, however, the models are not always available.

In a **model-free** approach we try to find a policy by directly optimizing the reward without focusing on the model of the environment. This approach relies on experience which over iterations improves the performance of the policy, being its main drawback the cost of exploration in the real-environment.

In this work we are going to focus on model-free algorithms and in the following Sections we will review policy optimization approaches.

2.5.2.2 Value-based vs Policy-based

Policy optimization approaches can be classified according the function that they try to optimize (ACHIAM, 2018). A **value-based** approach's objective often is to optimize its action-value function $Q^{\pi^*}(s,a)$. This approach's main advantage is its high efficiency due to its small variance of value function estimation. Additionally, this approach is hard to fall into a local optimum. However, it is often limited to discrete action spaces.

A **policy-based** algorithm optimizes the policy directly by maximizing the accumulated reward. This approach is suitable for continuous high dimensional action spaces. Additionally, this approach has better convergence and a simpler process for parameter definition than value-based methods.

We have divided policy optimization methods into policy-based and value-based however the most popular methods for this purpose is a combination of these two, also known as **actor-critic** (SUTTON; BARTO, 2018) methods. In this approach we learn an

action-value function to reduce the variance of the policy search space, which is suitable for discrete and continuous actions spaces.

2.5.3 Deep Deterministic Policy Gradient

The Deep Deterministic Policy Gradient (DDPG) is an off-policy RL algorithm proposed by (LILLICRAP *et al.*, 2015), whose main feature is its ability to learn tasks in environments with continuous action spaces. This is an algorithm that works under the standard RL setup and has a close relation with Q-learning (WATKINS; DAYAN, 1992).

The expected reward for every possible state of the policy π is defined by an action-value function $Q^*(s,a)$, defined by Bellman's equation (Equation 2.23) (SUTTON; BARTO, 2018) where $\mathscr{P}$ is a set for all possible states, $r(s,a)$ is the reward for the current state and a discount factor γ for the next state action-value.

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s' \sim \mathscr{P}} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right] \tag{2.23}$$

The policy π chooses the optimal action a^* for the current state s in order to obtain the best possible reward as depicted in Equation 2.24:

$$a^*(s) = \arg\max_a Q^*(s,a) \tag{2.24}$$

As we can see in Equation 2.24, to choose the best action we calculate the best value over all possible states. This works well for a finite number of actions, but when working in continuous action spaces this calculation becomes expensive if done for every state. DDPG (LILLICRAP *et al.*, 2015) approximates the optimal value function with a differentiable policy $\mu(s)$, that takes advantage of the continuity of the action space, then $\arg\max_a Q^*(s,a) \approx Q(s,\mu(s))$.

Let $Q_\phi(s,a)$ with parameters ϕ be our approximator for the optimal policy $\mu(s) = \arg\max_a Q(s,a)$ which is optimized by minimizing the loss function in Equation 2.25, where $\mathscr{D}$ is a set of transitions (s,a,r,s',d) that include flags for terminal states d, the action value function y_t is then calculated by the Bellman's equation (Equation 2.26):

$$L(\phi,\mathscr{D}) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim\mathscr{D}} \left[\left(Q_\phi(s,a) - y_t \right)^2 \right] \tag{2.25}$$

$$y_t = r(s,a) + \gamma(1-d) \max_{a'} Q_\phi(s',a') \tag{2.26}$$

In the Algorithm 1 we can see the whole process of policy optimization, which mainly consists in the optimization of the policy by learning the θ and ϕ parameters in two well

known stages *soft-update* in line 15 where we optimize the target parameters and *hard-update* in line 16 where we assign the current policy and Q-function parameters with a penalizing factor *gamma*.

Algorithm 1 – Deep Deterministic Policy Gradient (LILLICRAP *et al.*, 2015)

1: Input initial policy π_θ parameters θ, Q-function $Q_\phi(s,a)$ parameters ϕ and an empty replay buffer $\mathscr{D}$
2: Initialize target parameters equal to main parameters $\theta_{target} \leftarrow \theta, \phi_{target} \leftarrow \phi$
3: **repeat**
4: Gather state s and select action $a = clip(\mu_\theta(s) + \varepsilon, a_{Low})$, where $\varepsilon \sim \mathscr{N}$
5: Perform action a in the environment
6: Gather next state s', reward r, and done signal d that represents whether s' occurs to be a terminal state
7: Insert the tuple (s,a,r,s',d) into replay buffer $\mathscr{D}$
8: **if** $s'$is terminal **then** Reset environment state
9: **end if**
10: **if** update condition is fulfilled **then**
11: **for** many updates **do**
12: Randomly sample a a batch of transitions, $\mathscr{B} = \{(s,a,r,s',d)\}$ from $\mathscr{D}$
13: Compute targets

$$y(r,s',d) = r + \gamma(1-d)Q_{\phi_{target}}(s', \mu_{\theta_{target}}(s'))$$

14: Update Q-function by executing one step of gradient descent

$$\Delta_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_\phi(s,a) - y(r,s',d))^2$$

15: Update policy by executing one step of gradient ascend

$$\Delta_\phi \frac{1}{|B|} \sum_{(s) \in B} Q_\phi(s, \mu_\theta(s))$$

16: Update target network parameters

$$\phi_{target} \leftarrow \rho \phi_{target} + (1-\rho)\phi$$

$$\theta_{target} \leftarrow \rho \theta_{target} + (1-\rho)\theta$$

17: **end for**
18: **end if**
19: **until** convergence

2.6 Final Remarks

This chapter presented many concepts related to Recommender Systems including traditional approaches, recent techniques, common evaluation methods and other topics

relevant to this research. Moreover, we presented key features and drawbacks of each approach, featuring traditional and recent challenges. Also, we briefly reviewed the fundamentals of Reinforcement Learning techniques that have an important role in this research work.

Further, in Chapter 3, there is a review of recent works about the usage of Recurrent Neural Networks for Recommender Systems, describing how they are applied. In addition next chapter also presents some of the research challenges related to the area of session-based recommender systems.

CHAPTER

3

RELATED WORKS

After having reviewed several theoretical concepts about Recommender Systems and Reinforcement Learning techniques, in this Chapter we gathered several works that are closely related with this research.

In the realm of session-based recommendation we have several approaches ranging from k-nearest neighbors (JANNACH; LUDEWIG, 2017) to Deep Learning techniques such as the early introduced *GRU4REC* (HIDASI *et al.*, 2015)(HIDASI; KARATZOGLOU, 2017)(TAN; XU; LIU, 2016). Furthermore, the literature also reports approaches that include past sessions' information (QUADRANA *et al.*, 2017) (VASSØY *et al.*, 2019)(RUOCCO; SKREDE; LANGSETH, 2017) and attention based models (LI *et al.*, 2017). Deep RL techniques for recommendation have been applied to session-based scenarios mostly in the e-commerce domain for pairwise ranking (ZHAO *et al.*, 2018b), long-term user engagement (ZOU *et al.*, 2019), page-wise optimization (ZHAO *et al.*, 2018a) (LEI; LI, 2019). We divided this chapter in three sections session-based RS (Section 3.1), session-aware RS (Section 3.2) and Deep RL techniques (Section 3.3).

3.1 Session-based recommender systems

Session information plays an important role in many domains such as e-commerce where most of the transactions are preformed by non-logged users. Given this scenario, modeling user's sequential interactions has great influence into recommendation tasks. One of the earliest works to introduce RNNs in an attempt to solve this problematic was (HIDASI *et al.*, 2015)'s *GRU4REC* that adopted the next item to be clicked task for anonymous sessions using pairwise objective functions that were based on classical ranking techniques. In a subsequent work called *GRU4RECv2* (HIDASI; KARATZOGLOU, 2017), the authors improved several aspects such as negative sampling and the introduction of better ranking based objective functions resulting in significant improvements. Other ap-

proaches have used additional techniques such as data augmentation, a common practice when training deep learning models (TAN; XU; LIU, 2016), multimodal information (HIDASI *et al.*, 2016), attention mechanisms (LI *et al.*, 2017) where the authors introduced an encoder-decoder architecture to model global and local user intent for sessions pointing out the great impact of the sequential encoding of interactions on recommendation performance, contextual signals (i.e. device type, dwell time) (SMIRNOVA; VASILE, 2017) where the authors found out that modeling contextual information can bring benefits when predicting for long sessions. And, matrix factorization based embeddings for sessions (TWARDOWSKI, 2016) in which the authors combined sequential encoding of item's contextual information with item factorized representations concluding that the difference in availability of contextual information in datasets affects the final performance. Despite the remarkable results of RNN-based models, in a study (JANNACH; LUDEWIG, 2017) the authors showed that nearest neighbors based methods could have similar or even better results with less computational power.

Most of these approaches solely rely on item-to-item co-occurrence which carries several limitations and it's not suitable for the wide range of existing application domains. These limitations range from the heterogeneity of data attributes to the constant changes of user preferences which affect performance over time (WANG; CAO; WANG, 2019). Additionally, these approaches do not address other problems like users' changes of interest or real time user feedback (e.g. positive/negative interactions such as *click*, *remove-from-cart*) explicitly during sessions having all their prediction power supported by sequence modeling (LUDEWIG; JANNACH, 2018).

3.2 Session-aware recommender systems

These approaches main objective is to build representations of past sessions and use them to improve performance of fresh new ones. In some domains where user past information is available, it might be useful to model past interactions within the current session. This motivated (QUADRANA *et al.*, 2017) to introduce *HGRU4REC*, a hierarchical RNN model based on *GRU4REC* (HIDASI *et al.*, 2015) which can deal with past users' sessions information in session-based scenarios also called session-aware approaches. This architecture consists in two separated GRU networks to model sequences at two levels, user and session, so they can model inter and intra session dynamics by generating the initial state of the the next session using the user network. Being a similar architecture in (RUOCCO; SKREDE; LANGSETH, 2017) the authors dealt with the dynamics of user short-term interests and cold-start scenarios by modeling certain number of previous sessions to then initialize further users' sessions and concluded that modeling the time-difference between sessions may bring improvements. Later in (VASSØY *et al.*, 2019) based on (RUOCCO; SKREDE; LANGSETH, 2017), the authors added a Point Process

model to predict the returning time of the user in a future session.

In addition to the limitations presented in Section 3.1, in these techniques, real time interactions of inter-session and intra-session representations are yet to be explored which could be beneficial in a long sequence of interactions scenario (LUDEWIG; JANNACH, 2018). Detecting these changes does not have a trivial solution due to the chaotic nature of user behavior during sessions.

3.3 Deep Reinforcement Learning

Recently, due to their long-term goal orientation, Deep Reinforcement Learning techniques have been explored in the field of recommendations mostly in the e-commerce domain (ZHAO *et al.*, 2019) and specially for the next item prediction task (ZHAO *et al.*, 2018b)(ZOU *et al.*, 2019) and proved that they can outperform previous approaches. However, these new techniques require several adaptations for the recommendation problem and have several other new challenges such as high variance of resultant models and constantly changing preferences of users. especially to the sequential prediction task, due to their goal orientation and long-term optimization. For instance, in (ZHAO *et al.*, 2018b) the authors pointed out the potential of dealing with the imbalanced proportion between positive and negative feedback and introduced a novel framework for pairwise ranking that combines them in a Deep Q-Network model. In (ZHAO *et al.*, 2018a) the authors proposed a framework for generating and re-ranking 2-D pages of items using real-time user feedback. Furthermore, in (ZOU *et al.*, 2019) the authors introduced a framework that uses several kinds of user feedback signals to model instant and long-term user engagement. In (CHEN *et al.*, 2018) the authors presented strategies to deal with the dynamic nature of real-world online recommendation for training RL models. Most of these previous works have used RNN to encode interactions.

3.4 Final Remarks

In this chapter we have reviewed several recent works in RS pointing out several limitations and different approaches for session-based recommendation. We have seen that session-based RS techniques prediction power mostly rely on sequence modeling tasks. Also, these techniques do not exploit other sources of information which entail several limitations. Deep RL techniques were used for the session-based scenario due to their long term optimization objective and showed that can outperform several baselines despite of the new challenges and adaptations that have to be made for this area. Additionally, they DRL approaches do not treat internal aspects of a session such as users' changes of interests, which could be beneficial.

Session-aware approaches only use past sessions representations to initialize a new session to improve their predictions, however, the longer the sessions the less useful this initialization becomes because users tend to change their short-term interests. We believe that we can exploit inter-session information during long sessions by adjusting the current session state every certain number of interactions towards better results. For this end, in this work we apply Deep RL techniques to find a policy that can suit the adjustment of the current state of the session including inter-session dynamics. In the following chapter, we will present our solution proposal in detail.

CHAPTER

4

PROPOSAL

In previous chapters we reviewed theoretical concepts about traditional and session-based recommender systems. Additionally, we described the main contributions of several works that have strong relevance to our proposal. In this chapter we will present our proposal describing its components and all the stages of design.

The main objective of our proposal is to explore ways to improve recommendation during users' long sessions by adjusting the current state of the intra-session network by introducing inter-session dynamics information. In order to deal with this problem we established the following research questions: 1) what strategies should our proposed agent adopt to perform its actions?, 2) how should we build the observations that we use to train our agent?, 3) How well does our proposal performs only for the long sessions scenario? and finally 4) How well does our proposal performs in the overall recommendation scenario .

This Chapter is organized as follows: in Section 4.1 we describe our proposal in detail, including the list of adaptations made to fit the settings of the Reinforcement Learning Problem. Further, in Section 4.2 we cover all the strategies that we use to introduce signals generated by our model. Furthermore, in Section 4.3 we describe the architectural aspects of each component of our proposal. Moreover, in Section 4.4 we present our training method. Finally, in Section 4.5 we present this Chapter's final remarks.

4.1 Reinforced HGRU4REC

Our proposal consists of an external agent that is fed with the current internal states of a pre-trained instance of our baseline model *HGRU4REC* (QUADRANA *et al.*, 2017), that we chose to take advantage of the Parallel Minibatches training method and include the improvements presented in (HIDASI; KARATZOGLOU, 2017) in order

to improve recommendation. Due to the complexity of the action search space and the continuous nature of the hidden states of the networks of the environment, our agent is trained under the Actor-Critic schema and specifically we chose the Deep Deterministic Policy Gradient (LILLICRAP *et al.*, 2015) algorithm due to its ability to work in a continuous space of actions.

The main idea is to take observations from the environment to introduce inter-session dynamics information to the current session state towards dealing with concept drifts during long sessions. In the Figure 8 we show a simplified version of the components of our proposal and the interactions between them that we will review in detail in Section 4.1.1. The environment component (b) is composed of two GRU cells, GRU^{user} and GRU^{sess}, which respectively output the hidden states H_t^{user} and H_t^{sess}. These states are concatenated in (c), in order to generate the observation vector O_t or state S_t, which is fed to the Agent (e) that generates an action A_{t+1}. This action is performed on the environment in (f) according to the strategies described in Section 4.2 which updates the internal state of GRU_{sess}. With this new state we calculate the next item y_t which is evaluated in (d) and the reward R_{t+1} that is used to perform training. All the observations are gathered every k session steps. Additionally, in the Figure 9, we depict the internal structure of our agent Actor an Critic components, the signals R and S coming from the environment to output the next action. Furthermore, we can see the estimation of the value of the state or $Q-value$ that will be used to perform the optimization of the Actor, which acts as our policy.

4.1.1 Deep Deterministic Policy Gradient Adaptations

In order to fit our purpose we made several adaptations of the original DDPG algorithm defining several entities to train our agent as we can see in the list below.

- **Environment:** A pre-trained instance of an *HGRU4REC* model $\mathscr{E}$ that uses the first 80% of each user's sessions. From this model, we gather the internal states, H_t^{user} and H_t^{sess}, to train our agent.
- **State (S_t):** This is the set of observations of our environment that we gather every k intra-session steps are the concatenations of current hidden states:
  - Initial session hidden state(H_{init}^{sess}) or the state with which each new user session is initialized.
  - Current session hidden state (H^{sess}) or the resulting internal state of the session network after a forward pass through the intra-session component.
  - Current user predicted state (H_{pred}^{user}) or resulting state generated after a forward pass of the last hidden session state through the inter-session component.

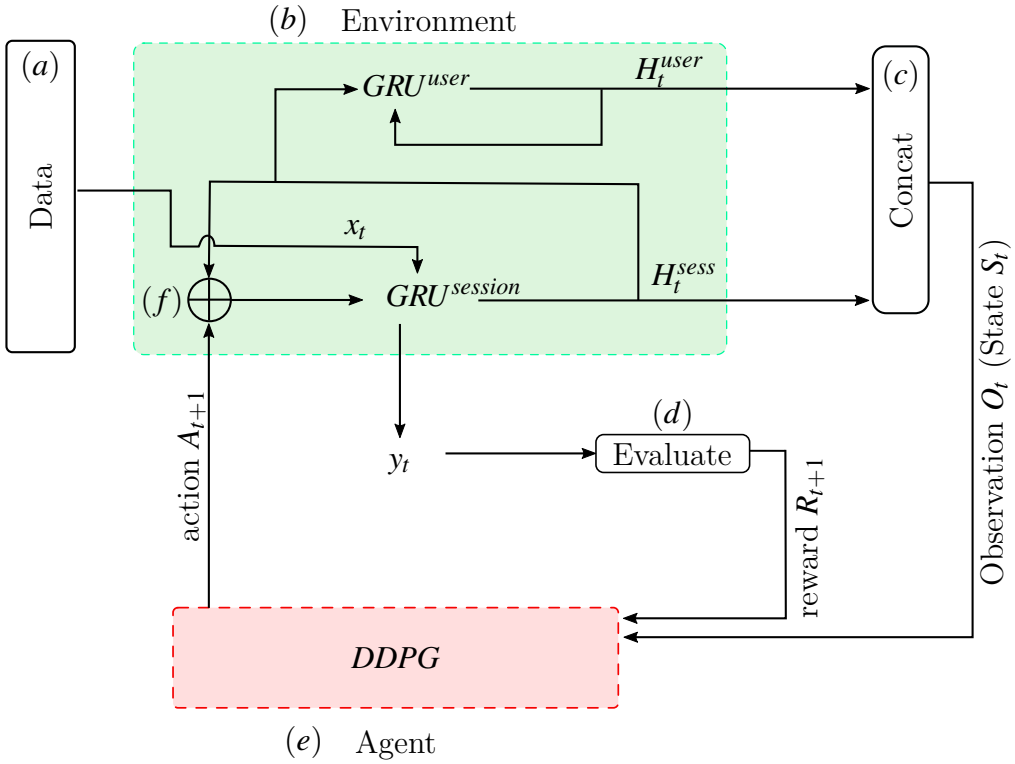


Figure 8 – Joint model dataflow for the Reinforced HGRU4REC

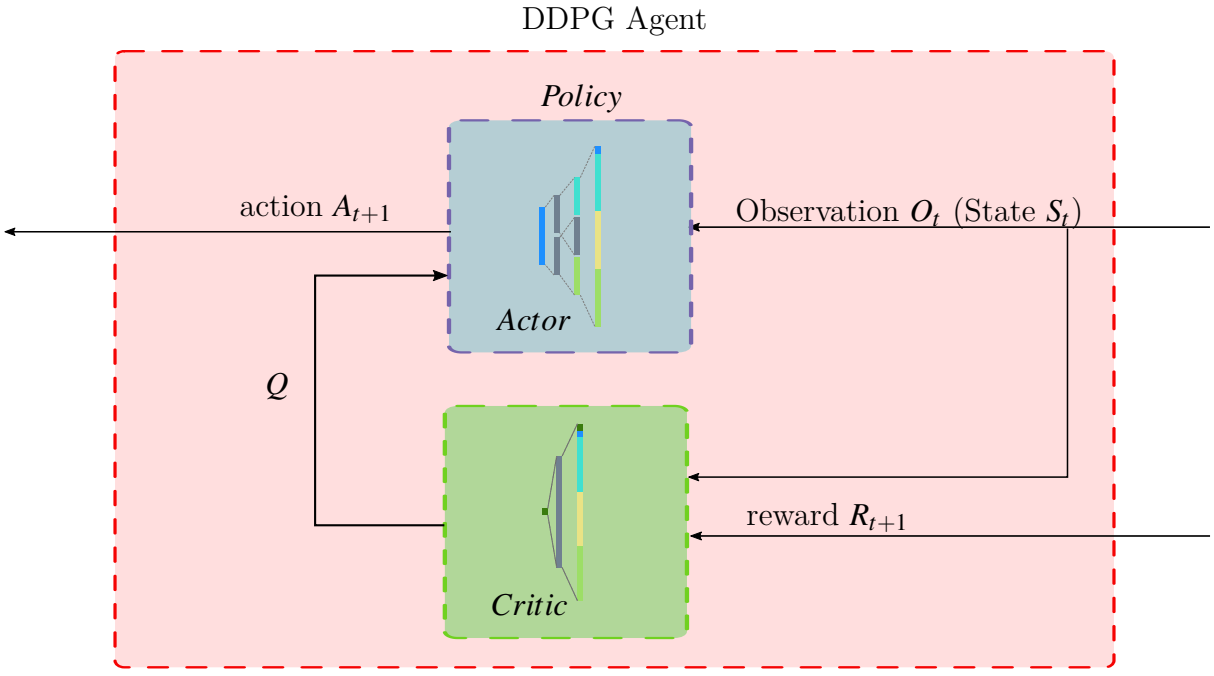


Figure 9 – Internal structure of the DDPG Agent

– Session step number($step\_number$) indicates how many steps have been performed during the current session.

These observations are gathered from the environment every $steps_{min}^{sess}$.

- **Action(A_t):** The agent outputs an action vector a which will be introduced to the current hidden state of H^{sess} with few strategies that we review in further Sections.

- **Reward(R_t):** Because we are trying to increase accuracy, we use the ratio between the number of correctly predicted instances instances and the maximum length of the sequence after performing the action during a session.

- **Done condition:** This is necessary to train our critic network by filtering out valid trajectories. We define it as whether the session has finished during the processing of intra session steps between observations.

4.2 Strategies

In our session-aware setting performing an action is not a trivial task, so to explore how we can perform the action generated by our agent we designed two strategies that we named **gate** and **noise**. Both of them are intended to introduce perturbations to the current hidden state of the intra-session network H_t^{sess}. Additionally, we introduce perturbations only for selected events according to the Equation 5.

- **Noise:** This strategy generates a noise vector that represents by applying the activation function *tanh* to the output of our agent. The result of this operation is added directly to the current hidden state of the session-level network H_{new}^{sess} to generate H_{new}^{sess} as in Equation 4.1.

$$H_{new}^{sess} = H^{sess} + action \tag{4.1}$$

- **Gate:** This strategy generates a vector with values in the interval $[0:1]$ by applying the activation function *sigmoid* to the output of our agent in order to filter out features of the current hidden state of the session-level network H^{sess}. Then the new state of the session-level network H_{new}^{sess} will be the product of the $gate_{mask}$ and current state H^{sess} plus the product of the predicted state coming from the user-level network and the complement to 1 of $gate_{mask}$ as in Equation 4.2.

$$H_{new}^{sess} = H^{sess} * action + H_{pred}^{user} * (1.0 - action) \tag{4.2}$$

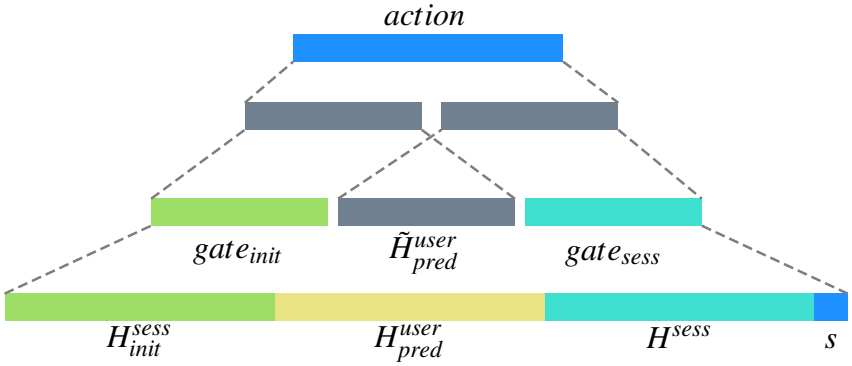


Figure 10 – Actor architecture

4.3 Architecture

According to the DDPG algorithm we need two sub-networks: an actor that generates the actions that are performed on the environment and a critic that will predict the estimated reward of the current state. Both networks are described in following sections.

4.3.1 Actor

We designed the actor following the intuition of filtering information from the current state of the network by means of predicting an action that exploits the inter-session dynamics of every user during the current session. For this end, we used several gating mechanisms for each part of our observation vectors presented in Equations 4.3 that filters out information from the current state os the session-level network, 4.4 encodes and filters out information from the initialization hidden state of the current session and 4.5 encodes and filters out information from the result of propagating the current hidden state of the session-level network through the user-level network.

$$gate_{sess} = \sigma(W_{sess} * [H^{sess}, s] + b_{sess}) \tag{4.3}$$

$$gate_{init} = \sigma(W_{init} * [H^{sess}_{init}, s] + b_{init}) \tag{4.4}$$

$$\tilde{H}^{user}_{pred} = \tanh(W_{pred} * \left[H^{user}_{pred}, s\right] + b_{pred}) \tag{4.5}$$

$$out = \left[gate_{sess} * \tilde{H}^{user}_{pred}, gate_{init} * \tilde{H}^{user}_{pred}\right] \tag{4.6}$$

$$Action_{noise} = tanh(W_{out} * out + b_{out}) \tag{4.7}$$

$$Action_{mask} = \sigma(W_{out} * out + b_{out}) \tag{4.8}$$

Additionally, we present the final activation functions in the Equations 4.7 and 4.8 according to each of the strategies previously described in Section 4.2. In the Figure 10, we can see how the observation vector is forward propagated through our actor to generate the action based on the equations presented above.

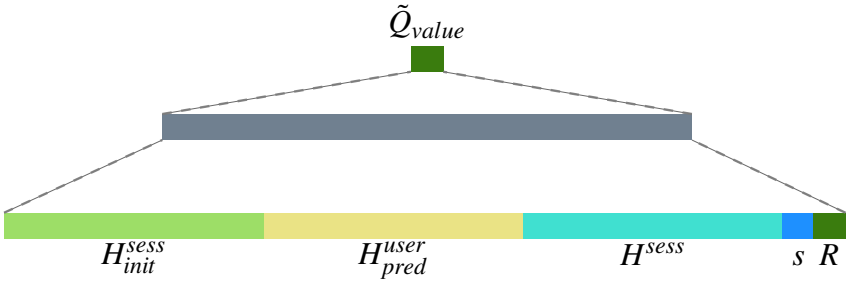


Figure 11 – Actor architecture

4.3.2 Critic

We used a multilayer perceptron applying the ReLU activation on each layer as in the Equation 4.9 for the input, Equation 4.10 for the hidden layer and Equation 4.11 for the output layer. This network outputs the predicted reward for the resulting state after performing the action. In Figure 11 we can see the organization of layers for this network.

$$input = RelU(W_{in} * [state, reward] + b_{in}) \tag{4.9}$$

$$inner = RelU(W_{inner} * input + b_{inner}) \tag{4.10}$$

$$\tilde{Q}_{value} = \sigma(W_{value} * inner + b_{value}) \tag{4.11}$$

4.4 Training method

To train our agent we first generate the final hidden representation of each user sessions for their first 80% of session by forwarding the data through the environment. Then with these final states we start to train our agent on unseen data for the environment which corresponds to the remaining 20% of the sessions of each user following our intra-session events' selection criteria defined in Section 4.4.1.

The training process consists on taking observations every $step^{sess}_{min}$ session steps to generate and perform an action. Then, we calculate the accumulated reward for each session sequence for the remaining $step^{sess}_{min} - 1$ session steps. We collect the accumulated reward value, state and next-state to generate transitions that are stored in a replay buffer. Then, we perform the optimization of our policy based on stored transitions.

The full training method of the agent is done by the Algorithm 2 where we can see all the adaptations over the original DDPG algorithm. To perform our training we use three auxiliary functions *build_state* (Algorithm 3) to generate the current state of the environment, *generate_valid_mask* (Algorithm 4) to select valid session interactions, *handle_action*(5) to perform the action generated by the policy and *evaluate_reward* to calculate the reward values.

Algorithm 2 – Reinforced HGRU4REC

Require: Trained instance of HGRU4REC with parameters γ set as environment $\mathscr{E}$, User Parallel Minibatches $\mathscr{P}$

Initialize empty Replay buffer $\mathscr{D}$
Initialize randomized parameters θ for the policy π^{θ} and ϕ for Q-network Q^{ϕ}
Initialize H^{sess},H^{user},H^{sess}_{init} with zeros
$step_{num} \leftarrow 1$
$count_{sess} \leftarrow 1$
$i \leftarrow 0$
$\mathscr{R} \leftarrow 0$
while $i \leq |\mathscr{P}|$ **do**
    Build state from observation of the environment
    $state \leftarrow build\_state(\mathscr{E}, H^{sess}, H^{user}, H^{sess}_{init}, step_{num})$
    Feed the current policy π^{θ} with state to generate the action:
    $action \leftarrow \pi^{\theta}(state)$
    Add noise generated from the random process $\mathscr{N}$ to the *action* for exploration
    $action \leftarrow action + \mathscr{N}^{a}$
    Generate mask for valid intra-session interactions :
    $mask^{valid} \leftarrow generate\_valid\_mask(step_{num}, count_{sess})$
    Perform the action a according to the current strategy for the valid interactions:
    $H^{sess}_{new} \leftarrow handle\_action(\mathscr{E}, H^{sess}, action, mask^{valid}, strategy)$
    $\mathscr{E}_{out}, H^{sess}, H^{user}, H^{sess}_{init} \leftarrow \mathscr{E}(\mathscr{P}[i], H^{sess}_{new}, H^{user})$
    $step_{num} \leftarrow step_{num} + 1$
    $\mathscr{R} \leftarrow \mathscr{R} + calculate\_reward(\mathscr{E}_{out}, steps_{num}, mask^{valid})$
    Calculate accumulated rewards for the next $step_{min}$ steps.
    for $k \leftarrow 1$ to $steps^{sess}_{min} - 1$ **do**
        $\mathscr{E}_{out}, H^{sess}, H^{user}, H^{sess}_{init}, step_{num} \leftarrow \mathscr{E}(\mathscr{P}[k+i], H^{sess}_{new}, H^{user})$
        $step_{num} \leftarrow step_{num} + 1$
        $\mathscr{R} \leftarrow \mathscr{R} + calculate\_reward(\mathscr{E}_{out}, steps_{num}, mask^{valid})$
        $i \leftarrow i + 1$
    end for
    Push valid transitions into the $\mathscr{D}$
    if $\mathscr{D}$'s size $\geq$ *policy_batch_size* **then**
        Sample a batch $\mathscr{B}$ of transitions from $\mathscr{D}$
        Perform update of policy π^{θ} and Q-network Q^{ϕ} feeding the batch $\mathscr{B}$
    end if
end while

4.4.1 Intra-session valid events selection

According to our objective of treating long sessions we only feed our agent with intra session events. However, it is very hard to know when to take observations from the environment because of the chaotic nature of user interactions. For this end, we heuristically defined two additional parameters $step^{sess}_{min}$ and $count^{sess}_{min}$ that represent the minimum number of steps performed during the current session and the minimum number of user sessions respectively. To include these conditions, we created a mask as shown in Equation 4.12. With these conditions we intend to have enough cross session information

and treat later events during sessions. We introduced this mask following the intuition of methods presented in Section 2.4.4.2, so we create a mask for every user-parallel minibatch in which each row represents a session. We included this mask in the training method of the agent by applying the Algorithm 5.

$$mask^{valid} = (step^{sess} \geq step^{sess}_{min}) \wedge (count^{sess} \geq count^{sess}_{min}) \tag{4.12}$$

After selecting valid interactions we store them in a replay buffer $\mathscr{D}$ which we use to train our agent.

Algorithm 3 – Build State Function

procedure $build\_state(\mathscr{E}, H^{sess}, H^{user}, H^{sess}_{init}, step_{num})$
  $H^{user}_{pred} \leftarrow GRU^{user}_{\mathscr{E}}(H^{sess})$
  $state \leftarrow [H^{sess}_{init}, H^{user}_{pred}, H^{sess}, step_{num}]$ ▷ Concatenation of internal states
  return $state$
end procedure

Algorithm 4 – Generate Valid Mask Function

procedure $generate\_valid\_mask(step_{sess}, count_{sess}, step^{min}_{sess}, count^{min}_{sess})$
  $mask^{valid} \leftarrow (step^{sess} \geq step^{sess}_{min}) \wedge (count^{sess} \geq count^{sess}_{min}$
  return $mask^{valid}$
end procedure

Algorithm 5 – Handle Action function

procedure $handle\_action(\mathscr{E}, H^{sess}, action, mask^{valid}, strategy)$
  if strategy = "gate" **then**
    $H^{user}_{pred} \leftarrow GRU^{user}_{\mathscr{E}}(H^{sess})$
    $H^{sess}_{new} \leftarrow H^{sess} * action * mask^{valid} + H^{user}_{pred} * (1.0 - action) * mask^{valid}$
  end if
  if strategy = "noise" **then**
    $H^{sess}_{new} \leftarrow H^{sess} + (action * mask^{valid})$
  end if
  return H^{sess}_{new}
end procedure

4.5 Final Remarks

In this Chapter we described in detail our proposal *Reinforced-HRGU4REC*, including its main architectural components, a list of adaptations of the original *HGRU4REC* for the RL problem setup, strategies for performing actions on the environment and training method. In following chapters we will describe our experiments, show our results and discuss our findings.

CHAPTER

5

EXPERIMENTS

In previous chapters we reviewed several concepts that are relevant to this research, such as recommender systems, machine learning and reinforcement learning. Also, we presented a detailed description of our approach. In this chapter we describe the adopted methodology to evaluate our approach, the datasets and more importantly our results and discussion.

This chapter is organized as follows in Section 5.1 we present our experiments' design including a description of the datasets and selection of hyper-parameters. In Section 5.2 we review an analysis of the components of our approach, reward function and our results quantitatively. Finally, in Section 5.3 we close this chapter with our final considerations.

5.1 Experiment Design

This section presents in detail the design of experiments including our databases, baselines, selection of hyper-parameters and the evaluation methods used in this work.

5.1.1 Datasets

In order to perform our experiments we gathered two publicly available datasets that are suitable for session-based recommendations. In this section we present a detailed review of the data and preprocessing stage.

5.1.1.1 Tianchi Repeat Buyers Challenge

On e-commerce platforms merchants offer deals in order to attract new buyers but this strategy does not have a long term impact on sales because some buyers are one-time

Table 1 – Datasets' statistics

Dataset	Tianchi	30Music
Users	19992	37667
Items	49682	232265
Sessions	238555	1213248
Events	2293121	15259198
Events per item ⋆	46.16/28.00/116.60	65.70/30.00/148.80
Events per session ⋆	9.61/6.00/16.72	12.58/8.00/17.71
Sessions per user ⋆	11.93/9.00/9.08	32.21/22.00/29.61
Training events	1482164	12120840
Training sessions	218050	1175616
RL-Training sessions	30998	212265
RL-Training events	366167	2667615
Test events	441933	470743
Test sessions	20398	37667

⋆ mean/median/std

deal hunters. This dataset was released[1] with the objective of identifying which buyers might buy from the same merchant in the next six months. All the data were collected from the website Tmall.com containing behavior logs of anonymized users.

5.1.1.2 30Music listening and playlists dataset

This is a dataset for music recommendation that was released[2] to cope with task such as user modeling and playlist generation. It contains historical user listening events gathered from Internet radio stations by using the Last.fm API. The listening events are organized into user listening sessions in a chronological order. This dataset also contains several additional data such as tracks' tags, album's info and users' explicit feedback records("love" events). Roughly, it is composed of 31M listening events, 45K users, and 5.6M tracks.

For our experiments and both datasets, we considered users with ≥ 5 sessions that have ≥ 3 interactions per session and items with support ≥ 20 to avoid cold start issues, we chose to leave repeated events because we wanted to simulate an online setup. To perform the training of our agent we split the data following the protocol *last-session-out* defined in Section 2.4.4.2, then we split the resultant training partition again under the 80/20 ratio for the total amount of sessions of each user. So, we used the 80% of user sessions to train the environment and trained the agent with the remaining 20%. In Table 1 we present statistical information about our datasets after having processed them.

[1] <https://tianchi.aliyun.com/competition/entrance/231576/information>
[2] <http://recsys.deib.polimi.it/datasets/>

5.1.2 Baselines

We selected relevant session based models and session aware the following models to compare against our results:

- **GRU4RECv2:** This is the second version of the original GRU4REC (HIDASI *et al.*, 2015) with the improvements presented in (HIDASI; KARATZOGLOU, 2017).
- **HGRU4REC:** This the original Theano[3] implementation of (QUADRANA *et al.*, 2017) trained with the new TOP1Max loss function presented in (HIDASI; KARATZOGLOU, 2017).
- **pyHGRU4REC:** This is our implementation of HGRU4REC using the Pytorch framework. This implementation for simplicity does not include momentum in its optimizer unlike the original implementation of (QUADRANA *et al.*, 2017).
- **Environment:** This is an instance of HGRU4REC that we used to perform the training of our policy. This model is trained only on the first 80% of each user's sessions. This will serve as point of reference for the improvements that our policy could bring.

It is also important to mention that previous RL recommendation approaches are not suitable for comparison with our methods because they do not treat the users' changes of interests during a session.

5.1.3 Hyperparameter selection

For our experiments we defined a delimited range of values for each hyper-parameter, we divided them in two groups as follows:

- **Environment Hyperparameters:** We performed training with learning rates from the set $[0.05, 0.10, 0.15, 0.2]$ with dropout probabilities for both levels of the network in the range $[0.0, 0.1, 0.2, 0.3]$ we tried batch sizes of length [100]. To carry on the process of optimization we used the TOP1Max loss function from (HIDASI; KARATZOGLOU, 2017) with an Adagrad optimizer. After performing 20 trials of Bayesian Optimization(AKIBA *et al.*, 2019) of parameters we found the following configurations in Table 2. We performed training only for small networks with 100 layers of stacked RNNs that correspond to the 100 positions of the batch, in which each position represents a single user.

[3] <http://deeplearning.net/software/theano/>

Env. Param.	Tianchi	30Music
drop_prob_init	0.1	0.0
drop_prob_session	0.1	0.1
drop_prob_user	0.1	0.3
learning_rate	0.1	0.05

Table 2 – Environment training parameters

Agent Param.	Tianchi	30Music
gamma	0.95	0.9
soft tau	10^{-2}	10^{-3}
batch size	100	100
policy-net lr.	10^{-3}	10^{-7}
value-net lr.	10^{-4}	10^{-6}

Table 3 – Agent training parameters

- **DDPG Agent Hyperparameters:** For both networks actor and critic we used learning rates from the set $\left[10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}\right]$, batch sizes $[100, 200]$ with an Adam optimizer and mean Squared Error Loss. Also, a discount factor γ from the set $[0.9, 0.95, 0.99]$, soft update coefficient τ from the values $\left[5 \times 10^{-4}, 10^{-3}\right]$. After having performed 5 trials of Bayesian Optimization(AKIBA *et al.*, 2019) of parameters we obtained the final configurations shown in Table 3.

  The decaying factor gamma(γ) has great importance into the value estimation for the state, so we made previous test with smaller values $[0.1, 0.3, 0.5]$ as in other previous works, but found no difference in the results.

- **Observation constraints:** In order to only treat observations according to the rules defined in Section 4.1 we chose $step_{min}^{sess} = 5$ to have enough valid transitions for each iteration over our training dataset and $count_{min}^{sess} = 3$ to ensure that the user has enough inter-session information. After having finished every iteration over the training data we performed a maximum of 30 iterations over the whole replay buffer. On every following iteration over the training data we emptied the replay buffer.

5.1.4 Evaluations methods

To evaluate our agent we chose two metrics *Recallk* and *Mean Reciprocal Rankk* for the top-k predicted items where we chose $k = 5$ as in (QUADRANA *et al.*, 2017). Additionally, we carried out statistical significance tests for our models using Wilcoxon signed rank method as in (QUADRANA *et al.*, 2017).

5.1.5 Implementation details

To run our experiments we used a computer with a processor Intel Core i7-6800k@3.4GHz×12 with 16GB RAM and GPU Nvidia Titan Xp with 12GB VRAM. Our models were developed using the Pytorch[4] framework and all the implementation source code was publicly released on github[5].

5.2 Results

In this section we will present the results of our experiments with a detailed discussion of our Findings. We divided this section according to the objectives of this research works. Consequently, we fist look into the global behavior of our models then a fine grained analysis for each of our research objectives.

5.2.1 Overall Performance

In Table 4 we present a summary of global results for our proposed strategies alongside our baselines. We performed 10 training epochs for every model additionally we used five different values of weight initialization seeds which had not a considerable impact on the final result. All the values presented in Table 4 are the averages of the results for different seeds and are plot in Figures 12 and 13 for the 30M and Tianchi datasets respectively.

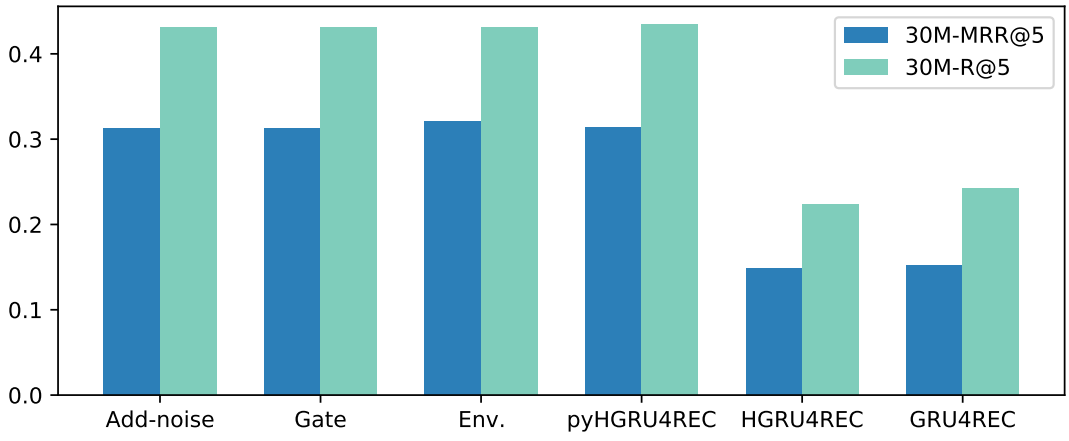


Figure 12 – Global results for the 30M dataset

For the 30M dataset our best performing models are pyHGRU4REC with a substantial improvement that almost doubles for both metrics MRR5 and R5 over the original implementation of HGRU4REC and GRU4REC. Surprisingly, for the MRR%5 the

[4] <https://www.pytorch.com>
[5] <https://github.com/gescobedo/ddpg-hgru4rec>

	Tianchi		30Music	
Model	**MRR@5**	**R@5**	**MRR@5**	**R@5**
Add-noise	0.131658	0.152801	<u>0.312971</u>	<u>0.430907</u>
Gate	<u>0.134775</u>	<u>0.155493</u>	0.312539	0.430617
Environment	0.136757	0.156822	**0.320875**	0.430959
pyHGRU4REC	0.165956	0.198200	0.313227	**0.433953**
HGRU4REC	**0.178551**	**0.205724**	0.148119	0.223193
GRU4REC	0.136350	0.184860	0.152142	0.241810

Table 4 – **Overall Performance for all models**. The values in **bold** are the best performing models, the <u>underlined</u> values represent the best performing DRL strategy

Environment model outperforms all models with statistical difference for $p < 0.05$ over pyHGRU4REC even though it was trained with less data. Both DRL strategies have competitive global performance with a drop drop of 1% with no statistical difference between them for $p < 0.05$ against pyHGRU4REC.

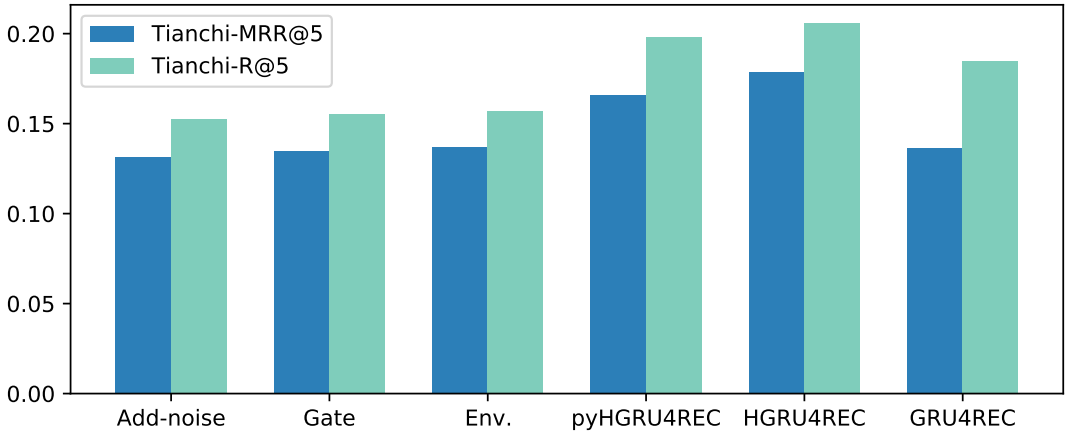


Figure 13 – Global results for the Tianchi dataset

For the Tianchi dataset our best performing model is the HGRU4REC which outperforms our implementation pyHGRU4REC in MRR@5 and R@5 in approximately 7%. Our DRL strategies showed a significant performance drop of approximately 23%, even lower thant GRU4REC.

For both datasets we have seen different performance for our strategies and implementations that might be attributed to the difference between their information domains. Furthermore, the proportions between the amount of interactions for training and test for our agent show an important difference which might affect the final results. Also, the proportion of data used to trained the environment, which could smaller than 80% which will give more data to the agent to learn from. Additionally, the Tianchi dataset could be biased by seasonality because it was gathered in previous days to the ”11.11”, a day when users are hunting for price deals, and are just browsing to figure out what

	Strategy	30M MRR@5	30M R@5	Tianchi MRR@5	Tianchi R@5
H_{init}^{sess}	add-noise	0.312663	0.430560	0.136399	0.156359
	gate	0.312580	0.430621	0.136636	0.156496
H^{sess}	add-noise	0.312533	0.430333	0.136154	0.156150
	gate	0.312567	0.430628	0.136622	0.156459
$step_{num}^{sess}$	add-noise	0.312747	0.430672	0.136574	0.156570
	gate	0.312579	0.430597	0.136608	0.156430

Table 5 – Impact of each architectural component for DRL strategies

items could actually are good deals during this seasonal phenomena. This portraits one of the common problems when analyzing not independently and identically distributed data (WANG; CAO; WANG, 2019). Additionally, this shows that the *last-session-out* splitting protocol has some limitations for evaluating global performance.

5.2.1.1 Analysis of architectural components

Our proposal uses the internal components of the original model as stated in Section 4.1, so we need to analyze how each of these components impact on the final result. In this section we present a detailed revision and analysis of each component. To perform this component analysis we turned off each of the components by setting the internal gates generated of our Actor sub-network, so we grouped the results according to the component that we are taking out from the prediction of the actor. In Table 5 we can see that for these extra components that we are including into the Actor sub-network do not have a great impact into performance which implies that the $\tilde{H}_{cand}^{user}$ component carries the most information for the generation of the actions.

5.2.2 Influence of session length

In order to analyze the degree of impact of our model on long sessions, we split the dataset in several slices according to defined intervals. We chose an interval length of 20 to have a fine grained vision of the data.

5.2.2.1 Global session performance

In the Figure 15, we can see that for the *30Music* dataset our proposed strategies get similar results to pyHGRU4REC for each of the slices despite of having trained our environment models with only 80% percent of the data. However, it performs slightly worse than the Environment model For the *Tianchi* dataset in Figure 14, we can see a significant drop in performance which could be due to the different user intent of the session in a

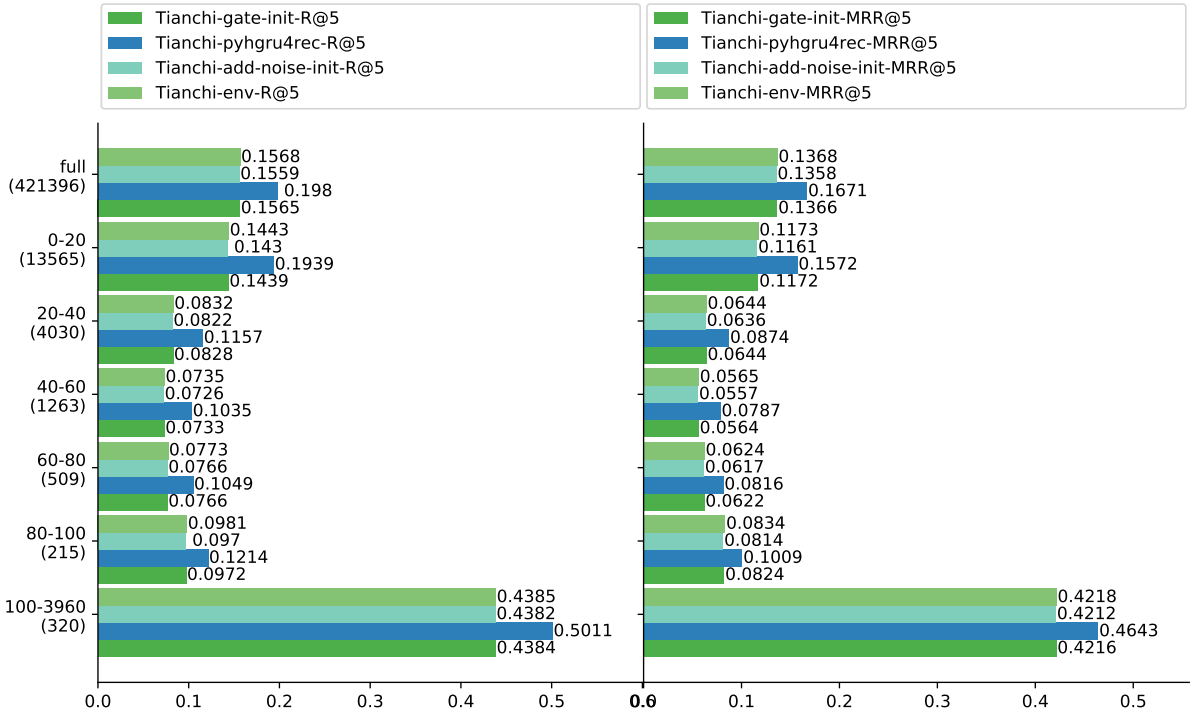


Figure 14 – The Y axis represents groups of users with average session length in the interval low-high(frequency of users), the X axis represents the value of the metric for each group of users

different domain (i.e. diversity of consumed items) and the unbalanced proportion of interactions generated by the applied data splitting protocol.

5.2.2.2 *Recurrent user session length*

Additionally, we grouped users according to the average length of their sessions to see the impact on users that tend to have regular session lengths and how they can benefit from inter-session dynamics information. We split the dataset in groups of interactions of length 10 for our intervals. In Figure 16 we show the results for each slice of the 30M dataset. We can observe our strategies *gate* and *noise* outperform our baselines for users with session average length in the interval $[20, 30]$ and $[30, 40]$ with improvements of 2-10% in *R@5* and 1-5% in *MRR@5*. However, it slightly outperforms the Environment model for the interval with average session length in the interval $[20, 30]$. On the other hand, for the Tianchi dataset we got a steady proportion between the baseline and our strategies with a drop of 18-30% for the most significant portion of users shown in Figure 17.

We also wanted to see whether or not for coarser intervals we might have different results so we sliced the datasets again this time with intervals of length 5 for the average session length of users. In the Figures 18 and 19 we show the results for top 6 smallest slices of our datasets. For the 30M dataset we can see that there is a significant improvement for users with recurrent short-length sessions. This means that for this group of users the

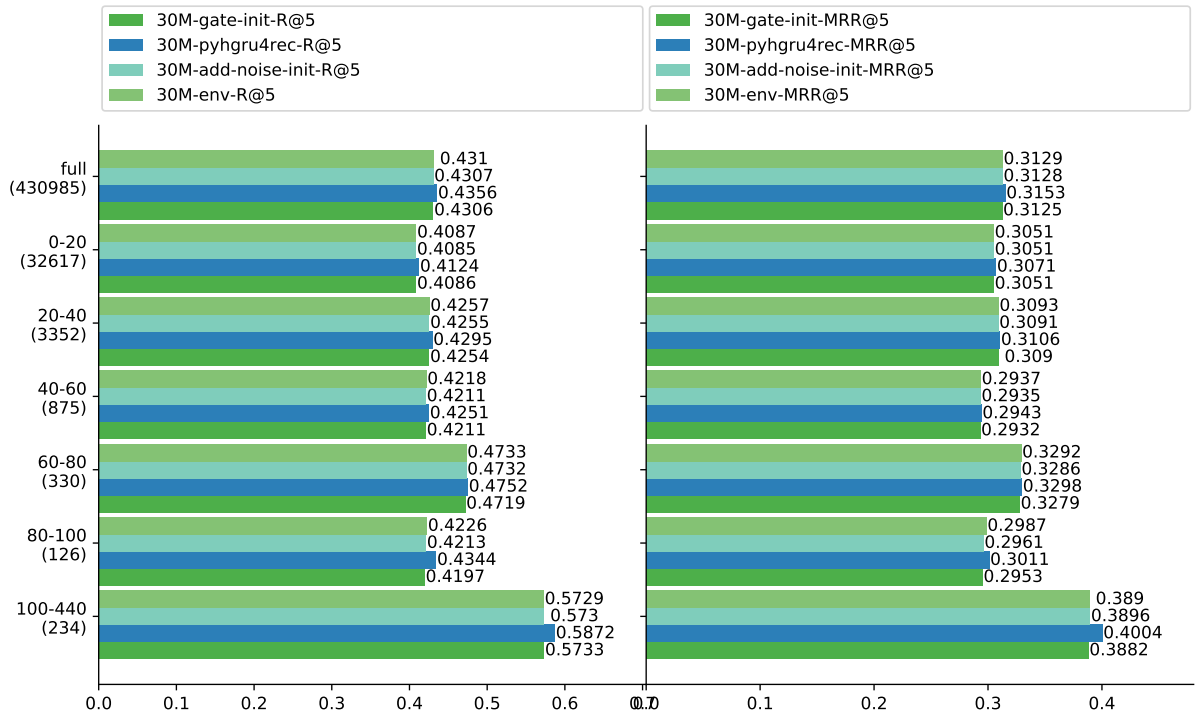


Figure 15 – Impact on recommendation according to session length for the 30M dataset. the Y axis represents groups of sessions with length in the interval low-high(frequency of sessions), the X axis represents the value of the metric for each group of users

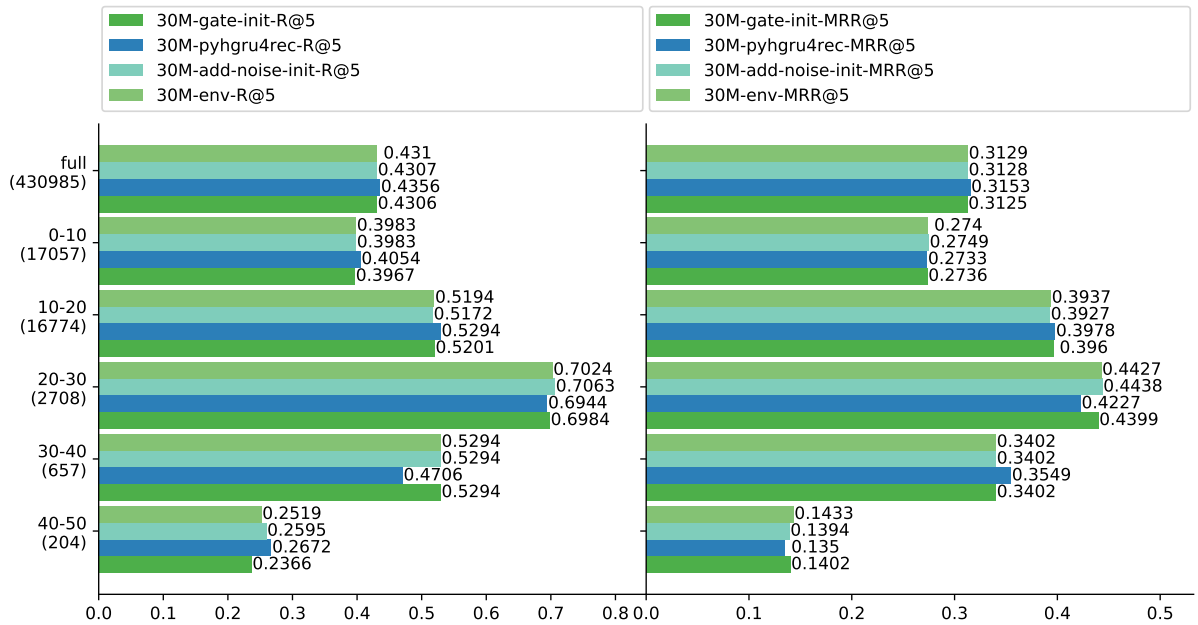


Figure 16 – Impact on recommendation according to average session length for the 30M dataset with equally sized intervals of length 10. On the Y axis we have the size of slice ”low-high(frequency of users)” on the X axis we have the value for the metric

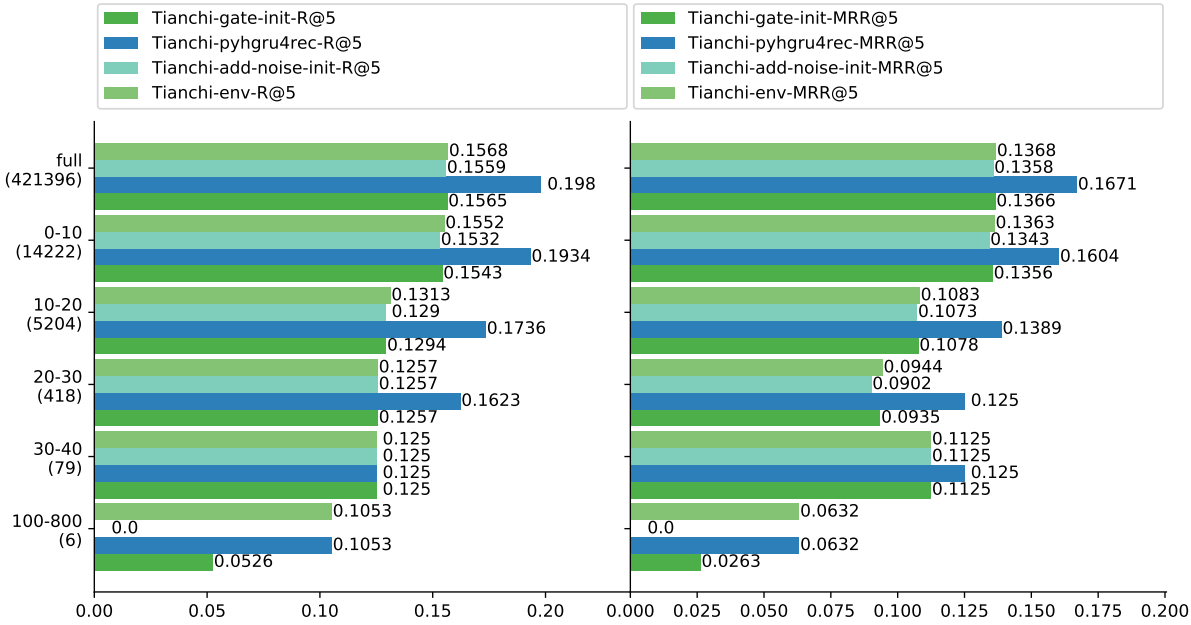


Figure 17 – Impact on recommendation according to average session length for the Tianchi dataset with equally sized intervals of length 10. On the Y axis we have the size of slice "low-high(frequency of users)" on the X axis we have the value for the metric

introduction of inter-session information acts as regularizer, so the resultant state at the end of each session carries a better representation than our baselines which improves the initialization of future sessions. However, these improvements only affect a minor portion of the users, as we can see a similar behavior for the Tianchi dataset in Figure 18 on the slice $[20, 30]$ with only 100 users.

5.2.3 Influence of user history length

One important part of our analysis is the degree of personalization on the long term, so we also divided our dataset into slices according to user interaction history length. To perform this analysis we used slices of length 2000 and 1000 for the 30M and Tianchi datasets respectively to see how our models behave through time. For the 30M dataset we see no significant difference between our strategies and the baselines. Furthermore, for the Tianchi dataset we can see that the majority of the users that have less than 1000 interactions, which has a great impact to global performance for all the models.

From the Figures 20 and 21 we can see a clear difference of distributions being the biggest groups of users the ones with total of interactions in the $[0, 2000]$ range for the 30M dataset and $[0, 1000]$ for the Tianchi dataset. For the 30M dataset all our models show steadily similar performance for all the slices that have a considerable number of users. Also we can see a growing tendency in performance in groups with larger size of history length which indicates the importance of long history of interactions.

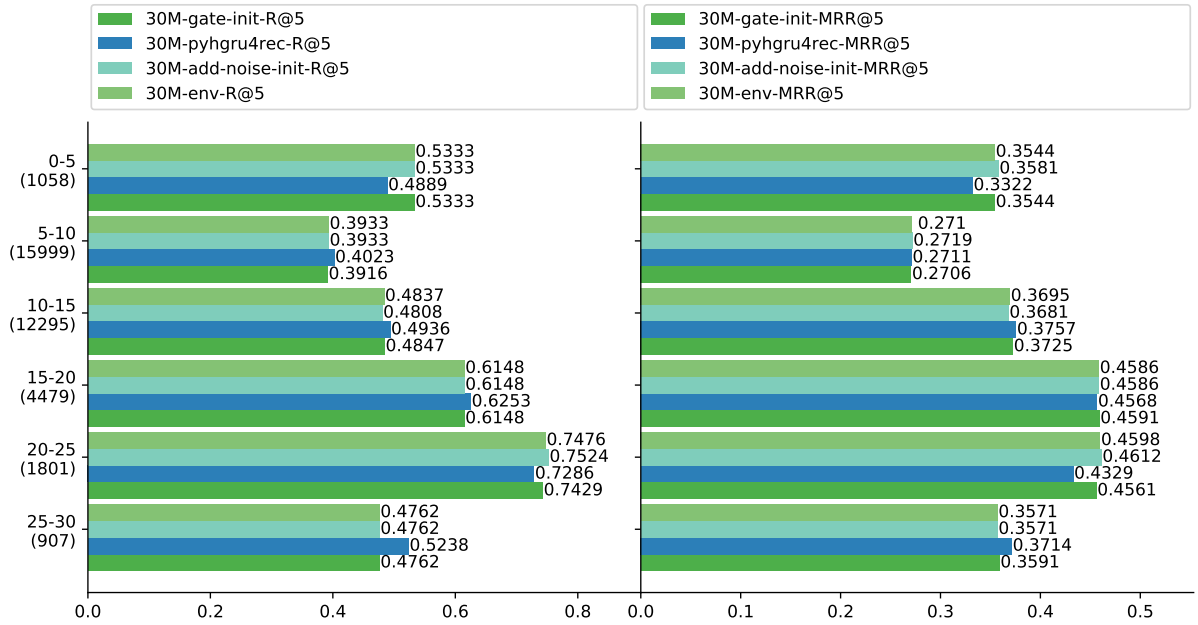


Figure 18 – Impact on recommendation according to average session length for the 30M dataset with equally sized intervals of length 5. On the Y axis we have the size of slice "low-high(frequency of users)" on the X axis we have the value for the metric

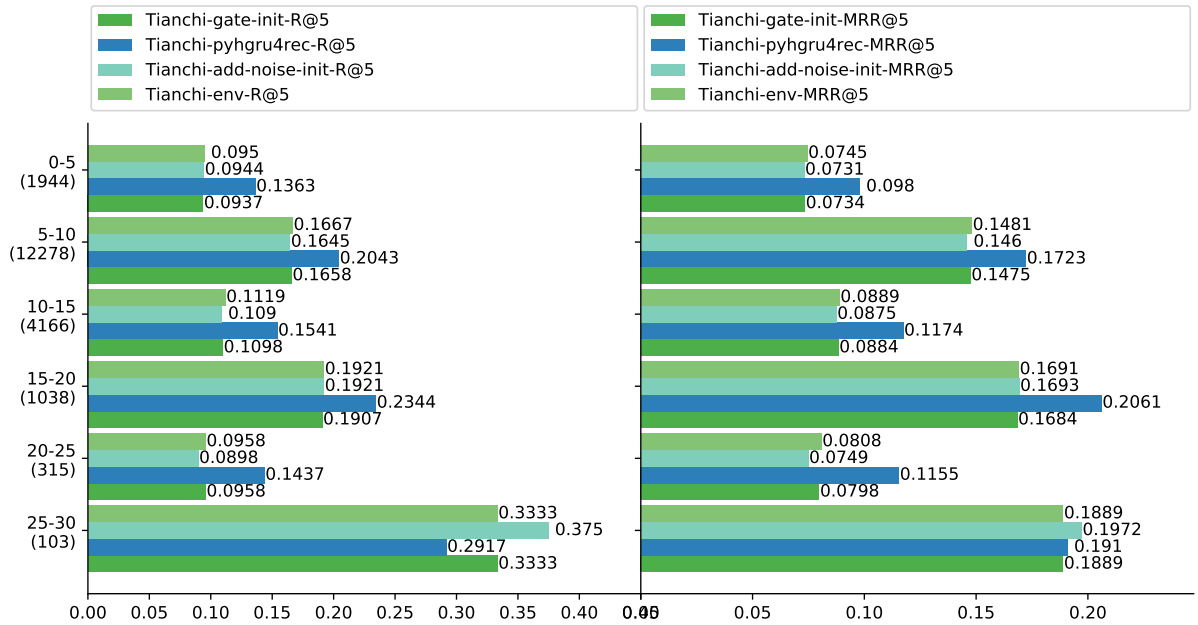


Figure 19 – Impact on recommendation according to average session length for the Tianchi dataset with equally sized intervals of length 5. On the Y axis we have the size of slice "low-high(frequency of users)" on the X axis we have the value for the metric

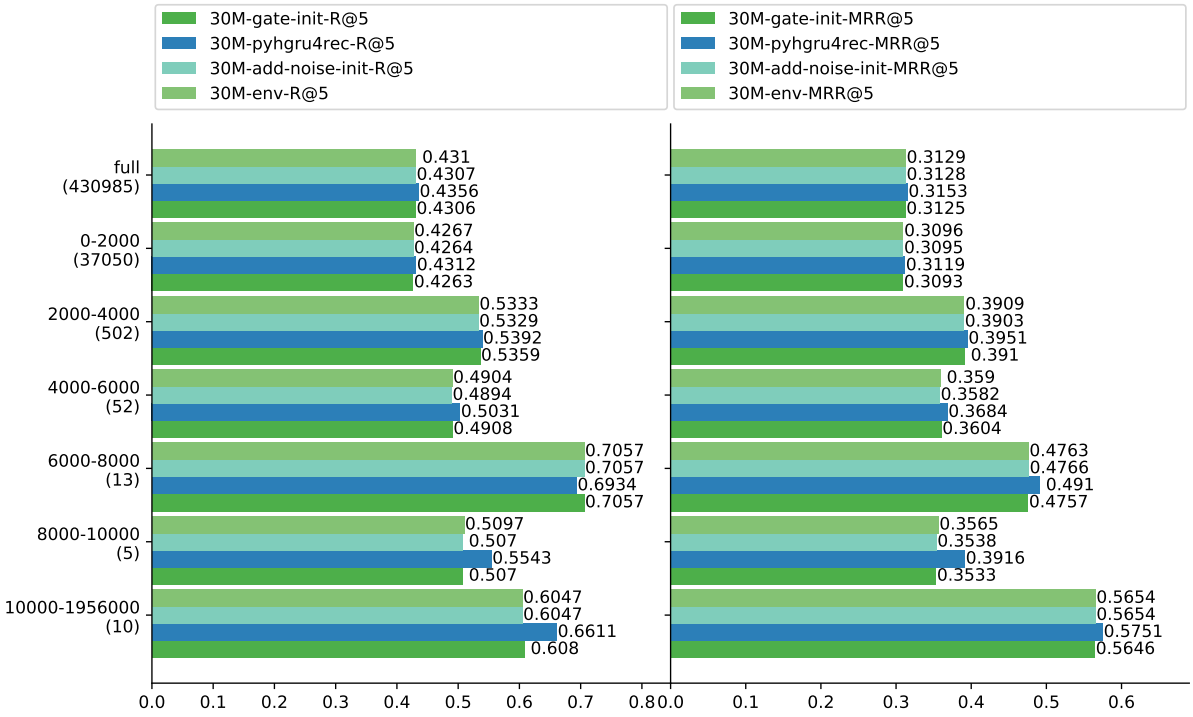


Figure 20 – Impact on recommendation according the total user history length for the 30M dataset. Y axis represent groups of users with a history length in the interval low-high(frequency of users), the X axis represents the value of the metric.

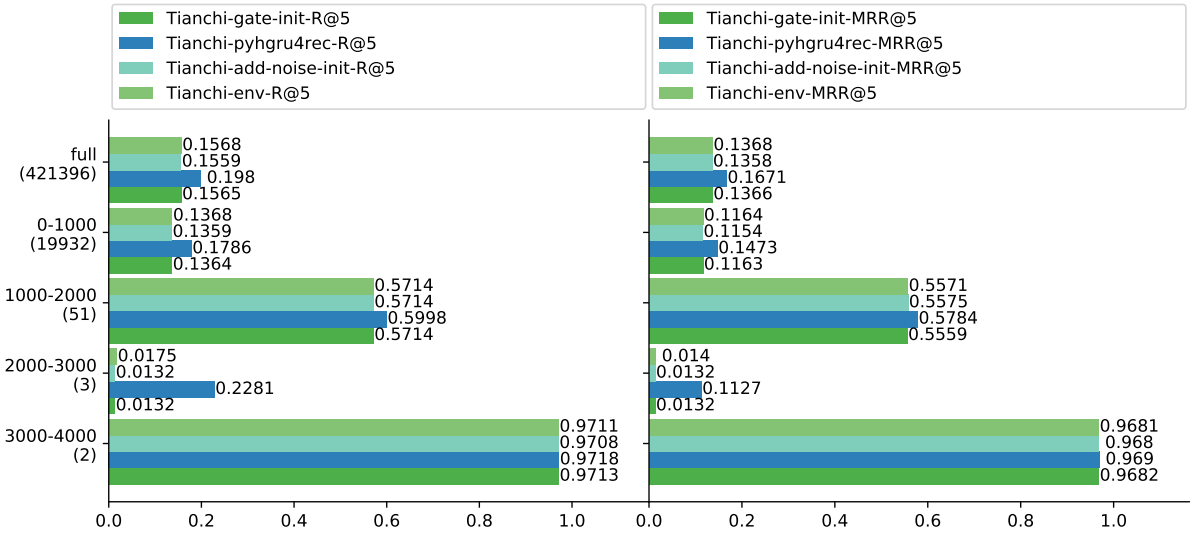


Figure 21 – Impact on recommendation according the total user history length for the Tianchi dataset. The Y axis represent groups of users with a history length in the interval low-high(frequency of users), the X axis represents the value of the metric.

5.3 Final Remarks

In this chapter we described all the methodological aspects of our experiments and analyzed our results in detail according to our objectives.

In the first part we briefly described the datasets used and how we processed them to perform training showing some statistical information. Furthermore, we presented a list of our baselines that we compared against our models. To perform training, we presented a detailed list of how we chose hyper-parameters to train all our models and their final configurations.

In the second part of this Chapter we presented our results that showed that our strategies *Add-noise* and *Gate* in general have similar performance to the baseline models and that show better results for some users that have a certain average length of sessions. Additionally, we observed that our strategies outperform the baseline for users with session average of ≤ 5, which means that through time we enrich the generated inter-session state for users' new sessions. However, these improvements cannot indicate general improvements because they only affect a small portions of users. All the tests performed provided a wide vision of the behavior of a session-aware recommender and the importance of sufficient historical user data.

Finally, in the following Chapter we will present a summary of this research work, establish our conclusions and indicate possible lines of improvement.

CHAPTER

6

CONCLUSIONS AND FUTURE WORK

In this Chapter we present a brief summary of this research work in Section 6.1, a list our main contributions in Section 6.2 and conclusions in Section 6.3. Finally, we will describe possible lines of improvements for this research in Section 6.4.

6.1 Summary

In this research work we applied Deep RL techniques to address the changes of short-term interests in long sequences of session interactions for session-aware recommendation by exploiting inter-session dynamics information based on the current states of a Hierarchical RNN. We presented several adaptations to the DDPG algorithm to train a policy for the session-aware recommendation setting and dealt with how to gather observations from the environment without being an exhaustive task and the strategies to perform the actions generated by our policy. Additionally, we analyzed how our policy could improve overall performance and specifically for long sequences of session interactions.

In our proposal we presented the algorithm *Reinforced-HGRU4REC* to train our policy based on observations from the original *HGRU4REC* (QUADRANA *et al.*, 2017). Our algorithm consists of a custom method to gather observations and two main strategies *Gate* and *Add-noise* that we used to perform the action on the environment. Additionally we presented the architecture of the customized Actor and Critic sub-networks to train under the DDPG algorithm. We performed several experiments over two datasets from different domains and compared with a set of strong baselines. Then we evaluated all our models following several data slicing criteria to analyze the impact of performance in long sequences of interactions. We obtained competitive results for both our strategies against our baseline models that were trained over full datasets despite of our environment models were trained using only 80% of the data, this could be attributed to the data splitting

protocol adopted. Through an extensive testing we analyzed in depth our models an also found out situations in which session-aware models struggle and observed the importance of the users' history length. Furthermore, we faced the issues of adopting the RL setting to the recommendation task, which is not a straight forward process.

6.2 Contributions

As a result of this research work we provided the main contributions:

- We provided an adaptation of the DDPG algorithm over the HGRU4REC model including a method for gathering observations and performing action with two different strategies. This adaptation includes a training method and strategy to gather enough data

- We provided an extensive analysis of the behavior of session-aware recommenders under different settings alongside our proposal

6.3 Conclusions

In this research work we extensively explored the possibility of exploiting inter-session dynamics information to improve intra-session predictions for long sequences of interactions by applying Deep Reinforcement Learning techniques. We presented an adaptation of the Deep Deterministic Policy Gradient algorithm to train a policy that can enrich the current state of the session network with inter-session dynamics information.

We proposed two different strategies for introducing the actions into session state being the noise-based one the best performing. We obtained competitive results only using a portion of the training data compared to the baselines that were trained with full datasets. Furthermore, we performed several analysis toward having a wide vision of the models' behavior. During our test we could see that the proposed strategies can have competitive results but they require a lot of tweaking in their training process.

We conclude that there is not strong evidence to claim that inter-session dynamics information can improve the performance for the global recommendation task even when the agent performs on unseen data. However, through an extensive analysis of results, we showed that the length of historical user data has a great impact in session-aware approaches.

6.4 Future Work

For our future works we can consider several other non-RNN based session-aware recommenders to test if our module can learn from those architectures such as (TANG; WANG, 2018).

One possible line of improvement is to try other architectures to better encode the current session context. Additionally attention mechanisms can be added to encode the sequences of observation, as applied in (LI *et al.*, 2017), that we are using to train our policy .

Side information could play an important role when gathering observations from the environment as in traditional RS bringing a better understanding of the current session context without having to exhaustively processing all its interactions(SMIRNOVA; VASILE, 2017)(TWARDOWSKI, 2016), improving the heuristics for event selection for the agent.

To some extend there is a huge variety of Deep Reinforcement Learning algorithms that have not been explored for the recommendation task. Especially, for learning models towards to qualitative aspects of recommendations such as long-term engagement (ZOU *et al.*, 2019).

BIBLIOGRAPHY

ACHIAM, J. **Kinds of Algorithms**. 2018. Accessed on: 22/10/2020. Available: <https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html>. Citation on page 46.

ADOMAVICIUS, G.; TUZHILIN, A. Context-aware recommender systems. In: **Recommender Systems Handbook**. Boston, MA: Springer US, 2011. p. 217–253. ISBN 978-0-387-85820-3. Available: <https://doi.org/10.1007/978-0-387-85820-3_7>. Citation on page 27.

AGGARWAL, C. C. **Recommender Systems: The Textbook**. 1st. ed. New York City, USA: Springer Publishing Company, Incorporated, 2016. ISBN 3319296574. Citations on pages 27, 34, 35, and 36.

AKIBA, T.; SANO, S.; YANASE, T.; OHTA, T.; KOYAMA, M. Optuna: A next-generation hyperparameter optimization framework. In: **Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining**. New York, NY, USA: Association for Computing Machinery, 2019. (KDD '19), p. 2623–2631. ISBN 9781450362016. Available: <https://doi.org/10.1145/3292500.3330701>. Citations on pages 67 and 68.

BAEZA-YATES, R.; RIBEIRO-NETO, B. **Modern Information Retrieval: The Concepts and Technology Behind Search**. 2nd. ed. USA: Addison-Wesley Publishing Company, 2008. ISBN 9780321416919. Citations on pages 27 and 36.

CHEN, S.-Y.; YU, Y.; DA, Q.; TAN, J.; HUANG, H.-K.; TANG, H.-H. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In: **Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining**. New York, NY, USA: Association for Computing Machinery, 2018. (KDD '18), p. 1187–1196. ISBN 9781450355520. Available: <https://doi.org/10.1145/3219819.3220122>. Citation on page 53.

CHUNG, J.; GÜLÇEHRE, Ç.; CHO, K.; BENGIO, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. **CoRR**, abs/1412.3555, 2014. Available: <http://arxiv.org/abs/1412.3555>. Citation on page 40.

DING, Z.; HUANG, Y.; YUAN, H.; DONG, H. Introduction to reinforcement learning. In: DONG ZIHAN DING, S. Z. H. (Ed.). **Deep Reinforcement Learning: Fundamentals, Research, and Applications**. [S.l.]: Springer Nature, 2020. chap. 2, p. 47–124. <http://www.deepreinforcementlearningbook.org>. Citations on pages 43, 44, and 45.

DONKERS, T.; LOEPP, B.; ZIEGLER, J. Sequential user-based recurrent neural network recommendations. In: **Proceedings of the Eleventh ACM Conference on Recommender Systems**. New York, NY, USA: ACM, 2017. (RecSys '17), p. 152–160. ISBN 978-1-4503-4652-8. Available: <http://doi.acm.org/10.1145/3109859.3109877>. Citation on page 39.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge, MA: The MIT Press, 2016. ISBN 0262035618. Citation on page 38.

HIDASI, B.; KARATZOGLOU, A. Recurrent neural networks with top-k gains for session-based recommendations. **arXiv preprint arXiv:1706.03847**, 2017. Citations on pages 51, 55, and 67.

HIDASI, B.; KARATZOGLOU, A.; BALTRUNAS, L.; TIKK, D. Session-based recommendations with recurrent neural networks. **arXiv preprint arXiv:1511.06939**, 2015. Citations on pages 29, 42, 51, 52, and 67.

HIDASI, B.; KARATZOGLOU, A.; SAR-SHALOM, O.; DIELEMAN, S.; SHAPIRA, B.; TIKK, D. Dlrs 2017: Second workshop on deep learning for recommender systems. In: **Proceedings of the Eleventh ACM Conference on Recommender Systems**. New York, NY, USA: ACM, 2017. (RecSys '17), p. 370–371. ISBN 978-1-4503-4652-8. Available: <http://doi.acm.org/10.1145/3109859.3109953>. Citation on page 28.

HIDASI, B.; QUADRANA, M.; KARATZOGLOU, A.; TIKK, D. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In: **Proceedings of the 10th ACM Conference on Recommender Systems**. New York, NY, USA: ACM, 2016. (RecSys '16), p. 241–248. ISBN 978-1-4503-4035-9. Available: <http://doi.acm.org/10.1145/2959100.2959167>. Citation on page 52.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Comput.**, MIT Press, Cambridge, MA, USA, v. 9, n. 8, p. 1735–1780, Nov. 1997. ISSN 0899-7667. Available: <http://dx.doi.org/10.1162/neco.1997.9.8.1735>. Citation on page 39.

JANNACH, D.; LUDEWIG, M. When recurrent neural networks meet the neighborhood for session-based recommendation. In: **Proceedings of the Eleventh ACM Conference on Recommender Systems**. New York, NY, USA: Association for Computing Machinery, 2017. (RecSys '17), p. 306–310. ISBN 9781450346528. Available: <https://doi.org/10.1145/3109859.3109872>. Citations on pages 51 and 52.

KOREN, Y. Collaborative filtering with temporal dynamics. **Commun. ACM**, ACM, New York, NY, USA, v. 53, n. 4, p. 89–97, Apr. 2010. ISSN 0001-0782. Available: <http://doi.acm.org/10.1145/1721654.1721677>. Citation on page 27.

LEI, Y.; LI, W. Interactive recommendation with user-specific deep reinforcement learning. **ACM Trans. Knowl. Discov. Data**, Association for Computing Machinery, New York, NY, USA, v. 13, n. 6, Oct. 2019. ISSN 1556-4681. Available: <https://doi.org/10.1145/3359554>. Citation on page 51.

LI, J.; REN, P.; CHEN, Z.; REN, Z.; LIAN, T.; MA, J. Neural attentive session-based recommendation. In: **Proceedings of the 2017 ACM on Conference on Information and Knowledge Management**. New York, NY, USA: ACM, 2017. (CIKM '17), p. 1419–1428. ISBN 978-1-4503-4918-5. Available: <http://doi.acm.org/10.1145/3132847.3132926>. Citations on pages 51, 52, and 81.

LILLICRAP, T. P.; HUNT, J. J.; PRITZEL, A.; HEESS, N.; EREZ, T.; TASSA, Y.; SILVER, D.; WIERSTRA, D. Continuous control with deep reinforcement learning. **arXiv preprint arXiv:1509.02971**, 2015. Citations on pages 17, 47, 48, and 56.

LUDEWIG, M.; JANNACH, D. Evaluation of session-based recommendation algorithms. **User Modeling and User-Adapted Interaction**, Springer, v. 28, n. 4-5, p. 331–390, 2018. Citations on pages 29, 37, 41, 52, and 53.

MANNING, C. D.; RAGHAVAN, P.; SCHüTZE, H. **Introduction to Information Retrieval**. USA: Cambridge University Press, 2008. ISBN 0521865719. Citation on page 36.

QUADRANA, M.; CREMONESI, P.; JANNACH, D. Sequence-aware recommender systems. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 51, n. 4, Jul. 2018. ISSN 0360-0300. Available: <https://doi.org/10.1145/3190616>. Citations on pages 28 and 37.

QUADRANA, M.; KARATZOGLOU, A.; HIDASI, B.; CREMONESI, P. Personalizing session-based recommendations with hierarchical recurrent neural networks. In: **Proceedings of the Eleventh ACM Conference on Recommender Systems**. New York, NY, USA: ACM, 2017. (RecSys '17), p. 130–137. ISBN 978-1-4503-4652-8. Available: <http://doi.acm.org/10.1145/3109859.3109896>. Citations on pages 28, 29, 41, 42, 43, 51, 52, 55, 67, 68, and 79.

RUOCCO, M.; SKREDE, O. S. L.; LANGSETH, H. Inter-session modeling for session-based recommendation. In: **Proceedings of the 2Nd Workshop on Deep Learning for Recommender Systems**. New York, NY, USA: ACM, 2017. (DLRS 2017), p. 24–31. ISBN 978-1-4503-5353-3. Available: <http://doi.acm.org/10.1145/3125486.3125491>. Citations on pages 28, 29, 51, and 52.

SMIRNOVA, E.; VASILE, F. Contextual sequence modeling for recommendation with recurrent neural networks. In: **Proceedings of the 2Nd Workshop on Deep Learning for Recommender Systems**. New York, NY, USA: ACM, 2017. (DLRS 2017), p. 2–9. ISBN 978-1-4503-5353-3. Available: <http://doi.acm.org/10.1145/3125486.3125488>. Citations on pages 52 and 81.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. Cambridge, MA, USA: A Bradford Book, 2018. ISBN 0262039249. Citations on pages 43, 46, and 47.

TAN, Y. K.; XU, X.; LIU, Y. Improved recurrent neural networks for session-based recommendations. In: **Proceedings of the 1st Workshop on Deep Learning for Recommender Systems**. New York, NY, USA: Association for Computing Machinery, 2016. (DLRS 2016), p. 17–22. ISBN 9781450347952. Available: <https://doi.org/10.1145/2988450.2988452>. Citations on pages 51 and 52.

TANG, J.; WANG, K. Personalized top-n sequential recommendation via convolutional sequence embedding. In: **Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2018. (WSDM '18), p. 565–573. ISBN 9781450355810. Available: <https://doi.org/10.1145/3159652.3159656>. Citation on page 81.

TWARDOWSKI, B. Modelling contextual information in session-aware recommender systems with neural networks. In: **Proceedings of the 10th ACM Conference on Recommender Systems**. New York, NY, USA: ACM, 2016. (RecSys '16), p. 273–276. ISBN 978-1-4503-4035-9. Available: <http://doi.acm.org/10.1145/2959100.2959162>. Citations on pages 52 and 81.

VASSØY, B.; RUOCCO, M.; SILVA, E. de Souza da; AUNE, E. Time is of the essence: A joint hierarchical rnn and point process model for time and item predictions. In: **Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2019. (WSDM '19), p. 591–599. ISBN 9781450359405. Available: <https://doi.org/10.1145/3289600.3290987>. Citations on pages 28, 29, 51, and 52.

WANG, S.; CAO, L.; WANG, Y. A survey on session-based recommender systems. **CoRR**, abs/1902.04864, 2019. Available: <http://arxiv.org/abs/1902.04864>. Citations on pages 52 and 71.

WATKINS, C. J.; DAYAN, P. Q-learning. **Machine learning**, Springer, v. 8, n. 3-4, p. 279–292, 1992. Citation on page 47.

WU, C.-Y.; AHMED, A.; BEUTEL, A.; SMOLA, A. J.; JING, H. Recurrent recommender networks. In: **Proceedings of the Tenth ACM International Conference on Web Search and Data Mining**. New York, NY, USA: ACM, 2017. (WSDM '17), p. 495–503. ISBN 978-1-4503-4675-7. Available: <http://doi.acm.org/10.1145/3018661.3018689>. Citation on page 38.

ZHAO, X.; XIA, L.; TANG, J.; YIN, D. "deep reinforcement learning for search, recommendation, and online advertising: A survey" by xiangyu zhao, long xia, jiliang tang, and dawei yin with martin vesely as coordinator. **SIGWEB Newsl.**, Association for Computing Machinery, New York, NY, USA, n. Spring, Jul. 2019. ISSN 1931-1745. Available: <https://doi.org/10.1145/3320496.3320500>. Citation on page 53.

ZHAO, X.; XIA, L.; ZHANG, L.; DING, Z.; YIN, D.; TANG, J. Deep reinforcement learning for page-wise recommendations. In: **Proceedings of the 12th ACM Conference on Recommender Systems**. New York, NY, USA: Association for Computing Machinery, 2018. (RecSys '18), p. 95–103. ISBN 9781450359016. Available: <https://doi.org/10.1145/3240323.3240374>. Citations on pages 51 and 53.

ZHAO, X.; ZHANG, L.; DING, Z.; XIA, L.; TANG, J.; YIN, D. Recommendations with negative feedback via pairwise deep reinforcement learning. In: **Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining**. New York, NY, USA: Association for Computing Machinery, 2018. (KDD '18), p. 1040–1048. ISBN 9781450355520. Available: <https://doi.org/10.1145/3219819.3219886>. Citations on pages 29, 51, and 53.

ZHOU, G.-B.; WU, J.; ZHANG, C.-L.; ZHOU, Z.-H. Minimal gated unit for recurrent neural networks. **International Journal of Automation and Computing**, v. 13, n. 3, p. 226–234, Jun 2016. ISSN 1751-8520. Available: <https://doi.org/10.1007/s11633-016-1006-2>. Citations on pages 39, 40, and 41.

ZOU, L.; XIA, L.; DING, Z.; SONG, J.; LIU, W.; YIN, D. Reinforcement learning to optimize long-term user engagement in recommender systems. In: **Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining**. New York, NY, USA: Association for Computing Machinery, 2019. (KDD '19), p. 2810–2818. ISBN 9781450362016. Available: <https://doi.org/10.1145/3292500.3330668>. Citations on pages 29, 51, 53, and 81.


ICMC
USP
SÃO CARLOS