TÉCNICO
LISBOA

# Recurrent Neural Networks for Next-Action Prediction

**Rita Brito Cunha Vieira Conde**

Thesis to obtain the Master of Science Degree in

**Information Systems and Computer Engineering**

Supervisors: Prof. Bruno Emanuel da Graça Martins
Dr. Pedro Paulo Balage

**Examination Committee**

Chairperson: Prof. João António Madeiras Pereira
Supervisor: Dr. Pedro Paulo Balage
Members of the Committee: Prof. Pável Pereira Calado

**May 2021**

# Acknowledgements

I would like to thank both my advisors, Professor Bruno Martins and Dr. Pedro Balage for their insight and given support during this year. The numerous video-call meetings were undoubtedly essential for the success of this work. Last but not least, I would like to thank my family and friends for their support.

Rita Brito Cunha Vieira Conde

For my family,

# Resumo

De modo a antecipar o comportamento do utilizador num site de e-commerce, o objetivo deste trabalho é prever a próxima ação do utilizador, após uma sequência de ações realizadas numa sessão de navegação. Uma ação num contexto de e-commerce ocorre em relação a um item para compra. Pode ser, p.e. um ‘clique numa saia’, o que revela que uma ação é composta por diferentes componentes como o item, o tipo de interação e características do item, como a sua categoria. Assim, o propósito de prever a próxima ação inclui a previsão de cada uma das suas componentes e, neste trabalho, estas são consideradas como tarefas distintas. Para abordar cada uma destas tarefas, foram explorados algoritmos capazes de modelar sequências. Primeiro, modelos estatísticos, baseados no modelo de Markov, foram testados. Segundo, modelos baseados em redes neuronais recorrentes foram explorados. Adicionalmente, numa tentativa de melhoria da performance de modelos de RNN, a informação contextual, derivada a partir das relações entre as diferentes componentes da ação (p.e. o item ‘saia’ pode pertencer à categoria ‘roupa mulher’, o que é uma hierarquia entre a componente item e a componente categoria do item) foi integrada na arquitetura da rede: através duma função de custo customizada, no processo de treino da rede, e duma filtragem do catálogo de elementos a considerar, no processo de avaliação na rede. Os resultados experimentais, no dataset RetailRocket, demonstram que esta última versão de RNN consegue superar a performance dos restantes modelos, com uma melhoria de 5% nos resultados das métricas.

# Abstract

To anticipate the user behavior on an e-commerce platform, this work predicts the action a customer is most likely to take next, after a sequence of actions in a browsing session. An action in an e-commerce context commonly revolves around a shopping item. It can be, for instance, a 'click on a skirt', which entails that an action itself is composed by a set of different components, as the click event, the item ID, and the item's category. That said, in this work, the prediction of the next action is decomposed into different action component prediction tasks. To address the proposed approach, sequence-modeling algorithms were explored for each task. First, statistical models as the N-th order Markov Model were applied to the problem. Second, more sophisticated models as Recurrent Neural Networks (RNN) were explored. Furthermore, in an attempt to enhance the performance of RNN-based models, the contextual information derived from existing relations between the action's components - for instance, the item 'skirt' can belong to the category 'women clothing', which represents a taxonomic relation between the components item and category - was taken into account through a customized loss function in the training process, and a short-listing of the catalogue of elements to consider in the evaluation process. Experimental results, on the RetailRocket dataset show that RNNs using contextual information outperform any of the other tested models, with a visible increase of about 5% in the evaluation metrics scores.

# Palavras Chave
# Keywords

## Palavras Chave

- sistemas de recomendação
- modelação de sequências
- cadeia de markov
- redes neuronais recorrentes

## Keywords

- recommender systems
- sequence modeling
- markov chain
- recurrent neural networks

# Contents

# List of Figures

# List of Tables

# 1 Introduction

Recommender systems (Resnick and Varian, 1997) have been re-shaping the digital world. In the e-commerce business, these can play crucial functions, such as to predict the next query the user is about to submit on the search-box, or predict the next item(s) to view or purchase. The common end-motivation here is to facilitate the shopping experience and inherently enhance the user experience on the website.

Users' activity through an e-commerce website occurs as a sequence of actions, which can be used to learn about future interactions. The research area of recommender systems capable to model sequences of actions is known as sequence-aware recommender systems (Quadrana et al., 2018), and this work lays in this area of study.

## 1.1 Motivation

In most online businesses, the customer does not have to be logged into a user profile to navigate the website, hence, the users are frequently anonymous and activity from previous visits to the website is not reachable. Consequently, the prediction of future interactions is based on the most recent sequence of actions, recorded in the customer's browser as a browsing session. The sequence-aware recommendation techniques that learn from short-term history are called session-based recommendation approaches (Wang et al., 2019).

Most academic research of sequence-aware recommender systems in the session-based domain (Ludewig and Jannach, 2018) is centered on the next-item prediction problem, where following a sequence of items clicked in a given session, the aim is to predict the most likely item ID the user will click on next. The early stages of scientific works, included item-to-item collaborative filtering (Linden et al., 2003), that performs sequence predictions using a similarity score between the last viewed item and the remaining items available. This is a statistical approach proved to work in many e-commerce settings (Fang et al., 2019), however it ignores information from past clicks. Later on, sequence modeling approaches based on Markov Chain (MC) were introduced (Mcfee and Lanckriet, 2011) (He et al., 2009), but due to data sparsity issues, these can hardly be applied to every sequence prediction problem. Finally, over the last few years, the focus has been on developing deep learning solutions based on Recurrent Neural Networks (RNN) as presented by Hidasi et al. (2015) and Hidasi and Karatzoglou (2018). Some significant extensions to these session modeling works have been later introduced, namely a parallel-RNN (Hidasi et al., 2016), that simultaneously models the item's properties as well as the item ID, in order to help on the main prediction of the next item ID.

## 1.2 Thesis Proposal

To model and anticipate the user behaviour on an e-commerce platform, the aim in this work is to find out the action a user is most likely to take next, after a sequence of actions in an ongoing session. An action in an e-commerce context commonly evolves around a shopping item. It can be, for instance, a 'click on a skirt', which entails that an action itself is composed by multiple components, such as the click event, the item ID, and other item features, as the item's category. That being said, the purpose to predict the next action, includes the prediction of each component of the action, meaning that in the example given, the prediction of the item ID, the type of event, and the item's category, occur in separate models.

The approach taken in each model was based on the approaches used in GRU4Rec (Hidasi et al., 2015) (Hidasi and Karatzoglou, 2018). In this work, similar to GRU4Rec, the sequence modeling task is viewed as a classification problem using ranking metrics for evaluation: each action's component has a finite catalogue of distinct elements, which are considered to be the available classes to classify each user session, and the model's output will be the catalogue ranked by likelihood of each element coming up next in the given sequence. To that end, both non-deep and deep learning models were experimented. As a starting point, the applicability of simple statistical models to the problem was tested, such as with the N-th order Markov Chain, and subsequently, Recurrent Neural Networks and its variants were also explored.

Some aspects inherent to the nature of e-commerce businesses, were thought as potential limitations to the performance of the used models, being one of them the over extensive catalogue of elements for certain components of the action. That said, the existing dependencies in an item-feature perspective as well as between the different item's features - for instance, an item can belong to a certain category, meaning that there is an hierarchical relation between the item and the category feature -, were used as contextual information to narrow the set of elements to consider in the catalogue, during the training and evaluation processes of the RNN-based models. This way, the network is led to predict elements within the same family as of the true element.

Hence, in this work, it is also studied how using contextual information can impact the prediction performance of a session-based recommender system, on an e-commerce setting.

## 1.3 Contributions

Throughout this dissertation, the following contributions are presented:

- This work decomposes an action and predicts every component separately, rather than focusing solely on the prediction of the central component of an action, the item. Considering that the catalogue of possible values to consider differs immensely between the different components, this was an interesting way to study and understand which model variations bring the most significant improvements to each action's component prediction task. For each sequence prediction task, the applicability of a wide range of models is studied. Most scientific research papers in the domain already benchmark their results based on deep learning methods (Wang et al., 2019), but in this work, simple statistical models are explored as well, namely with N-th order Markov Chain (MC), and for a more sophisticated deep-learning approach, Recurrent Neural Networks (RNN) are used. The limitations of using a regular RNN layer are also reviewed in this work, and consequently, variants of the RNN, as the GRU (Cho et al., 2014) and LSTM (Hochreiter and Schmidhuber, 1997) are tested, as well as a state-of-the-art LSTM version, the multiplicative LSTM (Krause et al., 2016). To fairly assess the performance of the diverse approaches taken, the same experiments were executed for each created model. In fact, the simple statistical approach proved to perform similarly to more modern deep learning approaches. Furthermore, when testing the set of RNN-based models, a significant difference between the use of a simple RNN to other RNN versions was detected, however, between the three more evolved RNN architectures (LSTM, GRU, mLSTM) the difference was not notorious.

- The provided clickstream logs along with other details about the catalogue of items in the website, are thoroughly analysed in this work, and from the data analysis phase one could understand how the action's components relate hierarchically between each other. From this point, this taxonomic information is incorporated in any RNN-based model used, through a customized loss function, in the training process of the network, and a filtering step of the catalogue of elements to consider, in the evaluation process. This upgraded version of an RNN-based model outperformed all other tested models by 5% in the results for all ranking metrics.

- From the review of public scientific works that use the same e-commerce dataset[1], it was possible to assess that some aspects, that are fully covered in this work, are not yet explored by others (Xu et al., 2019) (Lv et al., 2019) (Sheil et al., 2018). The existing relations between the action's components are neglected, thus this contextual information is not incorporated in any RNN-based model used; Additionally, the applicability of simpler models, such as the statistical model N-th order Markov Chain, is not tested.

## 1.4 Structure of the Document

The remaining of this document is organized as follows: Chapter 2, introduces Fundamental Concepts for the understandting of this work, as well as existing work in the Sequence-Aware Recommender Systems area. Chapter 3, describes the e-commerce dataset used, and the proposed methods to address the sequence modeling problem. Chapter 4 presents the evaluation methodology, including the resources supporting the experiments and the metrics used for assessing the quality of results, and discusses the obtained results. Finally, Chapter 5 summarizes the main conclusions from this research, and presents some ideas for future work.

[1]`https://www.kaggle.com/retailrocket/ecommerce-dataset`

# 2 Concepts and Related Work

This chapter is divided into two sections. First, fundamental concepts for the fully comprehension of the document are introduced in Section 2.1. Second, related work and advancements made in the topic of sequence-aware recommender systems are discussed in Section 2.2.

## 2.1 Fundamental Concepts

This section discloses the fundamental concepts that the user will need to fully understand this document. The presented concepts are divided into two sub-sections, where both focus on well-known models for sequence prediction. First, the Markov chain and its variants are studied. Their strengths and weaknesses are pointed out, where the latter acts as the breaking point that leads the reader to the following sub-section about Neural Networks for sequence modeling. In this sub-section, Recurrent Neural Networks are explored in depth.

### 2.1.1 Markov Chain

The Markov chain was introduced by Markov (2006), and has been since a popular stochastic model. It follows the Markov property, where the prediction of the next state only depends on the current state, and not on the sequence of events that occurred previously. This property is mathematically translated in Equation (2.1). The outcome of $x_n$ is only dependent on the outcome of $x_{n-1}$:

$$\mathrm{p}(x_n|x_{n-1}, ..., x_2, x_1) = \mathrm{p}(x_n|x_{n-1}) \tag{2.1}$$

To estimate sequence distributions using a larger portion from the history of events, higher-order Markov chains consider a variable look-back window of events (Ching et al., 2004). This extension of the Markov chain is particularly accepted for language modeling problems, such as to predict the word that follows in a phrase, and is commonly referred to as the $n$-grams model. The model predicts the occurrence of an event based on the occurrence of its $n$-1 previous events in the sequence. An $n$-gram of size 1 is called an unigram, an $n$-gram of size 2 is called a bigram, where $n$ = 2 and $n$-1 = 1, which means that the model predicts the occurrence of an event based on the previous event in the sequence. For the increasing values of $n$, the same logic applies. Thus, the general equation is:

$$\mathrm{p}(x_n|x_1^{n-1}) = \mathrm{p}(x_n|x_{n-N+1}^{n-1}) \tag{2.2}$$

To estimate the $n$-gram probabilities, the Maximum Likelyhood Estimation (or MLE) (Rossi, 2018) approach is applied. To calculate a particular bigram probability of an event $x_n$ given the previous event $x_{n-1}$ , that is the conditional probability $p(x_n|x_{n-1})$, it is necessary to count all the occurrences of the sequence $x_{n-1}x_n$ and count the number of sequences that start with $x_{n-1}$ - which is the same as the count of all the occurrences of the sequence $x_{n-1}$ alone:

$$\mathrm{p}(x_n|x_{n-1}) = \frac{\mathrm{count}(x_{n-1}x_n)}{\sum_x \mathrm{count}(x_{n-1}x_n)} = \frac{\mathrm{count}(x_{n-1}x_n)}{\mathrm{count}(x_{n-1})} \tag{2.3}$$

Computing $n$-gram probabilities for longer sequences is a more intricate task, since these tend to occur rarely in a corpus and that often leads to many probabilities of zero that results in noisy predictions. To exemplify, in a fashion e-commerce context, the sequence of clicked items [trousers; skirt; shoes; shirt] could form a 4-gram, but if there are only sequences as [trousers; skirt; belt; boots; shoes; shirt] in the corpus of logs, then that 4-gram will never be caught. To tackle the problem, a technique called Smoothing emerged. A relevant version of this technique, is the Additive Smoothing (Chen and Goodman, 1999), which consists on adding one to all counts before normalizing them into probabilities, as seen on Equation (2.4) where —V— stands for the set of all items considered.

$$\mathrm{p}_{\mathrm{additive\_smoothing}}(x_n|x_{n-N+1}^{n-1}) = \frac{1 + \mathrm{count}(x_{n-1}x_n)}{|\mathrm{V}| + \mathrm{count}(x_{n-1})} \tag{2.4}$$

Hence, unseen $n$-grams will never have probabilities of zero. Although this technique corrects the data sparsity problem, it can be said that the $n$-gram model itself does not deal well with long distance dependencies.

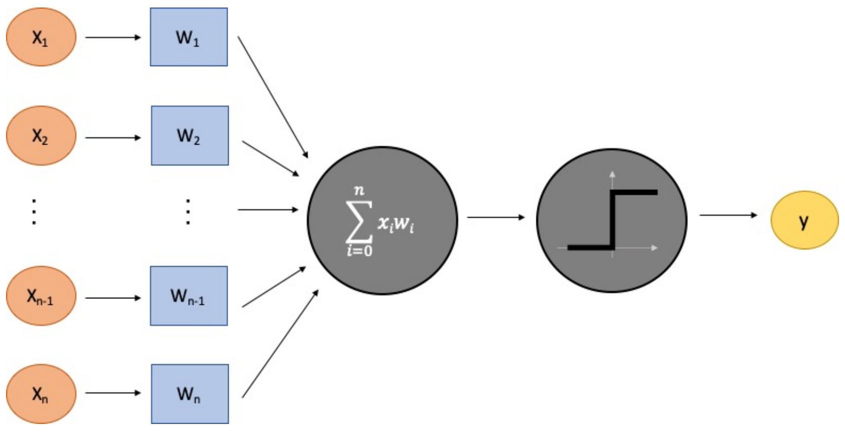


Figure 2.1: Perceptron architecture

### 2.1.2 Neural Networks for Sequence Modeling

A Neural Network (NN) (McCulloch and Pitts, 1943) attempts to mimic the behaviour of a biological brain. It is composed by a set of neurons, interconnected trough synaptic connections. The simplest model of a neural network, is the Perceptron (Rosenblatt, 1957), with only one artificial neuron.

$$\mathrm{NN}_{\mathrm{perceptron}}(\mathbf{x}) = \mathrm{g}(\mathbf{x}.\mathbf{w} + \mathbf{b}) \tag{2.5}$$

It is a linear classifier for binary predictions. All the inputs, **x**, are multiplied by their weight, **w**, - these reflect the importance of the inputs to the output. The resulting values are then passed trough an operation (usually a weighted sum), and the bias term, **b**, is added. Next, the neuron's output, 0 or 1, is determined by an activation function. The typical activation function used in a single layer NN is a binary step classifier:

$$\mathrm{f}(\mathbf{x_1}, ..., \mathbf{x_n}) = \begin{cases} 1 & \mathbf{w_0} + \mathbf{w_1 x_1} + ... + \mathbf{w_n x_n} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

As it is noticeable, the activation function (2.6) is needed to map the input into the required values 0 or 1.

On a side note, no bias is represented in Figure 2.1 for uncluttering reasons, and henceforth no biases will be represented in any figure.

A more complex NN model can be created by Multi-layer Perceptrons (MLP), which is also named as a Feedforward Neural Netowrk, since information flows in one direction, going from one layer to the other, from the leftmost layer to the rightmost layer. The MLP has an input layer, hidden layers (one or more), and output layers, which unlike in a perceptron, can have multiple neurons. This type of NN was specially designed to deal with non-linear data, where usually the activation function to be used is a non-linear function, such as sigmoid.

To train the network, the data to be fed to the network has to be annotated with the truth output, then the Back-Propagation (BP) algorithm is ready to act. Its main goal is to minimize the mean-squared distance - when using mean-square error as the loss function -, between the network's predicted values, and the target values. These errors are then propagated backwards trough the network, using the derivative of the cost function in each step. In the last phase of the algorithm, the weights and biases are readjusted, using gradient descent:

$$\Delta\mathbf{w} = -\eta \cdot \nabla \mathrm{E}(\mathbf{w}) \tag{2.7}$$

The weight update $\Delta\mathbf{w}$ is calculated by Equation (2.7), where $t$ is the target output for the given instance, and $\eta$ is the learning rate that controls how much the weights are adjusted. The gradient of E with respect to the vector $\mathbf{w}$ gives the information about the direction. According to Ruder (2017), there are different variants of the gradient descent method that can be applied: Batch Gradient Descent, which computes the gradient using the whole dataset and then the average of the gradients of all the training samples is used to update the weights. This can take too long for large datasets;  Stochastic Gradient Descent, calculates the error for each example within the training set at a time, and the weights of each layer are updated with the computed gradient. The parameters of all the layers of the network are updated after every training sample, and the process is repeated until all the examples of the training set are gone through. It is computationally more faster;  Mini-Batch Gradient Descent, which divides the

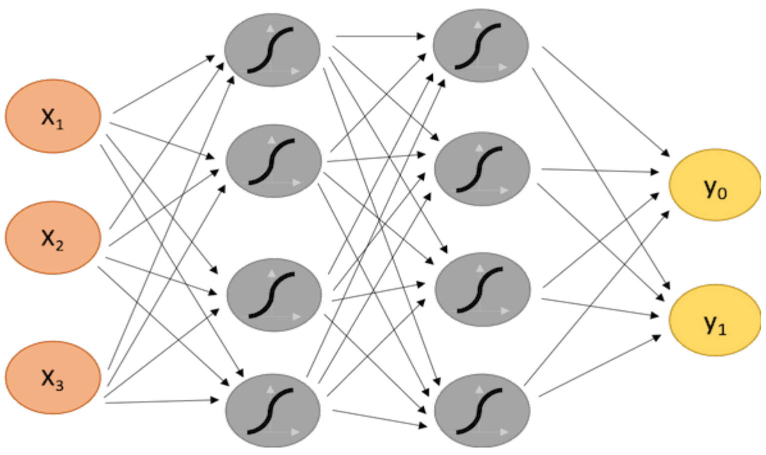


Figure 2.2: Feedforward Neural Network architecture

training dataset into small batches and performs an update for each of these batches. This achieves the advantages of both the former variants.

The classical Neural Network has no notion of order in time, and the only input it considers is the current example it has been exposed to. Hence, when aiming to build a model where the input's order has to be preserved and there is variable-length sequence data, an important sub-class of Neural Networks to consider is the Recurrent Neural Networks (RNN) (Elman, 1990). It is a type of neural network where outputs from previous time steps, $\mathbf{y_{t-1}}$, are taken as inputs, $\mathbf{x_t}$, for the current time step, forming a loop:

$$\mathbf{x_t} = \mathbf{y_{t-1}} \tag{2.8}$$

An RNN takes a sequence of vectors $\mathbf{x_1}, \mathbf{x_2}, ...\mathbf{x_t}$, where $t$ represents the time step in the sequence, and maintains a hidden state vector $\mathbf{h_t}$. When taking the next time-step in the sequence, in order to consider the information already extracted previously, the hidden state is calculated based on the previous hidden state, $\mathbf{h_{t-1}}$, and the current input, $\mathbf{x_t}$. That being said, the hidden states work as the memory of the network.

Translating the network mathematically:

$$\mathbf{h_t} = \mathrm{g}(\mathbf{V}\mathbf{x_t} + \mathbf{U}\mathbf{h_{t-1}} + \mathbf{b}) \tag{2.9}$$

In the previous equation, $\mathbf{V}$ represents the weights matrix used to transform the input into a hidden layer representation, $\mathbf{U}$ represents the weights matrix used to bring along information from the previous hidden state into the next time-step, and $\mathbf{b}$ represents the bias vector.

As it can be seen from the same Equation (2.9), the weights do not change depending on the time-step, which means that the recurrent layers share the parameters across time (also called Parameter Tying), with the purpose of reducing the number of parameters to be learned (Denil et al., 2013). Thus, when training the network, the exact same values are being multiplied layer by layer every time the Back-Propagation Trough Time (BPTT) process happens. On the downside, this can become a serious issue when the recurrent process is already at an advanced stage and has to backpropagate the error to the first steps in the network. This problem was explored in depth by Hochreiter et al. (2001) and Bengio et al. (1994) and was formally named as Vanishing Gradient. The name is self-explanatory: the gradient values exponentially shrink (vanish) as they propagate through each time step. Since small gradients

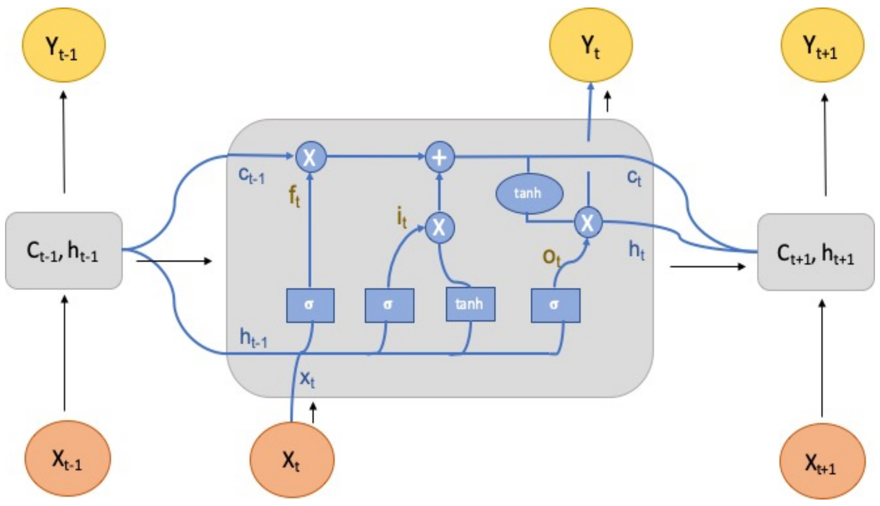


Figure 2.3: LSTM architecture, with three LSTM units depicted

mean small adjustments, in this way the purpose of using the gradient to make adjustments to the neural network's weights is lost. To overcome this condition, two sub-classes of Recurrent Neural Networks emerged: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks.

The LSTM neural network was introduced by Hochreiter and Schmidhuber (1997) and has been since popularized. The architecture of this network includes all the components from a regular RNN, plus a memory cell $\mathbf{c_t}$, where the information can be stored in, and to regulate the flow into the cell, it has three different gates: the forget gate **f** , input gate **i**, and output gate **o**.

The neural network can be formalized as follows:

$$\begin{aligned}
\mathbf{c_t} &= \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \mathbf{g} \\
\mathbf{h_t} &= \mathbf{o_t} \odot \mathbf{g}(\mathbf{c_t}) \\
\mathbf{f_t} &= \sigma(\mathbf{V_f x_t} + \mathbf{U_f h_{t-1}} + \mathbf{b_f}) \\
\mathbf{i_t} &= \sigma(\mathbf{V_i x_t} + \mathbf{U_i h_{t-1}} + \mathbf{b_i}) \\
\mathbf{o_t} &= \sigma(\mathbf{V_o x_t} + \mathbf{U_o h_{t-1}} + \mathbf{b_o}) \\
\mathbf{g} &= \tanh(\mathbf{V x_t} + \mathbf{U h_{t-1}} + \mathbf{b})
\end{aligned} \tag{2.10}$$

In the previous Equation (2.10), $\odot$ represents an element-wise multiplication or Hadamard product, which is given by: $c_{ij} = a_{ij}.b_{ij}$ , where A and B are two matrices ($a_{ij}$ and $b_{ij}$ represent elements in line $i$ and column $j$) of $n$ lines by $m$ columns, and C is the resulting matrix also with the same size $n$ x $m$.

As it can be visually noticeable, in Figure 2.3, the LSTM network, at time $t$, takes three inputs: the input of the current time-step, $\mathbf{x_t}$, the vector of the previous memory component $\mathbf{c_{t-1}}$ and the vector of the previous hidden state vector, $\mathbf{h_{t-1}}$. The forget gate $\mathbf{f_t}$ decides how much information from the cell state is going to be omitted or kept. It passes information from the previous hidden state, $\mathbf{h_{t-1}}$, and the current input, $\mathbf{x_t}$, through a sigmoid function which outputs values in the range [0,1]. Then, to update the cell state, the input gate $\mathbf{i_t}$ decides which information is relevant to keep from the $\mathbf{g}$ output. To get the new cell state, the previous state, $\mathbf{c_{t-1}}$, is pointwise multiplied by the forget vector. Here, if it gets multiplied by values close to zero, the previous information in the cell state will not be considered. Next, by adding the output from the input gate, we get a new cell state. Lastly, the output gate $\mathbf{o_t}$ determines the next hidden state, taking into account the newly modified cell state, $\mathbf{c_t}$. The information that will be carried to the next time step, will be the new cell state and the new hidden state.

Introduced by Cho et al. (2014), the GRU is a similar network to the LSTM. The main difference relies on the non-existence of a cell state. Instead, it solely uses the hidden state vector to transfer information over time. The architecture of the network also differs in the number and types of gates. It has two gates: reset gate $\mathbf{r}$ and update gate $\mathbf{u}$.

$$\begin{aligned}
\mathbf{h_t} &= \mathbf{u_t} \odot \tilde{\mathbf{h_t}} + (\mathbf{1} - \mathbf{u_t}) \odot \mathbf{h_{t-1}} \\
\tilde{\mathbf{h_t}} &= \mathbf{g} \\
\mathbf{g} &= \tanh(\mathbf{V}\mathbf{x_t} + \mathbf{U}(\mathbf{r_t} \odot \mathbf{h_{t-1}}) + \mathbf{b}) \\
\mathbf{r_t} &= \sigma(\mathbf{V_r}\mathbf{x_t} + \mathbf{U_r}\mathbf{h_{t-1}} + \mathbf{b_r}) \\
\mathbf{u_t} &= \sigma(\mathbf{V_u}\mathbf{x_t} + \mathbf{U_u}\mathbf{h_{t-1}} + \mathbf{b_u})
\end{aligned} \tag{2.11}$$

The candidate update, $\tilde{\mathbf{h_t}}$, uses the reset gate $\mathbf{r_t}$, to store relevant information from the past. Then, the network calculates $\mathbf{h_t}$, by using the update gate $\mathbf{u_t}$, and determines what to consider from the current memory content, $\tilde{\mathbf{h_t}}$, and from the previous step $\mathbf{h_{t-1}}$.

As it can be seen, the GRU has fewer operations, therefore it is faster to train than LSTMs, but none of them clearly outperforms the other - it all depends on the data used. In sum, the main difference between a regular RNN and its variants LSTM and GRU, is that instead of multiplying across time (which leads to exponential growth), these newer versions of RNNs have better gradient propagation due to the additive update function used.

In the paper written by Krause et al. (2016), a problem related with the LSTM is pre-

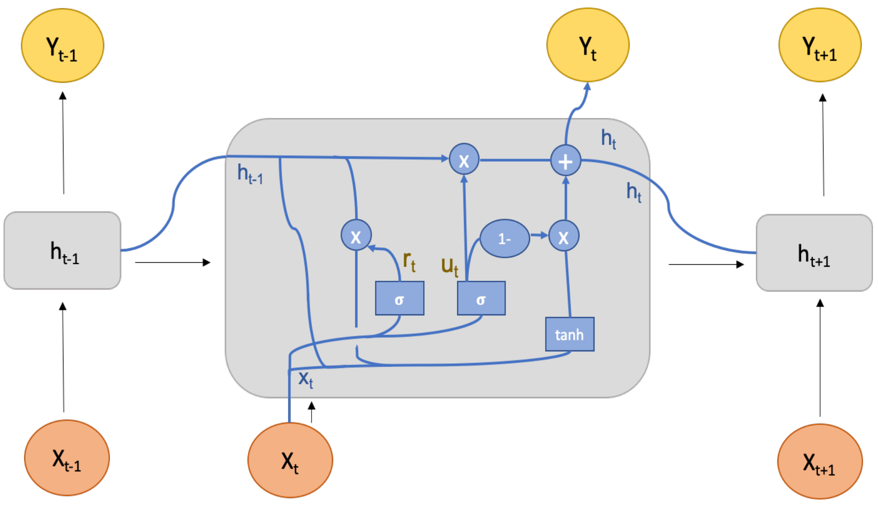


Figure 2.4: GRU architecture, with three GRU units depicted

sented: it happens that many of the operations in the network involve an addition of the transformed input, $\mathbf{V_*x_t}$, with the transformed hidden state vector, $\mathbf{U_*h_{t-1}}$, in the form $\sigma(\mathbf{V_*x_t} + \mathbf{U_*h_{t-1}} + \mathbf{b_*})$. Therefore, in case there is an unexpected input, $\mathbf{x_t}$, many operations are affected and it might lead to irregular future cell and hidden states. This is a situation of which the network may not be able to recover from. Hence, with the intention of overcoming wrong inputs fed to the LSTM, a newer version of this network was created: the Multiplicative LSTM (or mLSTM). In an mLSTM, there is an intermediate state, $\mathbf{m_t}$, adding to the already existing components of a regular LSTM:

$$\begin{aligned}
\mathbf{m_t} &= \mathbf{V_m x_t} \odot \mathbf{U_m h_{t-1}} \\
\mathbf{c_t} &= \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \mathbf{g} \\
\mathbf{h_t} &= \mathbf{o_t} \odot \mathbf{g}(\mathbf{c_t}) \\
\mathbf{f_t} &= \sigma(\mathbf{V_f x_t} + \mathbf{U_f m_t} + \mathbf{b_f}) \\
\mathbf{i_t} &= \sigma(\mathbf{V_i x_t} + \mathbf{U_i m_t} + \mathbf{b_i}) \\
\mathbf{o_t} &= \sigma(\mathbf{V_o x_t} + \mathbf{U_o m_t} + \mathbf{b_o}) \\
\mathbf{g} &= \tanh(\mathbf{V x_t} + \mathbf{U m_t} + \mathbf{b})
\end{aligned} \tag{2.12}$$

As it can be seen in the above Equation 2.12, the hidden state, $\mathbf{h_{t-1}}$, and the input, $\mathbf{x_t}$, are transformed with an element-wise multiplication in the intermediate state, $\mathbf{m_t}$. This allows $\mathbf{m_t}$ to flexibly change its value in accordance to $\mathbf{h_{t-1}}$ and $\mathbf{x_t}$, which implies that the mLSTM can learn parameters specifically for each possible input at a given time-step. The great advantage of this type of neural network is that it allows the hidden state to react with flexibility to a new surprising input, by using input-dependent transition functions.

## 2.2 Related Work

There are two main lines in recommender systems: session-aware and session-based recommender systems (Quadrana et al., 2018). The session-aware systems work from the assumption that the user can be identified in returning sessions and consequently the history of actions of the user is propagated from session to session, powering more personalized recommendations to a specific user. The session-based systems do not considered user identification and work with the actions from the current session to produce recommendations suitable to any user surfing the website. This section overviews, in a chronological order, previous studies mainly on session-based recommender systems, and one particular scientific work that combines both session-based and session-aware recommendations.

A frequently mentioned baseline model for systems aiming to predict the next item's choice of the user, includes simple item-to-item collaborative filtering recommendations (Schafer et al., 2007), where based solely on a similarity score between the last clicked item and the remaining items available, the system predicts the most likely items to follow next in the current session.

Each item is encoded as a binary vector where each position corresponds to a session, and in case the item appears in a specific session that position is positively flagged in the vector. The distance between a target item and the other available items is then calculated using a similarity function, such as the cosine similarity, and the top closest items are retrieved by the system as "others who viewed this item also viewed these ones".

In the end, items often interacted together in sessions from other users are considered as being similar to each other, and this has been actually a strong statistical approach proved to work in many e-commerce contexts (Fang et al., 2019) (Ning and Karypis, 2011) (Linden et al., 2003) (Ning et al., 2015), since the items similarity matrix can be pre-computed offline and sorted in the training process to ensure fast responses at recommendation time, however, it ignores information from past clicks.

To create complexer models that are able to take into account the items dependencies in the logs, there has been many scientific works focusing on this matter. A first advance was presented by Lin et al. (2011). An approach based in information retrieval methods is applied to predict future search actions of the user based on past ones. The main objective in this research is to find out whether the predictions are more accurate, when made based on the complete or recent history of queries and URLs the user has submitted or clicked. The article presents three methods to predict future actions. The first, called Weighted Tally (WTAL), in a historical portion of query logs, $H$, of a search session, checks the co-occurrence of a candidate future action a$'$, and a recent search action $a$ in the session, in order to compute a relevance score WTALSC:

$$\text{WTALSC}(a', a, H) = \sum_{X \in T_r} \sum_{a_i, a_j \in X : a_i = a, a_j = a', j > i} \left( \frac{1}{j - i} \right) \tag{2.13}$$

In Equation (2.13), $X$ is a session in the training set $T_r$, and $a$ and $a'$ are respectively the $i$-th and $j$-th actions in $X$. The relevance is higher when $a'$ is closer to $a$ in the session. In the end, the future actions candidates WTALSC scores are sorted in descending order, where the top rank scores are the most likely actions to be taken by the user in the next steps.

The second presented method is called Session Retrieval with Prediction by Frequency (SRPF). Firstly, there is an Indexing step, to distinguish the various types of actions in the training set, where each session, in the training set, is assigned with an intent between a set of groups, such as ‘Query terms’, ‘Clicked URLs’, or ‘Clicked URLs domain names’. This facilitates the next step, where the training sessions related to the current user’s search context $H$ are to be retrieved. After getting the training sessions related to $H$, the actions that occur more frequently in the retrieved training sessions are to be considered as candidates to user’s future actions. To tally those frequencies of actions, two weighting styles are considered. The first gives the same weights to every occurrence of an action:

$$\text{FREQ}(a, H) = \sum_{X \in \text{RETR}(H)} \sum_{a_i \in X : a_i = a} 1 \tag{2.14}$$

In Equation (2.14), RETR($H$) is the set of training sessions in the retrieval result of $H$. The second weighting method, weights the actions according to the relevance score of the sessions in which they are inserted.

$$\text{WFREQ}(a, H) = \sum_{X \in \text{RETR}(H)} \sum_{a_i \in X : a_i = a} \text{REL}(X, H) \tag{2.15}$$

In the above Equation (2.15), REL($X, H$) is the relevance score of session $X$ with respect to the retrieval result of $H$. With any of mentioned weighting methods, in the end, the top frequencies are sorted in descending order.

Lastly, another method to predict future actions is presented as ACTF (action flow graph method), which follows an approach based on the PageRank algorithm. Each node of the graph represents a query or clicked URL that appears in the search logs, and the weights are higher when closer to the nodes that represent queries or clicked URLs by the user. The nodes with top PageRank scores are considered the predicted future actions of the user.

In the end, it is shown that when using a combination of the three prediction methods, better results are achieved. Moreover, to answer the main question of the paper, of whether or not the prediction's accuracy would be higher when using full or recent history of actions, experimental results demonstrated that by using more historical search information in a user's search action it is not guaranteed to improve predictions performance.

From this point on, most of the literature found presents solutions that evolve from Recurrent Neural Networks (Wang et al., 2019; Ludewig and Jannach, 2018; Fang et al., 2019). The first application of RNN to model the session data is presented by Hidasi et al. (2015) and Hidasi and Karatzoglou (2018), as GRU4Rec, which is also the first approach to the recommendation problem as a ranking problem rather than solely a classification problem. A GRU model is used, since it is stated that when using LSTM the same results were achieved but the model's performance was slower.

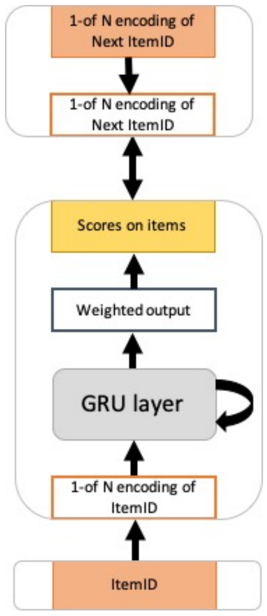

1-of N encoding of Next ItemID
1-of N encoding of Next ItemID
Scores on items
Weighted output
GRU layer
1-of N encoding of ItemID
ItemID

Figure 2.5: GRU4Rec architecture

The network's architecture consists of one or more GRU layers, depicted in Figure 2.5 with the cycle arrow, preceded by an optional embedding layer not represented in the figure. The input given is an item encoded in a 1-of-N way (also referred as one-hot encoding), where N is the total number of items, and the coordinate is 1 if the corresponding item is active in the session, otherwise is 0. The hidden state is also encoded in the same way, having a vector representing the entire item space of previously clicked items in the same session. The output is computed using the resulting weight output which is a vector of a similar shape as the input that contains the likelihood scores of each item in the catalogue being the next one inline. During training, scores are compared to the vector of the next item in the session, considered as the target item, to compute the loss using a ranking loss function (this action is represented by the bidirectional arrow in Figure 2.5).

When there are variable length sessions and a catalogue with a huge amount of items, the use of sequence by sequence training and the BPTT algorithm decrease the performance of the model. Thus, to optimize the training of the network, the authors use session-parallel mini-batches. As it is depicted in Figure 2.6, sessions are stacked on top of each other, and the items in the same position in the different sessions form a mini-batch. The input for each mini-batch, are the current events, and the output of each mini-batch are the next events. Whenever sessions are finished, these are replaced by the next available. In the end, instead of using the BPTT algorithm on each session separately, the model will do BP on the mini-batches assembled from multiple sessions, which decreases the amount of time spent in training. Furthermore, an optimization to reduce the computational complexity when computing the loss is also introduced. The loss is computed over the item ID the user has clicked, named as positive item, and a subset of the items' IDs that haven't been clicked upon the session, that are considered as the negative items. The ranking method that obtained best results was with Pairwise ranking, which looks at a pair of instances at a time, one positive and one negative item, and the loss function enforces the rank of the positive item to be lower than the negative one, and in this way the optimal order for that pair is found.

To evaluate the performance of the model, the position of the predicted item in the ranking output list of items is considered. The setting that achieved the best results, is composed of a single GRU layer without feedforward layers, and the TOP1 pair-wise loss function along with session-parallel mini-batching. In the end, the achieved model proved to be better than the baseline item-to-item recommendations.

All in all, GRU4Rec can accumulate the influence (rank) of all the sequential items from the session to make a more reliable recommendation, in the form of ranking of the items in the catalogue. In addition, it is worth mentioning that many of the upcoming works include, in someway, the use of a GRU4Rec model.

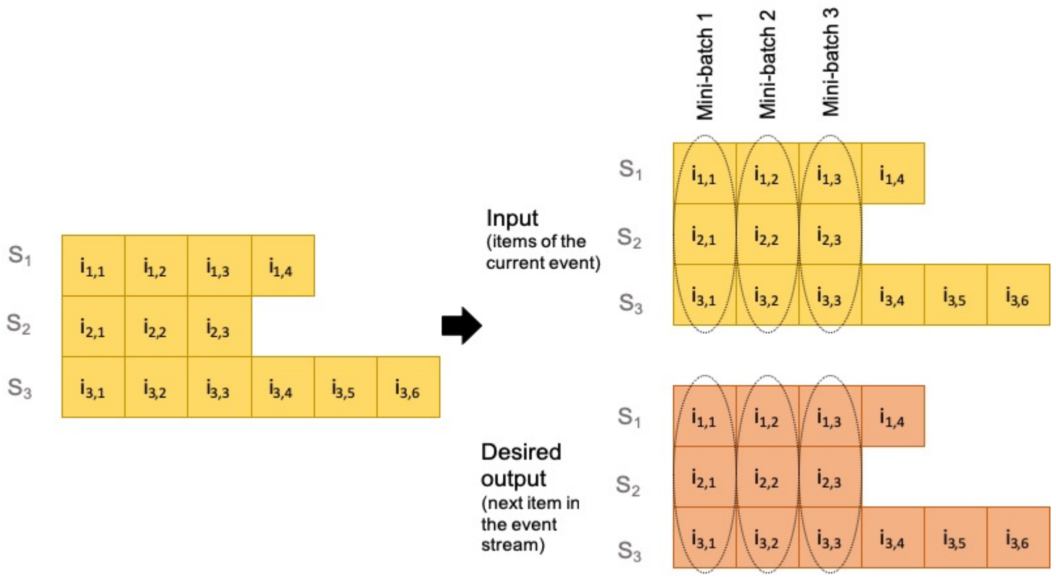

Mini-batch 1
Mini-batch 2
Mini-batch 3
$S_1$
$S_2$
$S_3$
$i_{1,1}$ $i_{1,2}$ $i_{1,3}$ $i_{1,4}$
$i_{2,1}$ $i_{2,2}$ $i_{2,3}$
$i_{3,1}$ $i_{3,2}$ $i_{3,3}$ $i_{3,4}$ $i_{3,5}$ $i_{3,6}$
Input
(items of the current event)
Desired output
(next item in the event stream)

Figure 2.6: Session parallel mini-batches

To model feature-to-item recommendations, a parallel RNN (p-RNN) version emerged in Hidasi et al. (2016), that can take into account the clicked items as well as their characteristics, such as text (e.g. item's short description), images (e.g. item's appearance), or price. Thus, the goal is to include these rich-features into session-based recommendations from Hidasi et al. (2015) previous work about GRU4Rec. The architecture of the model is composed by GRUs for each characteristic and another to model the session (a GRU4Rec), as presented in Figure 2.7. The networks work separately as independent neural networks, and in the end, the hidden layers are concatenated, by an element-wise multiplication of the hidden state of the ID subnet and the hidden state of the item feature subnet(s). Then, the output is computed from the concatenated hidden state.

The training strategies are crucial in the process, since simultaneous training in both subnets, by using a backpropagation method across the whole architecture, leads to the different components of the p-RNN learning the same relations from the data. Thus, to force the network to learn different aspects, the authors give out three alternative training strategies that can be applied: Alternative training, keep one subnet fixed and train the other in one epoch, then do the other way around in another epoch, and the cycle restarts after each subnet is trained; Residual training, a subnet A is trained and subnet B is trained on the residual error of subnet A, and so on, but subnets already trained will not be visited again; and lastly, Intervealed training, alternates training per mini-batch.

The evaluation of the model was done using the same metrics as in GRU4Rec research paper, and at last, it is shown that p-RNN, using any of the alternative training strategies, gives

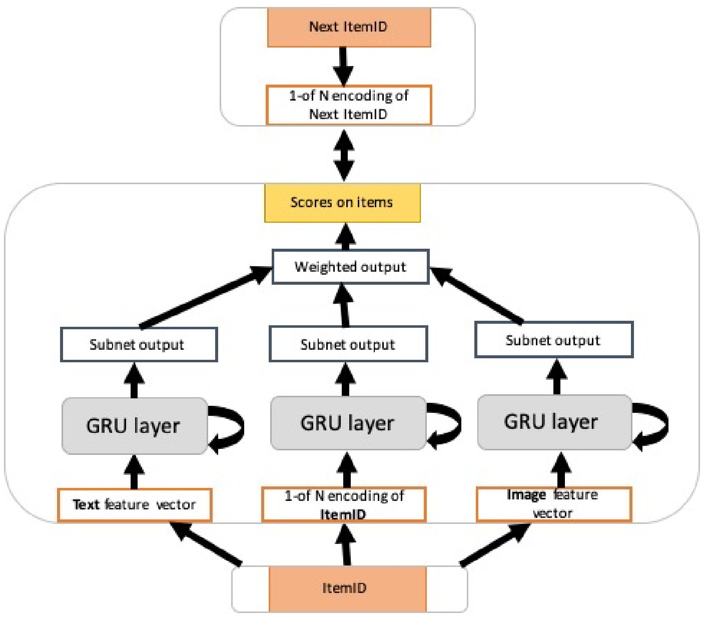


Figure 2.7: p-RNN architecture

more accurate recommendations than the baseline model of a single network that models the session with a GRU4Rec.

Hierarchical RNNs (HRNN) start to show up, by Quadrana et al. (2017), with the goal to, in addition to the information of the current session, also use information stored from previous sessions of the user, by using information associated to his account in the website. A combination between session-aware and session-based recommender systems is done hierarchically: the GRU that models the user activity across sessions, $\text{GRU}_{\text{usr}}$ is ran on top of the GRU that models the session, $\text{GRU}_{\text{ses}}$ - a GRU4Rec network. For a returning user, in a new session, the $\text{GRU}_{\text{usr}}$ takes the last hidden state of this network, and the last hidden state of the previous session network $\text{GRU}_{\text{ses}}$, and given these two hidden states, the $\text{GRU}_{\text{usr}}$ tries to predict a good initialization for the new session $\text{GRU}_{\text{ses}}$ (notice the 'session initialization' arrow in Figure 2.8) - this architecture is refered as HRNN Init. The session-level representation $\text{GRU}_{\text{sess}}$ is initialized and updated, from the second session $\text{s}_2$ on, as:

$$\mathbf{s_{m,0}} = \tanh(\mathbf{W_{init}c_{m-1}} + \mathbf{b_{init}}) \tag{2.16}$$

$$\mathbf{s_{m,n}} = \text{GRU}_{\text{ses}}(\mathbf{i_{m,n}}, \mathbf{s_{m,n-1}}) \tag{2.17}$$

In the previous equations, $\mathbf{W_{init}}$ and $\mathbf{b_{init}}$ represent the initialization weights and biases, and $\mathbf{i_{m,n}}$ represents the input item in session $m$, at time step $n$. Whereas, the user-level representation

$\text{GRU}_{\text{usr}}$ is updated as:

$$\mathbf{c_m} = \text{GRU}_{\text{usr}}(\mathbf{s_m}, \mathbf{c_{m-1}}) \tag{2.18}$$

In the previous Equation (2.18), $\mathbf{s_m}$ represents the last hidden state of session s, and $\mathbf{c_{m-1}}$ represents the previous user-state. $\mathbf{c_0} = 0$, is a null vector, and for that reason it is not represented in Figure 2.8.

In fact, the representation of the $\text{GRU}_{\text{usr}}$ can not only be used to initialize the new session representation, but also be used as input in all the steps of the user's session model - refered as HRNN All - which is marked as dashed grey lines in Figure 2.8, and in that case, the update in the session-level network is done considering $\mathbf{c_{m-1}}$:

$$\mathbf{s_{m,n}} = \text{GRU}_{\text{ses}}(\mathbf{i_{m,n}}, \mathbf{s_{m,n-1}}, \mathbf{c_{m-1}}) \tag{2.19}$$

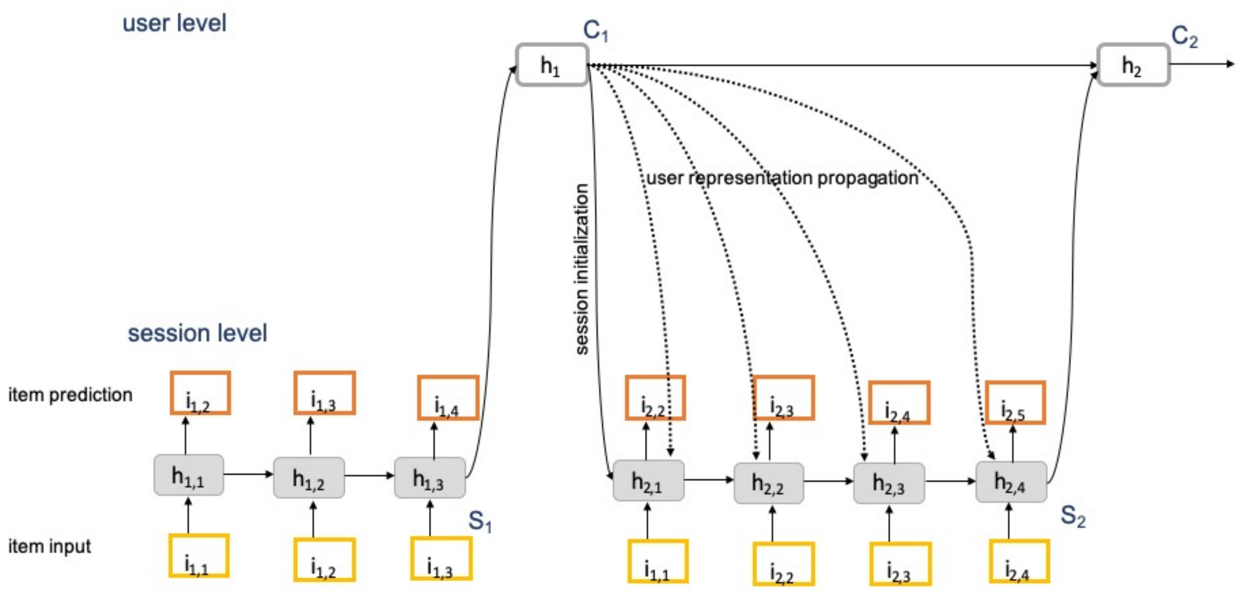


Figure 2.8: HRNN (HRNN Init and HRNN All) architecture

The training of the whole network is done end-to-end using back-propagation in mini-batches, in the same way as in the baseline GRU4Rec. When going through the experimental results, it can be realized that the HRNN All, with the forced user representation propagation degrades the performance of the model when used on a video recommendation website context. In any case, any of the HRNNs used, performs better than the baseline model GRU4Rec, which can indicate that returning users tend to click on similar items from previous sessions, and thus these personalized recommendations turn to be more accurate predictions.

Another approach to predict the next action of the user, is proposed by Smirnova (2018). An extension to the usual recommending system is presented, where not only the navigation history is taken into account, but also the history of recommended items. To that end, a new GRU model is created, which includes a recommendation action representation - a one-hot enconded items IDs recommended at timestep $t$, $\mathbf{a_t}$ - and state-action fusion mechanisms. The model is built on top of the GRU4Rec, from Hidasi et al. (2015), with the difference that the hidden state is computed based on one of two fusion mechanisms, in order to integrate $\mathbf{a_t}$. This step can either happen with early fusion, that updates the current RNN hidden state:

$$\mathrm{f}^{\mathrm{early}}(\mathbf{h_t}, \mathbf{a_t}) = \mathbf{h_t} \odot \mathbf{W^a a_t} \tag{2.20}$$

Or with late fusion, that updates the next hidden state:

$$\mathrm{f}^{\mathrm{late}}(\mathbf{h_{t+1}}, \mathbf{a_t}) = \mathbf{h_{t+1}} \odot \mathbf{W^a a_t} \tag{2.21}$$

In the previous Equation (2.21), $\mathbf{W^a}$ is the projection matrix between action space and RNN hidden state.

The main goal is to obtain:

$$\mathrm{p}(\mathbf{x_{t+1}} | \mathbf{a_t}, \mathbf{x_{\leq t}}) \propto \mathrm{f}^{\mathrm{out}}(\mathbf{h_{t+1}}) \tag{2.22}$$

In (2.22), $\mathrm{f}^{\mathrm{out}}$ represents the output layer of the neural network, where the next hidden state $\mathbf{h_{t+1}}$ of the model is:

$$\mathbf{h_{t+1}} = \mathrm{f}^{\mathrm{late}}(\mathbf{h_{t+1}}, \mathbf{a_t}) \tag{2.23}$$

The training of the model is done in the same way as in GRU4Rec, with a different loss function, a negative log-likelihood function. The authors concluded that the model that incorporates $\mathbf{a_t}$ in the next hidden state (late fusion), which indirectly conditions at output layer of the network, has the best results and furthermore it outperforms the baseline model GRU4Rec.

Another perspective to bear in mind in e-commerce recommending situations, is the fact that occasions such as a user's birthday, considered as a Personal occasion, or Valentine's day, considered as a Global occasion, can lead to deviated actions from the user's path in history. To address these circumstances, Wang et al. (2020) came up with the concept of Occasion-Aware Recommendation (OAR).

The attention mechanism, particularly the self-attention (Vaswani et al., 2017), takes a great part in the system. The self-attention measures the internal components of the input to identify the most valuable ones, wherein the resulting output, the attention vector **o**, is given by:

$$\mathbf{o} = \sum \alpha_{\mathrm{q}} \mathbf{v} \tag{2.24}$$

In the above equation, the attention scores, $\alpha$, are given by:

$$\alpha_{\mathrm{q}} = \operatorname{softmax}\left( \frac{\exp^{\operatorname{sim}(\mathbf{q},\mathbf{k})}}{\sum \exp^{\operatorname{sim}(\mathbf{q},\mathbf{k})}} \right) \tag{2.25}$$

The similarity function, $\operatorname{sim}(\mathbf{q}, \mathbf{k})$, used between keys, $\mathbf{k}$, and queries, $\mathbf{q}$, is the scale dot product. The shown mechanism is used throughout the components of the system, to give attention weights to the different types of occasion signals.

To model Intrinsic preferences, not taking into account any special occasions in the context but just the own preferences of the user, the correlation between the most recent purchase and the personal historic purchases is to be found, to predict the next item to be purchased. Thus, the query $\mathbf{q}$ is done towards the current timestamp, and the attention weights will be calculated as shown above. Next, the obtained output $\mathbf{o}$ will be used to infer the user's next purchase.

To model Personal occasions cases, it is important to check the neighbouring days, in relation to the current time step, in previous history, in order to give more attention to those items. Thus, the query $\mathbf{q}$ is done to the upcoming personal occasion timestamp, which will be mapped to the timestamps of the user's previous purchases until the current timestamp (keys $\mathbf{k}$), and its corresponding products, (values $\mathbf{v}$). This way, the model gives higher attention to items purchased a long time ago within an approximate time window from the current timestamp, which can capture personally reoccurring occasions.

Global occasions can be captured by the general behaviour of all users in the website, by again, checking a near time period in past history - here, separated memory slots are used to store global behaviours of the users. The query $\mathbf{q}$ is done in the same way as previously, where the values corresponding to the keys are retrieved from the memory slots this time.

In addition to the mentioned self-attention layers, a gating layer is also added to the model. The objective is to balance the global and personal occasions influences, in which it assigns different weights, with an additive attention operation, to the different components (Intrinsic, Personal and Global). This is a relevant mechanism, since different occasions can have different impact from user to user, at different time steps. In the end, OAR proved to outperform the baseline models used which include the GRU4Rec model.

## 2.3 Overview

This chapter started by presenting the fundamental concepts for this work. This included an introduction to a range of models applicable to a sequence prediction problem. In a second section, there was an overview of the literature which is relevant to this work. From the literature review, one can note that the initial steps in the domain of session-based recommender systems were taken with simpler non-deep learning models, which have shown to be effective in many e-commerce sequence modeling problems, yet these are not able to learn the dependencies between past actions in history, thus, the tendency of scientific research has been to focus on more sophisticated deep learning solutions based on Recurrent Neural Networks.

The next chapter presents the proposal for this thesis and the different architectures that will be used to model the sequence prediction tasks in hand.

# 3 The Proposed Approach

An action is represented by a set of different components, for instance, the shopping item ID or the interaction type made by the user to the item, would be regarded as distinct components of the action. That said, the defined problem to predict the next-action in a session, is disintegrated into individual sequence prediction tasks corresponding to the prediction of each action component. In this work, each prediction task is addressed as a classification problem, as it is further explained on Section 3.1, and the whole process behind the different classification models used is presented on Section 3.2.

## 3.1 Classification Problem

The proposed sequence modeling task for each action component, is addressed as a classification problem: the class labels are the distinct elements for the action component, and each session is to be classified with the last element in the sequence of elements. For instance, for the prediction of the item ID component of an action, the classes are the distinct items IDs available, and each session is classified with the last item ID present in the sequence of items.

## 3.2 Implemented Models

The starting point was given with the creation of a simple model easy to set up and that performs reasonably enough with the available data. Next, a statistical approach was taken, with N-th order Markov chain, and lastly, more complex models were created, based on Recurrent Neural Networks.

All models have been generated in a Python[1] 3 environment, using mainly the libraries Scikit-Learn [2], NLTK [3] and Keras [4], with Tensorflow [5] backend. The next sections introduce

[1]https://www.python.org
[2]http://scikit-learn.org
[3]https://www.nltk.org
[4]https://keras.io
[5]https://www.tensorflow.org

and describe the different implemented models.

### 3.2.1 Baseline Approach

The first approach taken was with a model easy to configure and fast to train, with the aim to set an initial benchmark before moving into more complex models that can be harder to interpret and deploy. Two models were initially created: one that classifies all sessions with the most frequent class appearing in the dataset, and another, that classifies each session with the exact same element as the one coming last in the session. The latter approach was the most suitable to the available data, and henceforth is the reference model to take into account.

### 3.2.2 Statistical Approach

A core statistical model for the study of the course of actions through time is the Markov chain (Section 2.1.1), also known as first-order Markov chain. This sequence model follows the assumption that the probability of a state is given only by the immediate previous state, and when wishing to use more preceding states to infer about the next state transition, a higher N-th order is to be considered.

To design an N-th order Markov chain, the first step taken was to define the set of states to be considered. The states are contiguous sequences of N elements, also referred as N-grams, extracted from the training sessions. Afterwards, for each N-grams model, a matrix was created with a dimension correspondent to the number of states, where each position describes the frequency counts of moving from one state to another. These frequencies were then translated into transitions probabilities, using additive smoothing due to many non-existent states transitions.

To get the most likely element coming next on a given test session, the last N elements from the sequence are checked against the created matrix, and the element with the highest transition probability corresponds to the predicted next element.

### 3.2.3 Recurrent Neural Networks Approach

When aiming to build a model where the input's order has to be preserved and there is variable-length sequence data, an important sub-class of neural networks to consider are the Recurrent Neural Networks (RNN). The architecture of the RNN model built in this work is similar to GRU4Rec (Section 2.2).

The model receives each training session, in the form of a sequence of elements, from a specific action's component. Considering that the available data sessions have variable-length, the input sequences that are shorter than the maximum length, are padded with a sentinel value that is later ignored when passed through a masking layer. The padding is made at the end of each sequence sample, since what is intended for the hidden state of an RNN is the state of the last valid non-padded input step.

To feed the prepared sequences to the network, the first aspect to bear in mind was the categorical data representation. By default, each element is one-hot encoded by the total number of distinct elements for the action component, giving an equal distance relationship between all elements (Cui et al., 2018). However, for action's components with an excessive number of distinct elements, the binary vectors representing each element result in overly large and sparse one-dimension vectors, which turn to be computationally inefficient, thus, in these cases, an embedding layer is added as an input layer. In this additional layer, each element is mapped to a distinct vector, and the properties of the vector are adapted while training the neural network, making commonly co-occurring elements to naturally group together in the representation space (Hancock and Khoshgoftaar, 2020).

The first hidden layer of the network, that receives the input, is an RNN type of layer, which can be a simple RNN, a GRU, an LSTM or an mLSTM (Section 2.1.2). In this layer, it is specified that the model is to be in the form Many-to-One, meaning that given a sequence as input it returns the output at the last timestep. That last vector is then passed through a dropout layer for regularization, followed by a final dense layer responsible for outputting the most likely element to follow next in the session.

Since this is a multi-class problem, the model uses a combination of a categorical cross-entropy loss function with a softmax activation function (Goodfellow et al., 2016). The softmax function, as in Equation 3.1, re-scales the output vector to a normalized probability distribution: it passes each output result, $x_i$, through an exponential, to really highlight the largest numbers and suppress the ones significantly below the maximum. Next, these output results are normalized to ensure that all probabilities sum up to 1.

The categorical cross-entropy loss function (Rubinstein and Kroese, 2004), formally represented in Equation 3.2, quantifies the difference between the two probability distributions, from the predicted vector and expected vector - which is in fact a one-hot vector considered as a probability distribution. A separate loss is calculated for each element in the catalogue (the class labels), that are then summed to form the loss for a particular sequence sample. The minus sign ensures that the loss gets smaller when the expected and predicted distributions are closer to each other. The output probabilities will be considered as scores, where the highest probability represents the highest ranked element, which is to be considered as the predicted one.

$$f(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))} \tag{3.1}$$

$$CE = -\sum_x y_i \log(f(x)_i) \tag{3.2}$$

The architecture of the described model is depicted in Figuere 3.1.

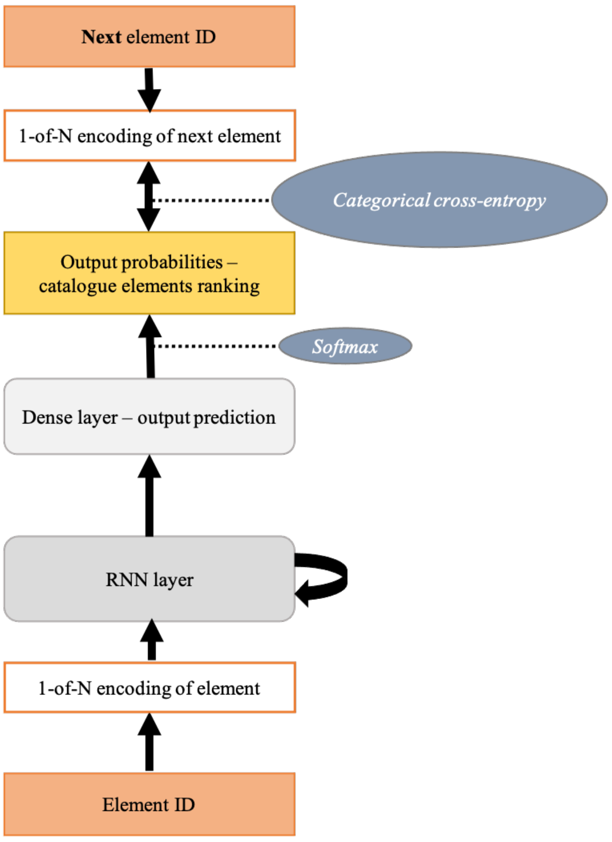
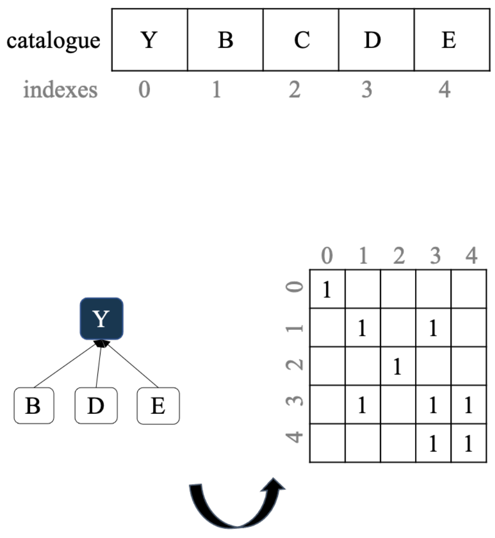

Next element ID
1-of-N encoding of next element
Categorical cross-entropy
Output probabilities – catalogue elements ranking
Softmax
Dense layer – output prediction
RNN layer
1-of-N encoding of element
Element ID

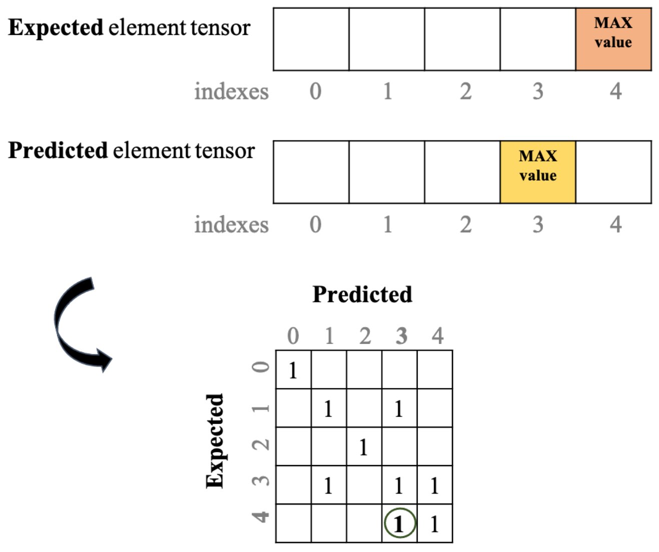
Figure 3.1: RNN model scheme.

#### 3.2.3.1 Modified training process

The different action's components can be related between each other in an item-feature perspective as well as between the different item's features - for instance, an item can belong to a certain category, meaning that there is an hierarchical relation between the item and the category feature.



(a) Relationships matrix creation.



(b) Relationship verification.

Figure 3.2: Workflow prior to the calculation of the categorical cross-entropy loss.

To take advantage of this contextual information, the existing dependencies are incorporated in the training process of the created RNN model. The objective is to lead the model into always predict an element within the same group as of the expected element. To exemplify, for a target shopping item ‘ring’ that belongs to the category ‘jewelry’, the items to be considered in the catalogue should only be jewelry pieces from the same group as well, such as a ‘necklace’ or ‘earrings’, meaning that although the model could miss the expected prediction of the item ‘ring’, at least it can give out a closely related shopping item from the same category group.

To that end, a modified version of the loss function was created. The first step taken here is to signal the type of relationship between the different elements. A square matrix is built where the columns and rows indexes represent each available element, in the same order as in the catalogue list. In case two elements share the same parent Y, the corresponding position in the matrix is flagged as a positive relation with the value ‘1’. On Figure 3.2a, the elements identified as B, D and E, share the same parent, thus the corresponding positions of the elements in the matrix are marked as being related to each other. This relationships matrix is then passed to the new loss function.

During the training process of the network, the vectorial representation of the expected and predicted elements is in the form of an immutable multi-dimensional array, named as tensor, which are accessed by the loss function. Each tensor has the exact array-shape of the catalogue of elements, and each index corresponds to the position of an element in the catalogue list. Hence, to get the expected and predicted element, the index of the maximum probability distribution in each tensor is extracted. Then, to check if these are related, the indexes of the expected and predicted elements are spotted in each side of the relationships matrix and the corresponding value indicates if there is a relation. In a positive case of a relationship, a penalization value will be applied to the categorical cross-entropy result, as it is expressed in Equation 3.3, otherwise the result is not altered. The penalization value, $p$, is used with the objective to decrease the error and therefore benefit these cases, this way it is expected that the model learns to only consider elements in the catalogue from the same family group as the expected one.

The workflow of these last steps in the loss function is depicted in Figure 3.2b, where the indexes for elements D and E are checked in the matrix for a relation that is indeed present, and therefore, a penalization value $p$ will be applied to the cross-entropy result.

$$CE = \begin{cases} -\sum_x y_i \log(f(x)_i) * p & \text{positive relation} \\ -\sum_x y_i \log(f(x)_i) & \text{negative relation} \end{cases} \tag{3.3}$$

The architecture of the described model is depicted in Figure 3.3.

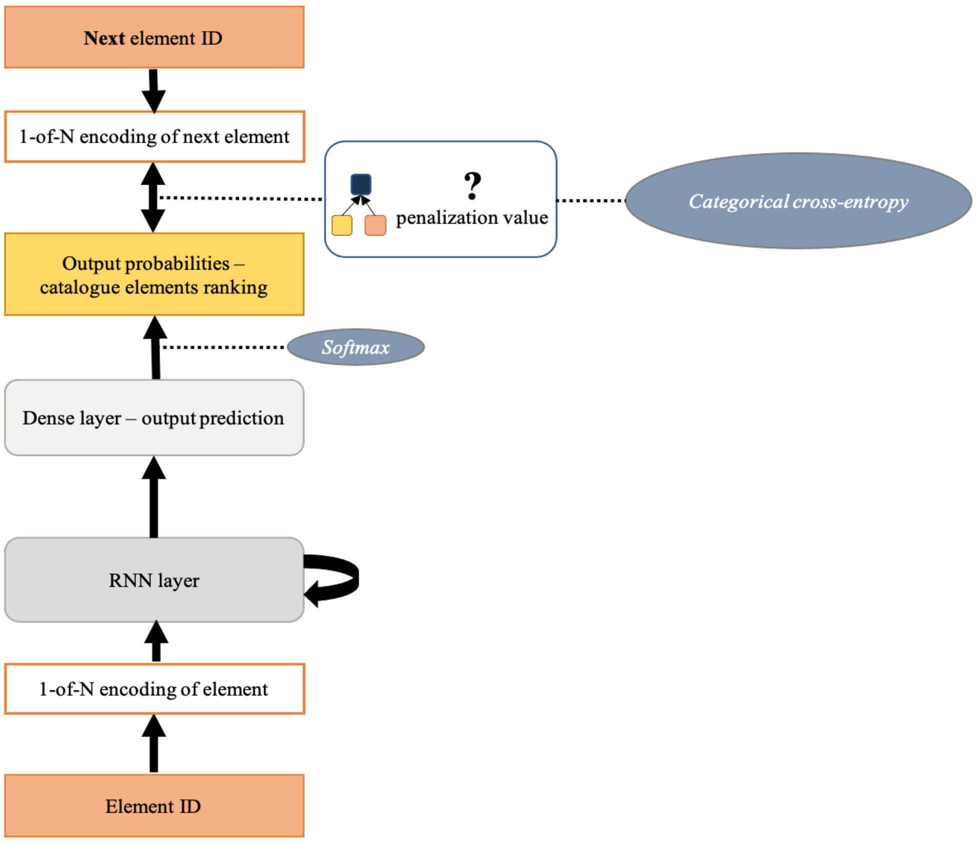

Next element ID
1-of-N encoding of next element
?
penalization value
Categorical cross-entropy
Output probabilities – catalogue elements ranking
Softmax
Dense layer – output prediction
RNN layer
1-of-N encoding of element
Element ID

Figure 3.3: RNN model scheme with modified loss function.

#### 3.2.3.2 Modified evaluation process

To prevent the RNN model from even considering elements in the catalogue that, once again, do not belong to the same family as of the expected element, the evaluation process was also modified. The final output probabilities list is cut down to only include elements corresponding to the same family group. This way, it is ensured that there are no noisy predictions with elements representing other family groups.

## 3.3 Overview

This chapter started by presenting the proposed sequence prediction problem as a classification problem. The next section introduces the three different approaches taken to create the classification models. The implemented models are iteratively more complex, starting from a simple baseline model, to a statistical N-th order Markov chain, and finally, to a Recurrent Neural Network model. Later on, some alterations are proposed to the RNN model, where it is suggested a modification in the training and evaluation processes, in order to incorporate contextual information coming from the existing relations between the action's components.

The use of different architectures allows, in the next chapter, to test which brings the best results for each action's component sequence prediction task.

# 4 Experimental Methodology and Results

This chapter presents all the process behind the evaluation of the designed models. Section 4.1 analyses the e-commerce dataset used. Section 4.2 introduces the metrics that were extracted to fairly compare the performance of the different models. Section 4.3 describes all the conducted experiments done to address the sequence modeling problem with the given data, and later reports and discuses the obtained results for each experiment.

## 4.1 Dataset

The dataset used by this work was collected from an e-commerce website, provided by RetailRocket [1] in a Kaggle repository [2], in a raw format, barring hashed values due to data anonymization issues. The data is provided in three separate files: one file containing the usage logs from the website, other containing the items details, and another with the item's categories details.

The collected usage logs are comprehended in a period of about 4 months, from May to September of 2015, where in total, without any processing of the data, there are 2 756 101 instances. Each instance is considered as an action in the website and is represented by the interaction made by a user - identified by its visitor ID - to an item - also represented by an item ID. This interaction is an event in the form of *view*, *add to cart* or *transaction*, and the exact timing is also recorded.

In the items' details file, each instance corresponds to a shopping item and its characteristics. Each item has a unique ID, a property and its corresponding value, and also a timestamp. The only unhashed properties for an item are the item's availability in the website and the category, and in this work, only the category was considered in the experiments. The timestamp associated to a category shows that an item can change of category over time, but for consistency purposes it was decided to keep the first category value recorded in the database - e.g. a shopping item

[1] `https://retailrocket.net`
[2] `https://www.kaggle.com/retailrocket/ecommerce-dataset`

‘denim jeans’, that is firstly introduced in the ‘trousers’ category, and that changes later to the ‘skirts’ category, would be kept with the original category ID ‘trousers’.

A first decision in the data processing phase, was to consider only the items to which there is access to additional information about its categorical identification, thus the actions with items not meeting this criteria were disregarded from the logs. Therefore, each action is formally characterized as {timestamp, visitor, event, item, category}.

### 4.1.1 Sessions Organization

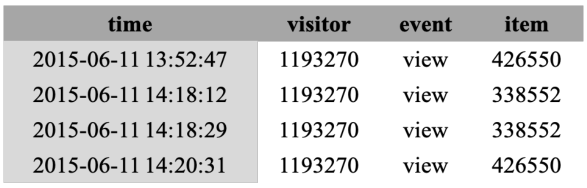
In order to have the dataset organized by different sessions in the website, the corresponding history of actions from each user was partitioned. The criteria used was that after 3 minutes of user inactivity in the website a different user session is to be considered. This separation frequently leaves sessions containing only few actions, and for the purpose of a sequence modeling task, a session has to have a minimum of two actions. Thus, users sessions not meeting this criteria were discarded.

With the dataset split through sessions, and by analysing their content, some inconsistencies were found. In general, the sequences have an abnormal amount of consecutive actions with exactly the same event, item and category. Hence, it was decided that if these repeated actions are separated from each other by less than a minute, these are considered as duplicated entries and are filtered out from the dataset. From Table 4.1, the resulting dataset has a new total of 297 868 actions (a reduction of 89%), through 17 992 sessions, from 212 434 different users.

| | Pre-Processing | Post-Processing | % Reduction |
|---|---|---|---|
| Total number of actions | 2 756 101 | 297 868 | 89% |
| Distinct number of users | 1 407 580 | 212 434 | 85% |
| Distinct number of items | 235 061 | 23 329 | 90% |

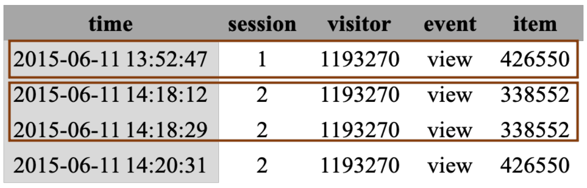
Table 4.1: Dataset statistics before and after data processing.

To exemplify the decisions made so far, Figure 4.1a showcases a set of actions from a certain visitor of the website. The first step taken is to apply the time difference criteria of 3 minutes to divide the actions through different sessions, which resulted into two separate sessions, session 1 and 2, as it is displayed on Figure 4.1b. Considering that the first session contains only an action, it has to be dropped from the dataset. Additionally, in session 2, the first two entries happen to be separated by seconds and have the exact same elements for all action components, which leads to the elimination of this duplicated entry. The final filtered sample is shown on Figure 4.1c.



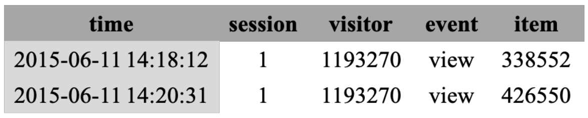
| time | visitor | event | item |
|---|---|---|---|
| 2015-06-11 13:52:47 | 1193270 | view | 426550 |
| 2015-06-11 14:18:12 | 1193270 | view | 338552 |
| 2015-06-11 14:18:29 | 1193270 | view | 338552 |
| 2015-06-11 14:20:31 | 1193270 | view | 426550 |

(a) Visitor actions prior sessions division.



| time | session | visitor | event | item |
|---|---|---|---|---|
| 2015-06-11 13:52:47 | 1 | 1193270 | view | 426550 |
| 2015-06-11 14:18:12 | 2 | 1193270 | view | 338552 |
| 2015-06-11 14:18:29 | 2 | 1193270 | view | 338552 |
| 2015-06-11 14:20:31 | 2 | 1193270 | view | 426550 |

(b) Visitor actions post sessions division.



| time | session | visitor | event | item |
|---|---|---|---|---|
| 2015-06-11 14:18:12 | 1 | 1193270 | view | 338552 |
| 2015-06-11 14:20:31 | 1 | 1193270 | view | 426550 |

(c) Visitor actions after dropping the single-action session and a duplicated action.

Figure 4.1: Showcase of steps taken to process the actions logs.

After the data cleaning steps taken throughout the totality of the dataset, it is worth mentioning that there is still a high number of sessions with repeating values for the action's components throughout the sequences, but at this stage it was not possible anymore to understand if the repetitions were intentional or not - it may be case that the visitors of the website are usually this indecisive and do frequently come back to check the same element. Thus, the dataset did not suffer any further changes.

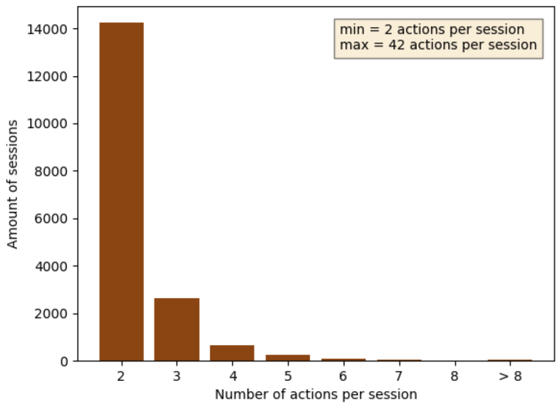


Figure 4.2: Distribution of actions per session.

As it can be seen in Figure 4.2, the resulting sessions are predominantly composed by 2 actions, and sessions with more than 5 actions are rare to occur.

### 4.1.2 Characterization of Action Components

As already stated, an action is characterized by {timestamp, visitor, event, item, category}. In a sequence modeling context, the components of the action to be modeled are: the event, the item, and the item's category. Each of the following sub-sections contains a detailed description of each component, including an analysis of the values distribution through the actions logs, and other relevant details.

#### 4.1.2.1 Event component

An event can be of the type *view*, *add to cart* or *transaction*. Figure 4.3 focuses on the event's type present through the different actions. The interactions are extremely imbalanced, where 98% are clicks to *view* an item, and very few are buying interactions of the type *add to cart* and *transaction*, but whenever an item has an *add to cart* event in a session, the purchasing intent is often concluded by a *transaction* for that item afterwards.

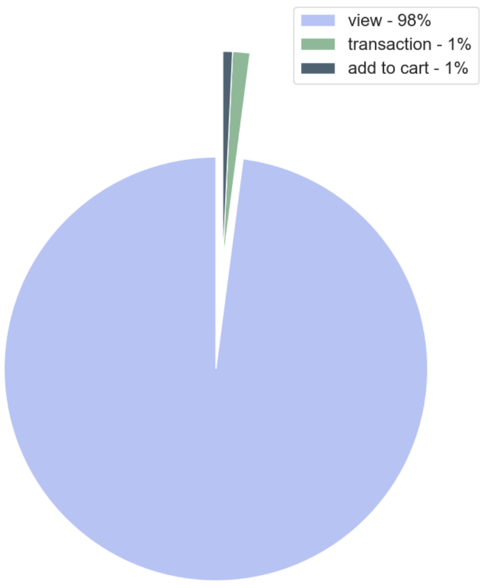
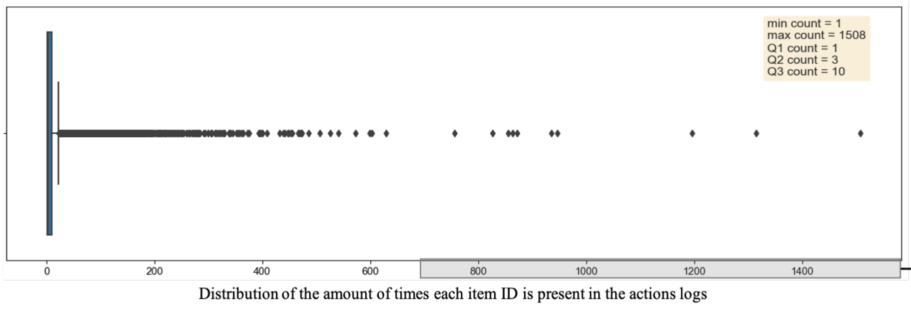


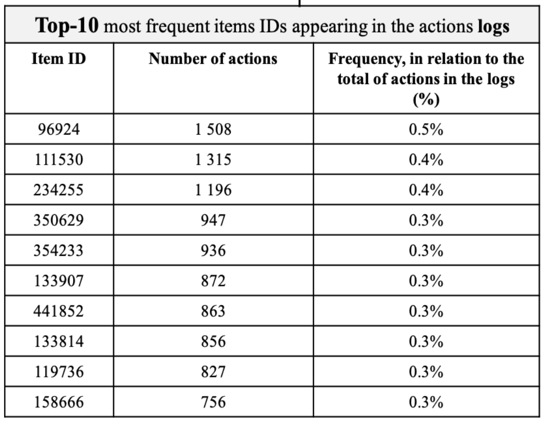
Figure 4.3: Events type distribution in the actions logs.

#### 4.1.2.2 Item component

In this sub-section, the frequency distribution of the 23 329 different items in the catalogue is analyzed. On Figure 4.4, one can quickly notice that most items are individually present in few of the actions in the logs. In fact, 25% (Q1) of the items was interacted just once in the website, and 75% (Q3) less than 10 times. On the other hand, there is a fraction of items that was clicked frequently, such as the Item ID ‘96924’ with a presence in 1 508 different actions, nevertheless, from the table shown in the same figure, it can be seen that this high frequency corresponds to a very small portion of 0.5% to the whole set of actions, and all top-10 most frequent items in the logs amount to a small percentage of 3.4% actions. That being said, it can be concluded that the dispersion of the different IDs through the existing actions is not severely imbalanced.

In contrast, when focusing on actions divided by the users sessions, the diversity of items IDs throughout each session is, in general, rather reduced. From the 17 992 available sessions, 33% of the sessions have the same item ID throughout the complete sequence of actions, and 45% of the sessions contain, at least, two actions with the exact same item ID.





**Top-10** most frequent items IDs appearing in the actions **logs**

| Item ID | Number of actions | Frequency, in relation to the total of actions in the logs (%) |
|---|---|---|
| 96924 | 1 508 | 0.5% |
| 111530 | 1 315 | 0.4% |
| 234255 | 1 196 | 0.4% |
| 350629 | 947 | 0.3% |
| 354233 | 936 | 0.3% |
| 133907 | 872 | 0.3% |
| 441852 | 863 | 0.3% |
| 133814 | 856 | 0.3% |
| 119736 | 827 | 0.3% |
| 158666 | 756 | 0.3% |

Figure 4.4: Boxplot showcasing the distribution of the items IDs frequencies in the actions logs. The table below zooms in the most frequent IDs from the plot and displays additional details.

#### 4.1.2.3 Category component

Each item present in the catalogue derives from a category, which can also derive from an upper-level parent category. This taxonomy was also provided in a separate file, in a tree format with 5 different levels of categories. To demonstrate the structure, a simplified tree is pictured in Figure 4.5. The blue color scheme is used to illustrate the different levels in the tree, going from darker to lighter blue tones, while going from higher to lower levels. The categories are

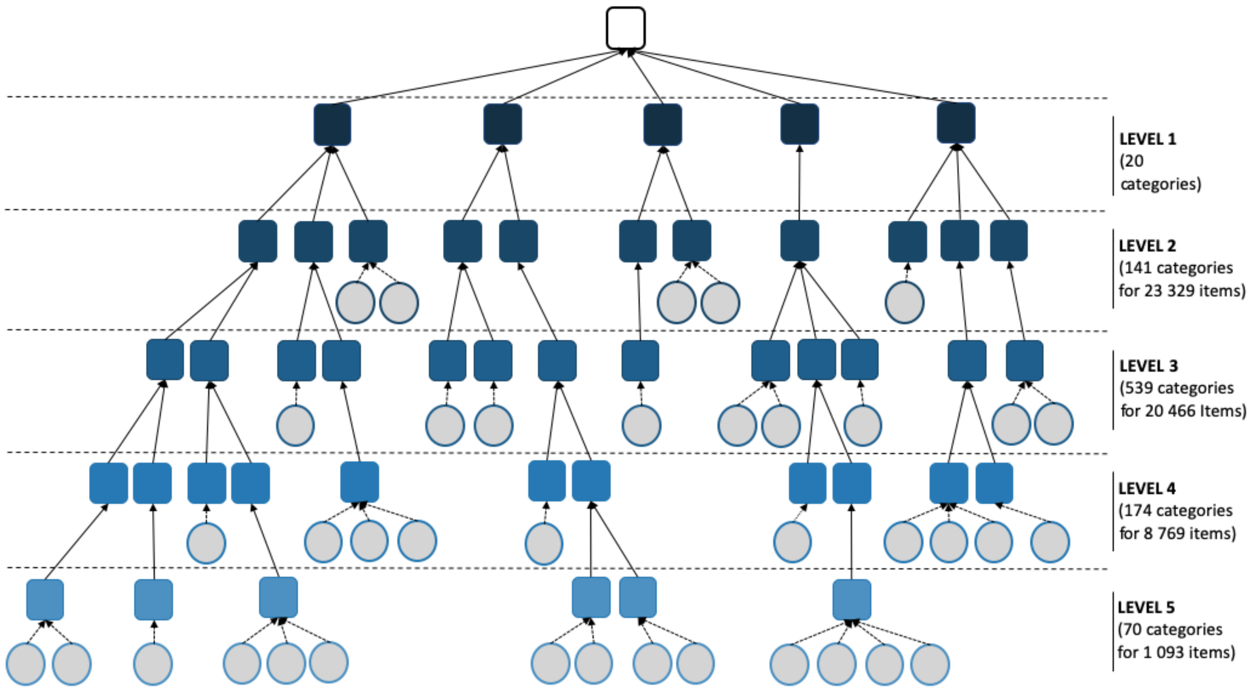
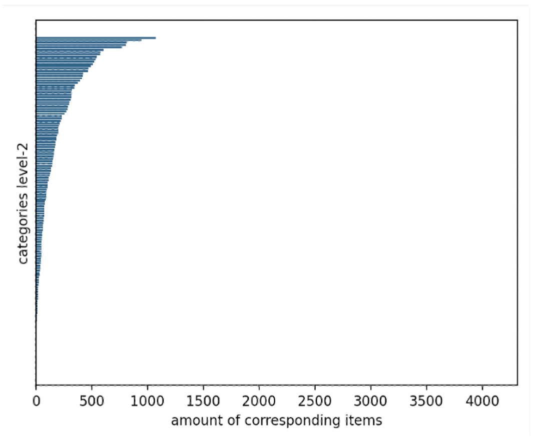


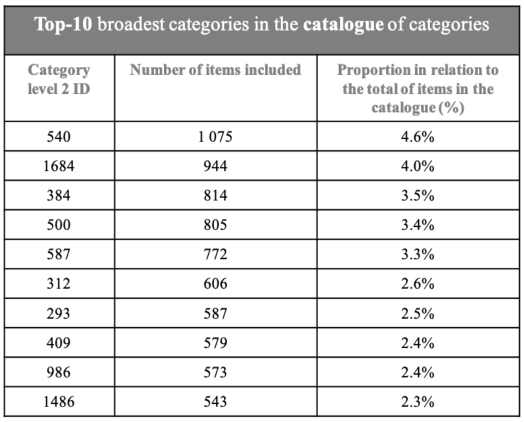
Figure 4.5: Dataset taxonomy with 5 levels, that displays the relations between the items (represented by a circle) and the corresponding category on each level (represented by a square).

represented by a square form and the items by grey circles. The root category does not have any actual categorical representation, it is only integrated to help understand the hierarchy visually. As already mentioned, the categories identifications are hashed, however, one can consider as an example in a fashion e-commerce setting, the shopping item 'necklace' as being integrated in the category level 3 'jewelry', which will be integrated in the category level 2 'accessories' that is also integrated in the super category level 1 'women'.

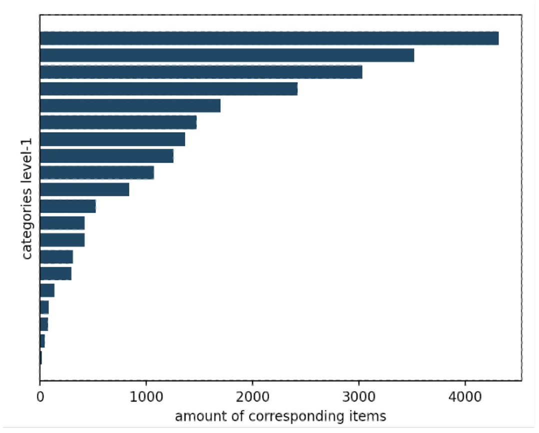
The level 1 categories can have up to four levels of descendants, and an item belongs to a single category from any of the levels. Consequently, an item can be represented by its formal category ID and also by the corresponding parent categories IDs from the same lineage. The only levels that are uniformly represented through all the items are the top two levels, level 1 and level 2. In fact, in level 3, just 88% of the available items have a representation in this category level. In level 4, 38% and in level 5, only 5% of the items.

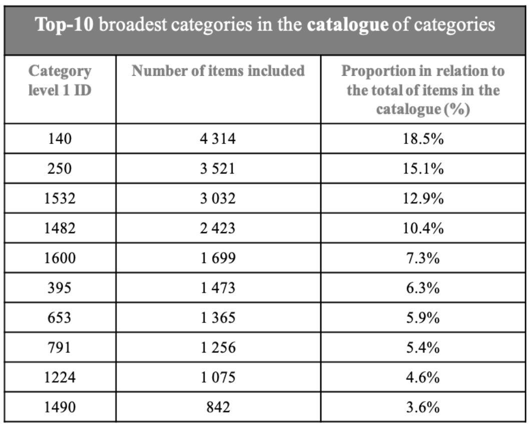
The totality of the available 23 329 items can, at least, be represented by both top two levels. To evaluate the distribution of the number of items included in the categories in these top two levels, the bar graphs in Figures 4.6 illustrate how the associations are dispersed, where each bin corresponds to a single category. One can quickly notice how uneven the distribution is: in both cases there are too many items corresponding to the same category, and other categories with too few items. In level 1, where there is a smaller set of categories, the items understandably





| **Top-10** broadest categories in the **catalogue** of categories | | |
|---|---|---|
| Category level 2 ID | Number of items included | Proportion in relation to the total of items in the catalogue (%) |
| 540 | 1 075 | 4.6% |
| 1684 | 944 | 4.0% |
| 384 | 814 | 3.5% |
| 500 | 805 | 3.4% |
| 587 | 772 | 3.3% |
| 312 | 606 | 2.6% |
| 293 | 587 | 2.5% |
| 409 | 579 | 2.4% |
| 986 | 573 | 2.4% |
| 1486 | 543 | 2.3% |





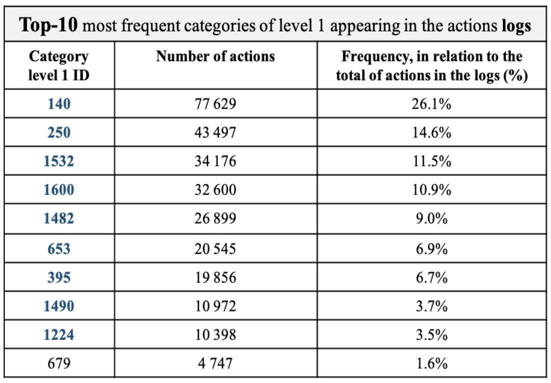
| **Top-10** broadest categories in the **catalogue** of categories | | |
|---|---|---|
| Category level 1 ID | Number of items included | Proportion in relation to the total of items in the catalogue (%) |
| 140 | 4 314 | 18.5% |
| 250 | 3 521 | 15.1% |
| 1532 | 3 032 | 12.9% |
| 1482 | 2 423 | 10.4% |
| 1600 | 1 699 | 7.3% |
| 395 | 1 473 | 6.3% |
| 653 | 1 365 | 5.9% |
| 791 | 1 256 | 5.4% |
| 1224 | 1 075 | 4.6% |
| 1490 | 842 | 3.6% |

Figure 4.6: Distribution of the amount of items per category, regarding the level 1 and level 2 categories present in the categories catalogue.

concentrate more in each category, but half of the categories hold up individually to a high amount of more than 1 000 items. In level 2, although the range of categories is much wider, the items still concentrate in a sub-set of the categories, where a single category can contain up to 1 075 items, as is the case of the category with ID ‘540’.

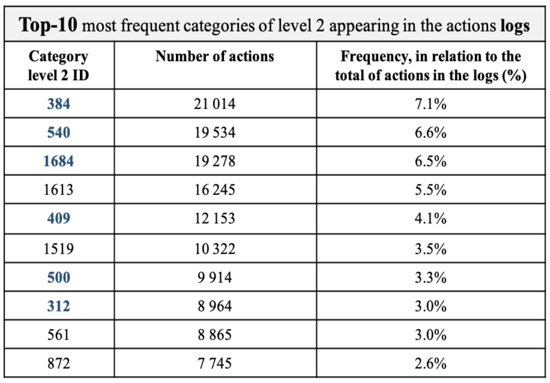
On Figure 4.7, the top-10 most frequent categories IDs, from levels 1 and 2, appearing in the logs are displayed. As it would be expected, these popular categories are mostly the same ones that also hold a wide range of items from the catalogue. The categories IDs that follow this correspondence, between the broadest categories from the tables on Figure 4.6 and the most common categories in the logs (Figure 4.7), are highlighted in blue color in both tables.

Regarding the actions’ distribution throughout the users sessions, the diversity of categories IDs in each session is nearly non existent. From the 17 992 sessions, 81% have the same category ID, and when taking into account only the category levels available to all items, the levels 1 and 2, the percentage of sessions having the same categories values is understandably higher, with 83% in regard to level 2 and 93% for level 1.



| **Top-10** most frequent categories of level 1 appearing in the actions **logs** | | |
|---|---|---|
| **Category level 1 ID** | **Number of actions** | **Frequency, in relation to the total of actions in the logs (%)** |
| **140** | 77 629 | 26.1% |
| **250** | 43 497 | 14.6% |
| **1532** | 34 176 | 11.5% |
| **1600** | 32 600 | 10.9% |
| **1482** | 26 899 | 9.0% |
| **653** | 20 545 | 6.9% |
| **395** | 19 856 | 6.7% |
| **1490** | 10 972 | 3.7% |
| **1224** | 10 398 | 3.5% |
| 679 | 4 747 | 1.6% |

(a) Top-10 frequent categories level 1



| **Top-10** most frequent categories of level 2 appearing in the actions **logs** | | |
|---|---|---|
| **Category level 2 ID** | **Number of actions** | **Frequency, in relation to the total of actions in the logs (%)** |
| **384** | 21 014 | 7.1% |
| **540** | 19 534 | 6.6% |
| **1684** | 19 278 | 6.5% |
| 1613 | 16 245 | 5.5% |
| **409** | 12 153 | 4.1% |
| 1519 | 10 322 | 3.5% |
| **500** | 9 914 | 3.3% |
| **312** | 8 964 | 3.0% |
| 561 | 8 865 | 3.0% |
| 872 | 7 745 | 2.6% |

(b) Top-10 frequent categories level 2

Figure 4.7: Tables displaying the top most frequent categories IDs appearing on the actions logs.

## 4.2 Evaluation Metrics

Each sequence modeling task in this work is taken as a classification problem. As mentioned in Section 3.2, the output probabilities from both the N-th order MC model and any RNN, are ordered by descending order and the first position in the rearranged catalogue of elements corresponds to the predicted next-element for the session. Hence, to assess the quality of the trained model, the predictions made in a separate portion of users sessions, the test sessions, are evaluated using classification metrics. These metrics measure the amount of predictions that were correctly and incorrectly classified as being the next-element in the different sessions. The most intuitive classification metric is the Accuracy, that simply considers a ratio between the correctly classified test sessions to the total of test sessions instances, as in Equation 4.1.

$$\text{Accuracy} = \frac{\#\text{ correctly classified test sessions}}{\#\text{ test sessions}} \tag{4.1}$$

The underlying goal of this work is to predict solely the next element in the session, rather than a set of next elements. Nonetheless, to assess the relevance of the predictions made, the number of positions to consider in the ordered output catalogue can be expanded, in such manner that a good classifier model is expected to place the true next-element in the top positions of the list. To evaluate this ranking approach, rank accuracy metrics Recall@K and MRR are used.

$$\text{Recall@K} = \frac{\#\text{ true next element within top-K output list}}{\#\text{ test sessions}} \tag{4.2}$$

Recall@K (or R@K), is the ratio between the number of test sessions predictions that contained the true next-element amongst the top-K elements in the output list to the total of test sessions predictions made, as shown in Equation 4.2. This metric does not consider the ranking of the predicted element, it only takes into account if it is within the top-K set of elements or not. Whereas, MRR, which stands for Mean Reciprocal Rank, considers the ranking position of the element in the list. MRR is translated in Equation (4.3), where *rank* is the position of the element in the output list from each test session prediction, and N is the amount of test sessions instances.

$$\mathrm{MRR} = \frac{1}{\mathrm{N}} \sum_{i=1}^{\mathrm{N}} \left( \frac{1}{rank_i} \right) \tag{4.3}$$

Taking into account the e-commerce context in which this work is integrated, the target element should be amongst the very top of the relevant elements, hence, the K to be used will not consider a long list of top-elements, and will therefore take the values of 1, 5 and 10. When using R@1, the metric is equal to the regular Accuracy metric.

## 4.3 Experimental Evaluation

This section evaluates the performance of all created models based on different carried experiments. Firstly, Section 4.3.1 demonstrates the preparation of the input data made for the distinct models' architectures. Next, with the data fully prepared to be fed to the models, on Section 4.3.2 the experiments executed, and the results from those experiments are then discussed in Section 4.3.3.

### 4.3.1 Dataset Preparation

To train and evaluate any of the models, the whole set of variable-length users sessions was divided into train and test sets, using a split proportion of 65:35. For hyper-parameters tuning of the RNN-based models, the train set was also divided into train and validation sets, with a split proportion of 80:20, where after selecting the hyper-parameter values, the model is re-trained over the complete train set and assessed in the holdout test set portion of users sessions. The division was done by assigning randomly different visitors to the different sets, so that sessions from the same user are not split up. On Table 4.2, one can see the details of the created train and test sets.

The different models have different requirements for the input sessions, hence the structure of the training sessions had to be tailored to each of the models used. For the N-th order

| | **Train sessions** | **Test sessions** |
|---|---|---|
| Distinct number of users | 74 351 | 16 241 |
| Distinct number of items | 20 321 | 16 135 |
| Distinct number of categories (any level) | 955 | 905 |
| Distinct number of level 1 categories | 20 | 20 |
| Distinct number of level 2 categories | 140 | 137 |
| Total number of actions | 27 250 | 14 790 |
| Total number of sessions | 11 697 | 6 323 |
| Avg length sessions | 2.0 | 2.0 |

Table 4.2: Dataset split into train-test sets, with a 65:35 proportion.

Markov chain model, the sessions are divided into fixed-size sequences of N elements. As for the RNN models, the training and validation sessions are populated using a moving window throughout each session. The sub-tables from Table 4.3 demonstrate the different sessions transformations used to train both deep and non-deep learning models. On the fist table, 4.3a, an example training session is displayed, from which it is transformed into fixed-size sequences of two elements for a bi-grams model in Table 4.3b. For a tri-grams model, the sample session would remain unchanged, and for larger N values this sequence would not be considered for the train set. For an RNN model, in the last Table 4.3c, the sample session is populated with a moving window of 1 element, which results in the creation of an additional session.

| Session | Elements' sequence | Next-element |
|---|---|---|
| 1 | [ element ID **1**, element ID **2** ] | element ID **3** |

(a) Sample training session.

| Session | Elements' sequence | Next-element |
|---|---|---|
| 1 | [ element ID **1**] | element ID **2** |
| 2 | [ element ID **2**] | element ID **3** |

(b) Sample training session from Table 4.3a, turned into fixed-size sequences for a bi-grams model.

| Session | Elements' sequence | Next-element |
|---|---|---|
| 1 | [ element ID **1** ] | element ID **2** |
| 2 | [ element ID **1**, element ID **2** ] | element ID **3** |

(c) Sample training session from Table 4.3a, populated with a moving window of 1 element for an RNN model.

Table 4.3: Sample training sessions input transformation for a bi-grams and RNN model.

The details for the resulting train sets for each model are displayed on Table 4.4, naturally

**Train sessions transformed**

| | 2-grams | 3-grams | 4-grams | 5-grams | RNN |
|---|---|---|---|---|---|
| Total number of actions | 31 106 | 11 568 | 5 632 | 3 505 | 37 113 |
| Total number of sessions | 15 553 | 3 856 | 1 408 | 701 | 15 398 |
| Avg length sessions | 2.0 | 3.0 | 4.0 | 5.0 | 2.4 |

Table 4.4: Dataset split into train-test sets, with a 65:35 proportion.

the distinct number of users, items and categories prevails the same from Table 4.2, and hence are not shown in this table. One can see the decreasing difference for the number of available sequences to train N-th order Markov Chains with larger N values, and on the other hand, for the RNN model the sessions transformation result in more 2 842 sessions, an increase of 24% instances in the training set.

To evaluate the tuned models, the total 6 323 variable-length test sessions always remain unchanged in order to have a fair ground for comparison.

### 4.3.2 Experimental Methodology

An action is an aggregation of different components, as exemplified in Figure 4.8. In this work, the proposed problem to predict the next-action after a sequence of actions, is broken down into individual sequence prediction tasks, as it is pictured in Figure 4.9.

This section defines the experiments executed on the evaluation of the models for each prediction task. The experiments definitions are introduced through three sub-sections. The first set of experiments, on Section 4.3.2.1, looks at the action's components as being non-related to each other, and on a next phase, in Section 4.3.2.2, the existing relations between the item and the category, as well as between the different categories levels, are taken into account as contextual information to help on the prediction of the next item and category. The last section, Section 4.3.2.3, formulates a modified version of the test sessions to evaluate the models performances under irregular conditions.

The results for each set of experiments are shown in the upcoming Section 4.3.3.

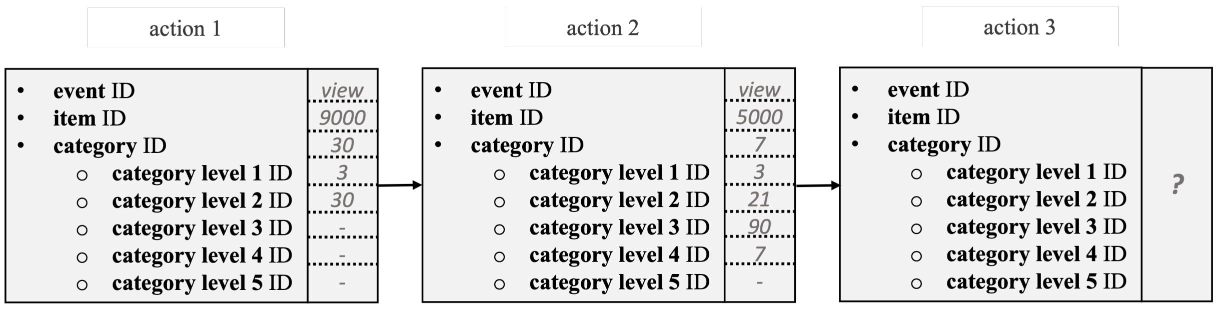


Figure 4.8: Sequence of actions.

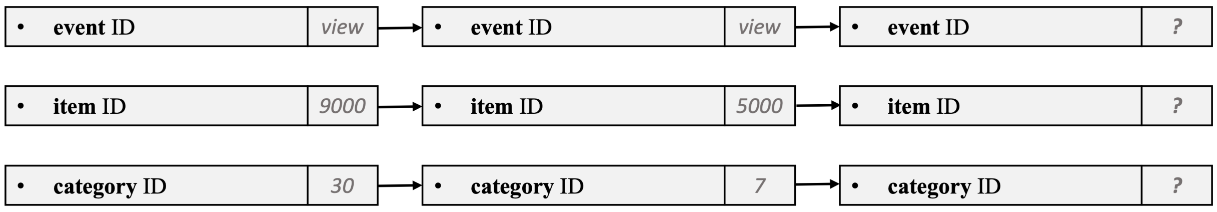


Figure 4.9: Sequence of decomposed actions.

#### 4.3.2.1 Next Element Prediction

The available components of an action to be predicted are: the item, the event, and the category. Taking into account that the dataset is heavily imbalanced in terms of events, as it was shown on Section 4.1.2.1, with 98% of events being *view* clicks, this is a component not worth modeling. Hence, the components to be predicted are the next item and next category. To that end, the baseline approach presented in Section 3.2.1, the N-th order Markov Chain in Section 3.2.2 and the RNN approach in Section 3.2.3 are all executed for these two sequence prediction tasks.

The first set of experiments was made with the prediction of the next category, since it has a fair amount of distinct elements, ideal to set benchmark results. For that, the sequential history of categories in a session is taken into account with no regard to any of the ascendants categories' levels, meaning that any of the models tested for this particular task is set to predict a category within a vocabulary containing all possible categories from any level. On a next phase, it was decided to focus on the prediction of a category level available to all items at the thinnest level possible, which could only be level 2, and for this case, the range of different categories to consider is much shorter. Lastly, the focus is shifted to the prediction of the next item ID, which has a substantial larger catalogue of distinct elements to consider.

#### 4.3.2.2 Next Element Prediction using Contextual Information

For the defined sequence prediction tasks, the contextual information coming from the existing taxonomic relations is now to be integrated in the training and evaluation processes of the RNN-based models, as it was explained in Sections 3.2.3.1 and 3.2.3.2.

For the prediction of the next category of level 2, the relation between the level 2 and level 1 categories is used. The objective is to guide the model to output a category level 2 that has the same level 1 category as of the true category of level 2. For the prediction of the next item ID, the taxonomic relation to be used is between the item and its corresponding category of level 2.

#### 4.3.2.3 Next Element Prediction on Irregular Test Sessions

To test the robustness of the already trained models under irregular circumstances, the sessions in the test set were altered.

Considering that there is a considerable amount of sessions with repeating elements through time (Sections 4.1.2.2 and 4.1.2.3), the test sessions being used so far were filtered to include consecutively different elements from action to action. That said, for the category of level 2 prediction, each test session contains consecutively different categories of level 2, and for the item ID prediction, different items IDs.

### 4.3.3 Experimental Results

This section showcases and discusses the results achieved through the execution of the previously defined experiments on Section 4.3.2.

#### 4.3.3.1 Next Category Prediction

Each item belongs to a single category, which can also have up to 4 ascendants categories. A first objective is to predict the next category, by considering the history of categories in a session with no regard to the level in which the categories are inserted. The model works with a vocabulary containing all possible categories values from any level, 1 244 distinct values. The results of this experience are shown on the left-side of Table 4.5.

From the analysis of the dataset, it was stated that the users often browse in the same category throughout an entire session. That said, the defined baseline model that predicts, as the next category, the category from the immediate previous action in the session is expected to perform well. Given the peculiar structure of the data, the baseline model reaches a high R@1 accuracy of 0.797.

| | next-**category any level** prediction | | | | next-**category level 2** prediction | | | | next-**item** prediction | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Baseline model** | R@1 | R@5 | R@10 | MRR | R@1 | R@5 | R@10 | MRR | R@1 | R@5 | R@10 | MRR |
| Assign previous value | 0.797 | 0.797 | 0.797 | 0.797 | 0.869 | 0.869 | 0.869 | 0.869 | 0.350 | 0.350 | 0.350 | 0.350 |
| **Statistical models** | R@1 | R@5 | R@10 | MRR | R@1 | R@5 | R@10 | MRR | R@1 | R@5 | R@10 | MRR |
| 2-grams | 0.772 | 0.898 | 0.918 | 0.827 | 0.860 | 0.951 | 0.968 | 0.900 | 0.279 | 0.436 | 0.456 | 0.344 |
| 3-grams | 0.163 | 0.176 | 0.177 | 0.169 | 0.185 | 0.200 | 0.200 | 0.192 | 0.098 | 0.102 | 0.102 | 0.099 |
| 4-grams | 0.048 | 0.051 | 0.051 | 0.049 | 0.057 | 0.060 | 0.060 | 0.058 | 0.033 | 0.033 | 0.033 | 0.033 |
| 5-grams | 0.020 | 0.021 | 0.021 | 0.021 | 0.023 | 0.024 | 0.024 | 0.023 | 0.014 | 0.014 | 0.014 | 0.014 |
| **Recurrent Neural Networks** | R@1 | R@5 | R@10 | MRR | R@1 | R@5 | R@10 | MRR | R@1 | R@5 | R@10 | MRR |
| SimpleRNN | 0.780 | 0.904 | 0.926 | 0.835 | 0.870 | 0.955 | 0.973 | 0.908 | 0.312 | 0.445 | 0.483 | 0.374 |
| LSTM | 0.786 | 0.908 | 0.928 | 0.838 | 0.874 | 0.957 | 0.975 | 0.911 | 0.320 | 0.453 | 0.500 | 0.389 |
| mLSTM | 0.785 | 0.908 | 0.928 | 0.838 | 0.874 | 0.957 | 0.975 | 0.911 | 0.321 | 0.460 | 0.500 | 0.389 |
| GRU | 0.783 | 0.908 | 0.929 | 0.838 | 0.875 | 0.957 | 0.975 | 0.911 | 0.322 | 0.455 | 0.499 | 0.385 |

Table 4.5: Evaluation on the prediction of the attributes of the next-action: category of any level, category of level 2 and item

The N-th order Markov chain model considers the frequency of co-occurrence of sequences of elements, in order to infer which element might follow. Once again, due to the nature of the data with most of the sessions having a length of two actions and where these have frequently the same values, the model excels particularly well when using the frequency of pairs of actions - bi-grams - to infer about the next transition. Naturally, when increasing the look-back window, the co-occurrence of longer sequences becomes scarcer, and the performance of the MC with higher orders decreases.

Each type of Recurrent Neural Network - RNN, LSTM, mLSTM, GRU - was trained under the same circumstances for comparison purposes. The chosen values for the architecture's parameters are displayed in Table 4.6. Each model was trained over 100 epochs with 70 neurons, and with Adam optimizer (Kingma and Ba, 2017). For this sequence modeling case, as one can tell from the results above, the performances do not diverge significantly between the different types of RNN-based networks used.

| Nr. Neurons (RNN layer) | Nr. Training Epochs | Batch Size | Optimizer | Learning Rate | Dropout Rate |
|---|---|---|---|---|---|
| 70 | 100 | 16 | Adam | 0.050 | 0 |

Table 4.6: Parameters values for any RNN model, for next category prediction.

#### 4.3.3.2 Next Category Level 2 Prediction

In this experiment, the aim is to predict the next level 2 category in a session. All models were trained with the same settings as before (Table 4.6), yet considering the history of categories from level 2, with a vocabulary of 141 distinct categories. The results of this experience are shown on the middle-side of Table 4.5.

Since now it is strictly being considered a higher and also broader level in the hierarchy of categories, it is expected that the results are equal or better than previously - there are categories that were being considered from levels 3, 4 or 5, that are now considered as being from the same category of level 2, which leads to more actions having the same category value. As expected, the performance of the baseline model is even better, and just as previously, the Markov chain executes well when considering bi-grams.

Any of the RNN types used was trained with the same hyperparameters values from Table 4.6. Naturally, any of these models performs better than before, given that the number of distinct values is much smaller, and in addition, as it was stated in Section 4.1.2.3, when focusing on a higher level in the taxonomy tree the frequency of repeated values in a session is higher, which simplifies the data to be learned. From the three RNN models used, the GRU proved to perform slightly better and was therefore used in further experiments.

#### 4.3.3.3 Next Category Level 2 Prediction using Contextual Information

In an attempt to help on the prediction of the category level 2, the complementary information coming from a higher category level was thought to be used as an advantage. The objective is to influence the level 2 prediction towards a value within the same family as the expected category level 2 - the categories are considered as being part of the same family group if these share the same level 1 parent. This contextual information is incorporated as shown in the defined RNN architecture in Section 3.2.3. Through the modified loss function, introduced in Equation 3.3, the parents of the predicted and expected categories of level 2 are compared, and when equal, a penalization value $p$ is applied to the categorical cross-entropy result, and when not equal, the categorical cross-entropy result remains unchanged.

Table 4.7 displays the results of a GRU model using the different $p$ values, starting from $p$ value of 1.0 that is equivalent to a GRU with a regular categorical cross-entropy loss function. One can see that the metrics results, for the different penalizations used, do not differ greatly from each other, but the $p$ value that achieves slightly the best results for this prediction task is of 0.1, and is therefore the value to be used in the modified loss function for the GRU model.

| | next-**category level 2** prediction | | | | next-**item** prediction | | | |
|---|---|---|---|---|---|---|---|---|
| **penalization value** $p$ | R@1 | R@5 | R@10 | MRR | R@1 | R@5 | R@10 | MRR |
| $p$ = 1.0 | 0.875 | 0.957 | 0.975 | 0.911 | 0.322 | 0.455 | 0.499 | 0.385 |
| $p$ = 0.5 | 0.875 | 0.957 | 0.975 | 0.911 | 0.321 | 0.455 | 0.499 | 0.385 |
| $p$ = 0.1 | 0.875 | 0.958 | 0.978 | 0.912 | 0.321 | 0.456 | 0.502 | 0.386 |
| $p$ = 0.01 | 0.874 | 0.957 | 0.976 | 0.911 | 0.321 | 0.457 | 0.502 | 0.388 |

Table 4.7: GRU using the modified loss function with different penalization values.

With the inclusion of this new loss function, it was possible to assess that the model is in most cases benefiting the categories belonging to the same group of the expected category level 2, by detecting in each output ranking-list that its top positions are mainly constituted by sibling categories. To demonstrate, a sequence from a random visitor is fed to the model and the output ranking lists using both the regular loss function and the modified loss function, are depicted in Figure 4.10a and Figure 4.10b, respectively. The expected output has its output position highlighted with green color, and the positions of the categories that belong to the same family as the expected value are highlighted in blue color.

On Figure 4.10b, although the output list contains elements that belong to other family groups, the higher ranked positions are exclusively composed by categories that have the same parent as of the expected value. Whereas when using the regular categorical cross-entropy loss function, in Figure 4.10a, the output list can contain a similar range of categories, but its top positions are not entirely made up by categories from the same family group as the expected category. Comparing both 'GRU' and 'GRU + modified loss' results in Table 4.8, the difference is not that notorious. In fact, the number of times that the model hits and misses the expected category value is very much similar, and even when expanding the top-positions to consider -

| | next-**category level 2** prediction | | | | next-**item** prediction | | | |
|---|---|---|---|---|---|---|---|---|
| **Recurrent Neural Networks** | R@1 | R@5 | R@10 | MRR | R@1 | R@5 | R@10 | MRR |
| GRU | 0.875 | 0.957 | 0.975 | 0.911 | 0.322 | 0.455 | 0.499 | 0.385 |
| GRU + modified loss | 0.875 | 0.958 | 0.978 | 0.912 | 0.321 | 0.457 | 0.502 | 0.388 |
| GRU + filtered group scores | 0.917 | 0.983 | 0.997 | 0.938 | 0.350 | 0.512 | 0.592 | 0.395 |
| GRU + modified loss + filtered group scores | 0.924 | 0.991 | 0.999 | 0.952 | 0.359 | 0.522 | 0.594 | 0.438 |

Table 4.8: Evaluation of the GRU models that use side information to predict the next category level 2 and item.

top-5 and top-10 -, the results are not much improved.

As a further experiment using the provided relational information, the output ranking list, containing the whole catalogue of categories level 2, can be narrowed down in order to include solely the categories that do share the same parent level 1, as the one from the expected category. This way, noisy predictions with categories values from other groups are forcibly left out from the output catalogue list. This filtering process will hereafter be referred to as filtered group scores step.

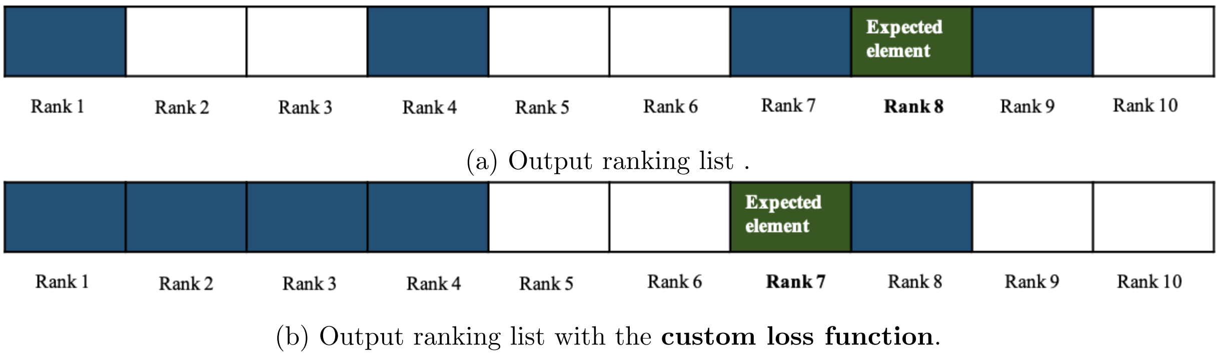


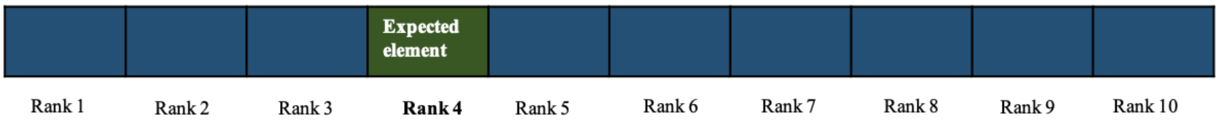
Figure 4.10: Difference between output lists using different loss functions.

The output ranking lists for the same example session would now be as in Figure 4.11a, when using the regular cross-entropy loss function, and as in Figure 4.11b, when using the modified loss function version. As it can be seen, in both cases, after applying the filtered group scores step, the output lists now contain only elements belonging to the same family group, since there are only blue color positions and no positions marked in white.

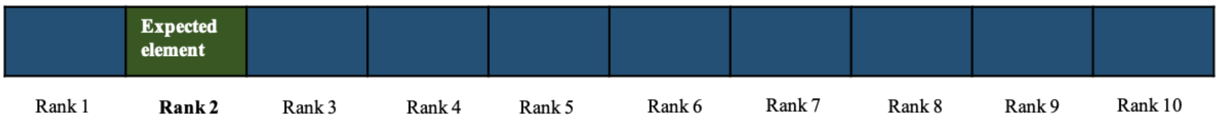
By forcibly excluding any non-related categories, it is assured that the list exclusively comprises categories within the same family. As one can tell from the results shown on Table 4.8, the GRU performance using the combination of both steps - 'GRU + modified loss + filtered group scores' - upgrades the model's performance, especially when considering the very top values with R@1 and R@5, by having about a 5% increase in scores. Hence, the last model created is the best tailored to the problem of predicting the next level 2 category.

#### 4.3.3.4 Next Category Level 2 Prediction on Irregular Test Sessions

In general, it can be said that all models tested so far achieve high scores for the category of level 2 prediction, which may be due to the over simplified distribution of categories values in the dataset sessions. Hence, to test if the models created can still perform well enough in irregular conditions, a set of data was put apart, where actions repeatedly with the same



(a) **Filtered group scores** of the output ranking list.



(b) **Filtered group scores** of the output ranking list with the **custom loss function**.

Figure 4.11: Difference between output lists using the filtered group scores step and different loss functions.

| | next-**category level 2** prediction | | | | next-**item** prediction | | | |
|---|---|---|---|---|---|---|---|---|
| **Baseline Model** | R@1 | R@5 | R@10 | MRR | R@1 | R@5 | R@10 | MRR |
| Assign previous value | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **Statistical models** | R@1 | R@5 | R@10 | MRR | R@1 | R@5 | R@10 | MRR |
| 2-grams | 0.049 | 0.655 | 0.781 | 0.314 | 0.339 | 0.536 | 0.561 | 0.420 |
| 3-grams | 0.016 | 0.023 | 0.023 | 0.019 | 0.018 | 0.018 | 0.018 | 0.018 |
| 4-grams | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 |
| 5-grams | 0.001 | 0.001 | 0.001 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 |
| **Recurrent Neural Networks** | R@1 | R@5 | R@10 | MRR | R@1 | R@5 | R@10 | MRR |
| GRU | 0.039 | 0.534 | 0.639 | 0.272 | 0.103 | 0.262 | 0.309 | 0.175 |
| GRU + modified loss + filtered group scores | 0.278 | 0.861 | 0.987 | 0.519 | 0.143 | 0.347 | 0.429 | 0.242 |

Table 4.9: Evaluation on the prediction of the next category level 2 and item, on test sessions with consecutive distinct values.

categories were taken out from the sessions. This step resulted in a high number of single-action sessions, which had to be dropped from the set, and consequently the new test set has only 712 sessions. Nevertheless, it is still considered to be enough data to verify the applicability of the created models. Table 4.9 showcases the results for the irregular set of data. Evidently, the simple baseline model is not suitable for this experiment.

In this sub-set of sequences, looking at the results from R@1 and MRR metrics, the models do not output the true category value in the highest ranks, nevertheless when expanding the top-list to consider, the results are satisfactory for both statistical and simple GRU models. The GRU that uses the taxonomic relations between the categories of level 2 and level 1, notoriously outperfoms all the others, reaching similar results to the previous experience made when using the complete dataset on Section 4.3.3.2, in regard to the R@5 and R@10 results, which leads to the conclusion that using the side information provided by the different grouping of categories is indeed advantageous.

#### 4.3.3.5 Next Item Prediction

The same models used for category prediction are applied to the item ID prediction task. The first experiment aims to predict the next item ID and the results are shown on the right-side of Table 4.5. As one can notice, in general, the scores are meaningfully lower in comparison to the prediction of the category. There are a couple of reasons that may explain the poorer performance: the catalogue of items is much larger, with 23 329 distinct values, and the frequency of repeated items is not as significant as the frequency of repeated categories in the sessions. Nonetheless, as stated in the data analysis Section 4.1.2.2, there is still a considerable amount of sessions where the items are the same throughout the sequence, and for that reason, the baseline model that gives out the next item inline as being the same as previously is applicable to this task as well. In fact, comparing to all the other models, the baseline model reaches the best R@1 result with 0.350.

Considering the N-th order Markov chain, it proves to work better when using short-distance dependencies, as with bi-grams. Since this is a purely statistical model that is significantly dependant on the train set corpus, it is unable to generalize, and so when used with unseen bigger sequences of elements it has demonstrated to be incapable of giving an oriented prediction. Going through the whole set of results, the n-grams models have the lowest outcomes.

| Nr. Neurons (RNN layer) | Nr. Training Epochs | Batch Size | Optimizer | Learning Rate | Dropout Rate |
|---|---|---|---|---|---|
| 10 160 | 70 | 16 | Adam | 0.001 | 0.4 |

Table 4.10: Parameters values for any RNN model, for next item prediction.

The tuned RNN-based models were trained with the specified paratemers values in Table 4.10. Since the volume of distinct elements to consider is much higher in this task, the optional embedding layer was added to the network's architecture, and the number of neurons to train was also greatly increased with a total of half of the number of distinct items IDs. In addition, the neural networks for this task benefit from a dropout layer with a rate of 0.4 prior to the forecasting of the prediction, given that by dropping some neurons from the training phase, the gap between the training and validation results is shorter when using this specific dropout rate value. In the end, the different types of networks do not present a critical difference between each other in terms of results, in any case, the RNN type to be considered in further experiences is also the GRU, since it presents the best results and is simpler to train (there are fewer computations in a GRU than in an LSTM).

#### 4.3.3.6 Next Item Prediction using Contextual Information

On a next experiment, the hierarchical relation between an item and its category level 2 was used, just as in the category prediction when using the hierarchical relation between categories level 1 and 2. In this case, a prediction can be conducted towards an item that belongs to the same category as the expected item. The GRU model is now set to use this contextual information with the same mechanisms as before, using a modified version of the loss function as seen in Equation 3.3, with the best penalization value $p$ of 0.01 as seen on Table 4.7, and by dropping out items belonging to other family groups in the output catalogue with the filtered group scores step. From the results on Table 4.8, the best performing GRU version uses these two additional steps.

#### 4.3.3.7 Next Item Prediction on Irregular Test Sessions

For this experiment, the actions consecutively with the same item ID are taken out from the sessions, which results in 3 134 testing sequences. As one can tell from the right-side of Table 4.9, the results from the statistical models do not differ greatly from the ones on the first item prediction experience on Table 4.5. This may be due to the fact that items repetitions throughout consecutive actions do not occur that often, thus the resulting test set does not differ significantly from before, meaning that the amount of pairs transitions ought to be similar, and therefore the 2-grams model presents robust results. On the other hand, the RNN-based model drops its performance by half. It seems that the model may be too attached to the training data and even the slightest modification to the testing data deviates the performance. Nevertheless, when further analysing the results, the difference between the 'GRU + modified loss + filtered group scores' and the 'GRU', is quite notorious, given that although there are no repeated items, the taxonomic relations between the item and category still exist and naturally impact positively the prediction of an item in the same way.

### 4.3.4 Overview

This chapter presented the experimental evaluation carried out to correctly test the performances of the different models.

Initially, Section 4.1 introduces the e-commerce actions logs dataset used for the implemented models. In Section 4.1.1, the set of actions is processed and organized through users sessions, to be used in the implemented session-based models. In Section 4.1.2, a statistical analysis, e.g. values distribution, is done to each action component.

Section 4.2 presents the different metrics that were used to evaluate the designed models between each other.

Section 4.3 evaluates the performance of all models based on different carried experiments: Section 4.3.1 starts by demonstrating the preparation of the input data for the training of each model. Next, the experiments to be executed are defined over Section 4.3.2, and later on, in Section 4.3.3, the outlined experiments are performed and the results are explored and discussed. For each action component prediction task, the same set of models was tested, which allowed to conclude that the architecture that yielded the most interesting results, for all prediction tasks, was the GRU able to incorporate additional contextual information in the training and evaluation processes. Nevertheless, the simpler statistical models proved to be particularly robust for the next item ID prediction task.

# 5 Conclusions

The underlying aim for this work was to predict the next action a customer is most likely to take after a sequence of actions in a browsing session from an e-commerce website. To that end, distinct approaches for the same sequence prediction problem were explored and compared throughout this dissertation work. This chapter concludes the dissertation by presenting in Section 5.1 a quick overview of the main contributions of this work, and in Section 5.2 a description of some possible ideas to extend and possibly improve the results of this thesis.

## 5.1 Contributions

This work decomposes an action and predicts every component separately, rather than focusing solely on the prediction of the central component of the action, the item. Considering that the catalogue of possible values to consider differs immensely between the different components, this was an interesting way to study and understand which model variations bring the most significant improvements to each action's component prediction task.

To that end, a wide range of models architectures was tested, from non-deep to deep learning techniques, unlike in most recent scientific works in the domain that already benchmark their results based on deep learning methods (Wang et al., 2019). In this work, statistical models were initially explored, namely with N-th order Markov Chain (MC), and for a more sophisticated deep-learning approach, Recurrent Neural Networks (RNN) were used. The limitations of using a regular RNN layer were also reviewed, and consequently, variants of the RNN, as the GRU (Cho et al., 2014) and LSTM (Hochreiter and Schmidhuber, 1997) were tested, as well as a state-of-the-art LSTM version, the multiplicative LSTM (Krause et al., 2016). To fairly assess the performance of the diverse approaches taken, the same experiments were executed for each created model. In fact, the simple statistical approach proved to perform similarly to more modern deep learning approaches. Furthermore, when testing the set of RNN-based models, a significant difference between the use of a simple RNN to other RNN versions was detected, however, between the three more evolved RNN architectures the difference was not notorious.

In a regular e-commerce website, the clickstream logs along with other provided details about the catalogue of items in the website, can be used to derive insightful information. In this work, the way the action's components relate between each other hierarchically was found as relevant, and used as potential advantage in the RNN-based model. This contextual information was incorporated in both training and evaluation processes of the network, and this version of an RNN-based model proved to outperform all others by an interesting margin.

## 5.2 Future Work

For future work, it would be interesting to extend this study in other directions, by trying out new ideas and possibly enhancing the obtained results. A first step for future work would be to join all action's components predictions into a single model that could predict the complete next action. Currently, there are different models optimized to the prediction of a specific action component, thus, it would be interesting to be able to predict each component in parallel, and just as in the initial attempt of a p-RNN (Hidasi et al., 2016), the model could simply return all hidden states from the different created networks. This could result in a rather heavy model to train, thus the next step would be to study different training strategies to come up with a lighter version.

The following ideas focus on the prediction of the next action in a more personalized way for each website visitor. In this work, only session-based models are used, ignoring long-term interests coming from previous sessions of the users, since it is thought to fit better a real-world setting where the consumer behaviour is rapidly changing. Nevertheless, it would be interesting to see the impact of using both long and short term interests of the visitors, to make personalized recommendations to each user. To that end, a similar model to an H-RNN presented by Quadrana et al. (2017) could be applied to the given dataset. Two distinct networks would be used, one that models user's activity across sessions and other that models the sessions. For a returning user, in a new session, the model should take the last hidden state from both networks, in order to predict a good initialization for a new user session. By propagating the user history to help make new predictions, this should result in more personalized recommendations for each specific user. Furthermore, if a new goal for this work would also be to recommend a next action at the start of a brand new session, a potential cold-start problem would also be tackled with this new approach.

In addition, a limitation to bear in mind in e-commerce settings is the heterogeneity of items popularity Resnick and Varian (1997), which is not being specifically modeled in this work. As stated in Section 4.1.2.2, the items popularity is not severely imbalanced, however, in terms of items' categories, there are certain categories that appear much more frequently in history than others. That being said, attention mechanisms (Vaswani et al., 2017) could be used to highlight certain categories, just as in the work presented by Wang et al. (2020), where depending on the user and the time of the year, a different emphasis is given to certain elements. This is a relevant mechanism since different dates can have different impact from user to user, such as the user's birthday.

# Bibliography

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on, 5(2):157–166.*

Chen, S. F. and Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Article No. csla.1999.0128.*

Ching, W. K., Fung, E. S., and Ng, M. K. (2004). Higher-order markov chain models for categorical data sequence. *DOI 10.1002/nav.20017.*

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078.*

Cui, L., Xie, X., and Shen, Z. (2018). Prediction task guided representation learning of medical codes in ehr. *J Biomed Inform.*

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and de Freitas, N. (2013). Predicting parameters in deep learning. *arXiv preprint arXiv:1306.0543.*

Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14(2).*

Fang, H., Zhang, D., Shu, Y., and Guo, G. (2019). Deep learning for sequential recommendation: Algorithms, influential factors, and evaluations. *arXiv preprint arXiv:1905.01997.*

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning.* MIT Press. `http://www.deeplearningbook.org`.

Hancock, J. T. and Khoshgoftaar, T. M. (2020). Survey on categorical data for neural networks. *J Big Data 7, 28.*

He, Q., Jiang, D., Liao, Z., Hoi, S. C. H., Chang, K., Li, E.-P., , and Li, H. (2009). Web query recommendation via sequential query prediction. *ICDE '09. 1443–1454.*

Hidasi, B. and Karatzoglou, A. (2018). Recurrent neural networks with top-k gains for session-based recommendations. *arXiv preprint arXiv:1702.03847.*

Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. (2015). Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:511.06939.*

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

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhubere, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A field guide to dynamical recurrent neural networks.*

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

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization. *arXiv:1412.6980v9.*

Krause, B., Lu, L., Murray, I., and Renals, S. (2016). Multiplicative lstm for sequence modelling. *arXiv preprint arXiv:1609.07959.*

Lin, K. H.-Y., Wang, C.-J., and Chen, H.-H. (2011). Predicting next search actions with search engine query logs. *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology.*

Linden, G., Smith, B., and York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering.

Ludewig, M. and Jannach, D. (2018). Evaluation of session-based recommendation algorithms. *arXiv preprint arXiv:1803.09587.*

Lv, Y., Zhuang, L., and Luo, P. (2019). Neighborhood-enhanced and time-aware model for session-based recommendation. *arXiv:1909.11252.*

Markov, A. (2006). Extension of the law of large numbers to quantities, depending on each other (1906). reprint. *Journal Électronique d'Histoire des Probabilités et de la Statistique [electronic only]*, 2(1b):Article 10, 12 p., electronic only–Article 10, 12 p., electronic only.

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

Mcfee, B. and Lanckriet, G. (2011). The natural language of playlists. *ISMIR '11. 537–541.*

Ning, X., Desrosiers, C., and Karypis, G. (2015). A comprehensive survey of neighborhood-based recommendation methods. *Recommender systems handbook. Springer, 37–76.*

Ning, X. and Karypis, G. (2011). Slim: Sparse linear methods for top-n recommender systems. *11th IEEE International Conference on Data Mining. IEEE, 497–506.*

Quadrana, M., Cremonesi, P., and Jannach, D. (2018). Sequence-aware recommender systems. *arXiv preprint arXiv:1802.08452.*

Quadrana, M., Karatzoglou, A., Hidasi, B., and Cremonesi, P. (2017). Personalizing session-based recommendations with hierarchical recurrent neural networks. *arXiv preprint arXiv:1706.04148.*

Resnick, P. and Varian, H. R. (1997). Recommender systems. *Commun. ACM*, 40(3):56–58.

Rosenblatt, F. (1957). The perceptron—a perceiving and recognizing automaton. *Report 85-460-1. Cornell Aeronautical Laboratory.*

Rossi, R. J. (2018). *Mathematical Statistics : An Introduction to Likelihood Based Inference.*

Rubinstein, R. Y. and Kroese, D. P. (2004). *The Cross-Entropy Method.* Springer, New York, NY.

Ruder, S. (2017). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747.*

Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. (2007). Collaborative filtering recommender systems.

Sheil, H., Rana, O., and Reilly, R. (2018). Predicting purchasing intent: Automatic feature learning using recurrent neural networks. *arXiv:1807.08207v1.*

Smirnova, E. (2018). Action-conditional sequence modeling for recommendation. *arXiv preprint arXiv:1809.03291.*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Łukasz Kaiser, and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems. 5998–6008.*

Wang, J., Caverlee, J., Louca, R., Hu, D., Cellier, C., and Hong, L. (2020). Time to shop for valentine's day: Shopping occasions and sequential recommendation in e-commerce. *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20), February 3–7, 2020, Houston, TX, USA.*

Wang, S., Cao, L., and Wang, Y. (2019). A survey on session-based recommender systems. *arXiv preprint arXiv:1902.04864*.

Xu, C., Zhao, P., Liu, Y., Sheng, V. S., Xu, J., Zhuang, F., Fang, J., and Zhou, X. (2019). Graph contextualized self-attention network for session-based recommendation. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19).*