Recurrent Neural Networks for Next-Action Prediction

Rita Vieira Conde

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 tested. 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.

Index Terms—recommender systems; sequence modeling; markov chain; recurrent neural networks

1 Introduction

Recommender Systems (1) 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 (2), and this work lays in this area of study.

The goal 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 browsing 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 (3) (4). 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, the applicability of both non-deep and deep learning models to the problem was tested. As a starting point, simple statistical models were experimented, 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 created models, being one of them the over extensive catalogue of elements for certain components of an action. That said, the existing dependencies in an item-feature perspective as well as between the different item's features - for instance, the item 'skirt' can belong to the category 'women clothing', which represents a taxonomic relation between the components item and category -, were used as additional information to narrow the set of elements to consider in the catalogue during the training and evaluation process of the RNN-based models. 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.

The remaining of this paper is organized as follows: Section 2 introduces existing work in Sequence-Aware Recommender Systems. Section 3 describes the dataset used, and the proposed methods to address the sequence modelling task. Section 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, Section 5 summarizes the main conclusions from this research, and presents some ideas for future work.

2 Related Work

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 (5), where based solely on a similarity score between the last clicked item and the remaining items available, the system predicts which of the items is most likely to appear next on the current session. A popular criteria used to measure item similarity is to assume that items often interacted together, in sessions from other users, are similar to each other (6). This statistical approach has actually proved to work in many e-commerce contexts (7), however it ignores information from past clicks.

The first model using a Recurrent Neural Network for sequential recommendation was presented by Hidasi et al.(3) and Hidasi and Karatzoglou (4), as a GRU4Rec, which models the problem as a ranking problem rather than a classification problem. The RNN takes as input an item ID, encoded in a 1-of-N way, where N is the total number of items. The hidden state is also encoded in the same way, having a vector representing the entire item space of previously clicked items in the current session. The output is computed using the resulting weight output which is a vector, of similar shape as the input, and 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, in order to compute the loss using a ranking loss function. For a more efficient training, the network was tested with different ranking loss functions, such as cross-entropy, BPR and TOP1. In addition, the authors use session-parallel mini-batches. In this work, a similar model to GRU4Rec is used, using sessions with the original length at a time, as well as the categorical cross-entropy loss function, and another modified version of the function tailored to the dataset used.

To model feature-to-item recommendations, the p-RNN version was introduced (8), which can take into account the clicked items as well as their features, such as text (e.g. item's description), images (e.g. item's appearance), or price. The architecture of the model is composed by RNNs for each feature and another to model the item (a GRU4Rec). 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 item ID subnet and the hidden state of the item feature(s) subnet(s). Then, the output is computed from the concatenated hidden state, and gives out the predicted next item ID in the session. 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 reveal alternative training strategies that can be applied. In the end, it is shown that a p-RNN, using any of the alternative training strategies presented, gives more accurate next-item recommendations than the baseline model of a GRU4Rec. In this work, side information is also used to model certain components of an action, such as the item ID, but using a distinct training strategy as the ones for p-RNN.

3 Proposed Approach

An action is represented by a set of different components, subsequently, the defined problem to predict the next-action in a session, can be disintegrated and approached as individual sequence prediction tasks corresponding to the prediction of each action component. 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. This section introduces the whole process behind the different classification models designed.

3.1 Implemented Models

The different classification models to be tested, in the sequence modeling tasks, were created under three different approaches that are introduced over the following sections.

3.1.1 Baseline Approach

The starting point was given with the creation of a simple 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. The designed procedure is a simple last sequence element copy, which only takes into account the very last element in the session, and predicts the next element inline as being equal to the exact previous one.

3.1.2 N-th order Markov Chain Approach

A traditional problem in statistical language modeling is to model the probability that a given word appears after a sequence of words, which is usually accomplished through a N-th order Markov Chain (9), also commonly referred to as *n*-grams model (10). The model predicts the occurrence of an element based on the occurrence of its *n*-1 previous elements present in the sequence:

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

To estimate the *n*-gram probabilities, the Maximum Likelihood Estimation (11) approach is applied. To calculate a particular bi-gram probability of an element x_n given the previous element 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:

$$\mathrm{p}(x_n|x_{n-1}) = \frac{\mathrm{count}(x_{n-1}x_n)}{\sum_x \mathrm{count}(x_{n-1}x_n)} = \frac{\mathrm{count}(x_{n-1}x_n)}{\mathrm{count}(x_{n-1})} \tag{2}$$

Computing *n*-gram probabilities for longer sequences is a more intricate task, since these tend to occur rarely in a corpus which often leads to many probabilities of zero. To address this condition, additive smoothing (12) can be applied. It consists on adding one to all counts before normalizing them into probabilities:

$$\mathrm{p}_{\mathrm{additive\_smoothing}}(x_n|x_{n-N+1}^{n-1}) = \frac{1+\mathrm{count}(x_{n-1}x_n)}{|\mathrm{V}|+\mathrm{count}(x_{n-1})} \tag{3}$$

In Equation 3, |V| stands for the set of all elements considered. In the end, unseen *n*-grams will never have probabilities of zero.

The proposed problem, to predict the next element in the session, can also be treated as a language modeling problem using a *n*-grams model. In this work, the grams are the different elements extracted from the corpus (the website logs). The transitions between the different elements are then set out in a matrix that translates the transitions into probabilities using additive smoothing. Thereafter, to get the most likely element to come next in the sequence, the last *n* elements from the given session are checked against the matrix, and the element from the catalogue getting the highest transitioning probability corresponds to the predicted next element coming in the session.

3.1.3 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 is the Recurrent Neural Networks (13). It is a type of neural network in which the outputs from previous time steps, $\mathbf{y_{t-1}}$, are taken as inputs, $\mathbf{x_t}$, for the current time step, forming a loop:

$$\mathbf{x_t} = \mathbf{y_{t-1}} \tag{4}$$

An RNN takes a sequence of vectors $\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_t}$, where *t* represents the timestep 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 leading to:

$$\mathbf{h_t} = \mathrm{g}(\mathbf{V}\mathbf{x_t} + \mathbf{U}\mathbf{h_{t-1}} + \mathbf{b}) \tag{5}$$

In the Equation 5, $\mathbf{V}$ 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 **b** represents the bias vector. As it can be seen from the same Equation 5, the weights do not change depending on the timestep, thus the exact same values are being multiplied layer by layer every time the Back-Propagation Trough Time (BPTT) process happens. 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 formally named Vanishing Gradient (14) (15). To overcome this condition, two sub-classes of Recurrent Neural Networks emerged: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks. The architecture of an LSTM network includes all the components from a regular RNN, plus a memory cell, where the information can be stored in, and to regulate the flow into the cell, it has three different gates: the forget gate, input gate, and output gate. The GRU(16) 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 and update gate. In fact, the GRU has fewer operations, which can mean that it can be faster to train than LSTMs, nevertheless none of them clearly outperforms the other, it all depends on the data used.

In this work, the structure of the RNN model built is depicted on the left-side of Figure 1. It receives each session from the input data, in the form of sequence of elements and each element is one-hot encoded by the total number of distinct elements. 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 or an LSTM. 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 final layer that outputs the most likely element to follow in the given 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 (17). The softmax function (Equation 6), re-scales the model output. 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. Formally, the categorical cross-entropy function (18), shown in Equation 7, is designed to quantify the difference between two probability distributions, from the

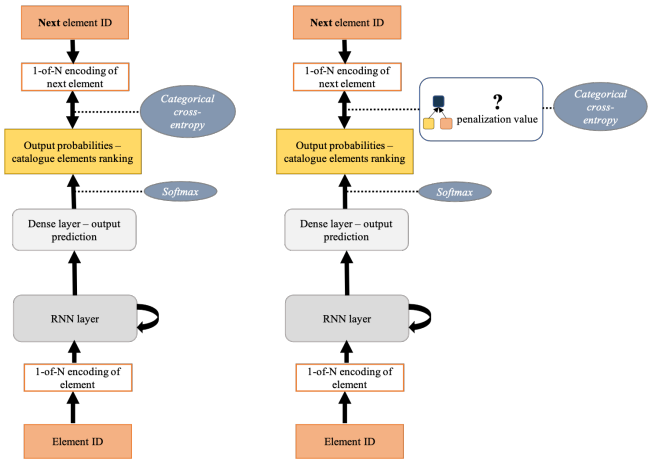


Figure 1: On the left-side, the RNN structure using a regular categorical cross-entropy loss function. On the right-side, the same RNN using a modified version of the loss function.

predicted vector and expected vector - which is, in this case, a one-hot vector that will be considered as a probability distribution.

$$f(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))} \tag{6}$$

$$CE = -\sum_x y_i \log(f(x)_i) \tag{7}$$

A separate categorical cross-entropy loss is calculated for each class label, which 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 element.

3.1.4 *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. 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.

This additional information is incorporated in a modified loss function to lead the model into considering only elements that are strictly related. The predicted and expected elements are checked against a relations matrix, where it is verified if these are siblings - elements sharing the same parent. In case the elements are related, a penalization is applied to the categorical cross-entropy result, in order to decrease the error and therefore benefit these cases. This way, the model is expected to learn to predict an element that belongs to the same family as of the expected one. Equation 8 presents the modified categorical cross-entropy loss function equation, where p represents the penalization value to be used in the positive case of a relation. The structure of the new model is depicted on the right-side of Figure 1, where the small simplified taxonomy tree represents the verification step to check if the expected element, in orange color, has the same parent element of the predicted element, in yellow color.

$$CE = \begin{cases} -\sum_x y_i \log(f(x)_i) * p & \text{positive relation} \\ -\sum_x y_i \log(f(x)_i) & \text{negative relation} \end{cases} \tag{8}$$

3.1.5 *Modified evaluation process*

Additionally, to prevent the model from even considering elements in the catalogue that, once again, do not belong to the same family as of the expected label, the final output probabilities list is cut down to only include elements correspondent to the same family group.

4 Experimental Evaluation

This section presents all the process behind the evaluation of the designed models. The first two sections analyse the e-commerce website dataset used and also present the dataset preparation required to train the models. The following section introduces the metrics that were extracted to fairly compare the performance of the different models.

Lastly, the final sections describe all the conducted experiments to address the sequence modeling tasks with the given data, and later report and discuss 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 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 corresponds to 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 of the type *view*, *add to cart* or *transaction*, and the exact timing is also recorded.

Each item has a unique ID, a property and its corresponding value, and a timestamp. The property can be the item's availability or category. 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 only the very first category recorded in the database. Hence, each action instance is formally characterized as {timestamp, visitor, event, item, category}

An item belongs to a category which can belong to an upper-level parent category. This taxonomy is described in a tree format with 5 different levels of categories. To demonstrate the structure, a simplified tree is pictured in Figure 2. 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 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 these 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. That said, the totality of the existing 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 levels, the bar graphs in Figures 3 illustrate how the associations are dispersed, where each bin corresponds to a single category. In level 1, where there is a smaller set of categories, the items understandably 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. In both cases the distribution is uneven, having too many items corresponding to the same category, whereas other categories have too few corresponding items.

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 the repeated actions are separated from each other by less than a minute, these are considered as duplicated entries and are to be left out from the dataset. The resulting dataset has a new total of 297 868 actions (a reduction of 89%), through 17 992 sessions, from 212 434 different users. As it is seen in Figure 4, the sessions are predominantly composed by 2 actions, and sessions with a length higher than 5 actions rarely occur.

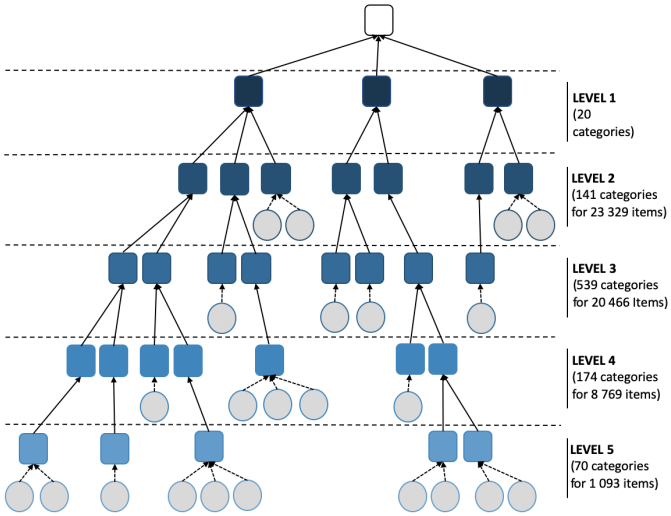


Figure 2: Dataset taxonomy with 5 levels, that displays the relations between the items (represented by a circle) and the corresponding category on each level (represented by a square)

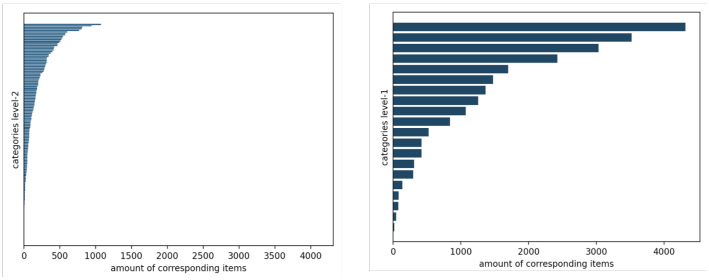


Figure 3: Distribution of the amount of items per category, regarding only level 1 and level 2 categories.

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

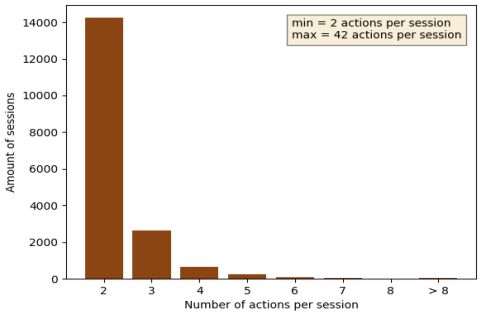


Figure 4: Distribution of the amount of actions by session.

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. From the final 17 992 sessions, 33% of these have the same item ID throughout the complete sequence of actions, and 45% contain at least two actions with the exact same item ID in the sequence. Regarding the category component, the diversity of different categories IDs over the sessions is nearly non existent, given that 81% of the sessions have the same category ID throughout the whole sequence, and when taking into account only the category levels available to all items, 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.

4.2 Dataset Preparation

To train and evaluate each sequence prediction task, the set of variable-length sessions was divided into train, validation and test sets. For hyper-parameters tuning, the train and validation sets were used, and to assess the tuned model, the test set was used. The division was done by assigning randomly different visitors to the three sets, so that sessions from the same user are not split up. The test set is the same for all the assessed models, containing 6 323 variable-length sessions.

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 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 in Table 1 demonstrate the different sessions transformations used to train both deep and non-deep learning models. On the fist table, 1a, an example training session is displayed, which is then transformed into fixed-size sequences of two elements for a bi-grams model in Table 1b. 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. Regarding the RNN model, in the last Table 1c, 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 1a, 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 1a, populated with a moving window of 1 element for an RNN model.

Table 1: Sample training sessions input transformations.

4.3 Evaluation Metrics

Each sequence modeling task in this work is taken as a classification problem. As mentioned in Section 3, 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 9.

$$\text{Accuracy} = \frac{\text{\# correctly classified test sessions}}{\text{\# test sessions}} \tag{9}$$

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.

$$\text{Recall@K} = \frac{\text{\# true next element within top-K output list}}{\text{\# test sessions}} \tag{10}$$

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 10. 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 (11), 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.

$$\text{MRR} = \frac{1}{\text{N}} \sum_{i=1}^{\text{N}} \left(\frac{1}{rank_i} \right) \tag{11}$$

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.

	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	0.840	0.874	0.957	0.974	0.911	0.320	0.470	0.524	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 2: Evaluation on the prediction of the attributes of the next-action: category of any level, category of level 2 and item

4.4 Experimental Methodology

An action is an aggregation of different components, as exemplified in Figure 5. 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 6.

This section presents the set of experiments to be executed in order to evaluate the models performances in each prediction task. The different experiments are defined through three sub-sections, and the corresponding results are later shown in the upcoming Section 4.5.

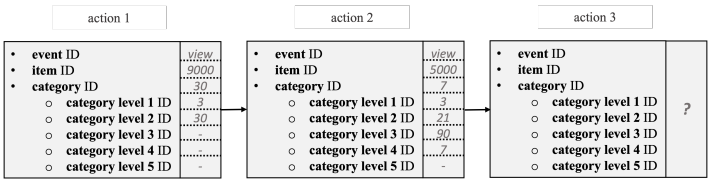


Figure 5: Sequence of actions.

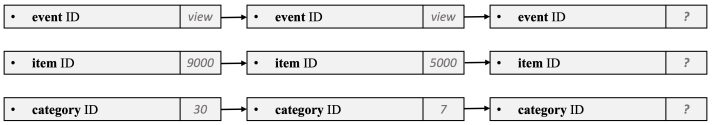


Figure 6: Sequence of decomposed actions.

4.4.1 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, with 98% of events being *view* clicks, this is a component not worth modeling. Hence, the components of the next action to be predicted are the next item and next category. To that end, the baseline approach presented in Section 3.1.1, the N-th order Markov Chain in Section 3.1.2 and the RNN approach in Section 3.1.3, 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 used for this particular tasks is set to predict a category within a vocabulary containing all 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, level 2, and for this case, the range of 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.

4.4.2 Next Element Prediction using Contextual Information

For the defined sequence prediction tasks, the contextual information coming from the existing taxonomic relations between the actions components is now to be integrated in the training and evaluation processes of the RNN-based models, as it was explained in Sections 3.1.4 and 3.1.5.

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 level 2 category that has the same level 1 category as of the true level 2 category. For the prediction of the next item ID, the taxonomic relation to be used is between the item and its corresponding level 2 category (a category level that all items IDs have correspondence).

4.4.3 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, 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.

4.5 Experimental Results

This section showcases and discusses the results achieved through the execution of the previously defined experiments over Section 4.4.

4.5.1 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 2.

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 this peculiar structure of the data, the baseline model reaches a high R@1 accuracy of 0.797.

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

Recurrent Neural Networks	next-**category level 2** prediction				next-**item** prediction			
	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 3: Evaluation of the GRU models that use side information to predict the category level 2 and item on the next-action.

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. Each model was trained over 100 epochs with 70 neurons, and with Adam optimizer (19). 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.

4.5.2 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, 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 2.

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.

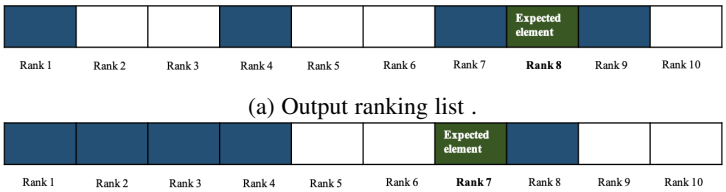
Any of the RNN types used was trained with the same hyperparameters values as previously. Naturally, any of these models performs better than before, given that the number of distinct values is much smaller, and in addition, as it was stated in Section 4.1, when focusing on a higher level in the taxonomy 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.

4.5.3 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 3.1.3. Through the modified loss function, introduced in Equation 8, 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 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.

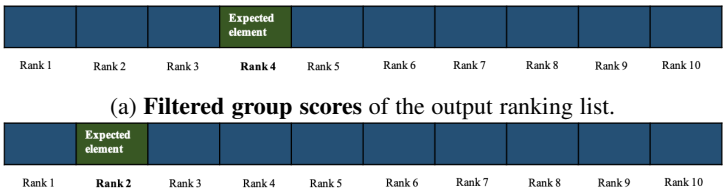
With the inclusion of this new loss function, it was possible to assess that the model is in most cases benefiting the categories



(a) Output ranking list .

(b) Output ranking list with the **custom loss function**.

Figure 7: Difference between output lists using different loss functions.



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

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

Figure 8: Difference between output lists using the filtered group scores step and different loss functions.

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 7a and Figure 7b, 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 7b, 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 7a, 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 3, 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 - 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.

The output ranking lists for the same example session would now

be as in Figure 8a, when using the regular cross-entropy loss function, and as in Figure 8b, 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 colored positions.

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 3, 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.

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

4.5.4 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 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 5 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 4.5.2, 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.

4.5.5 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 2. 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, 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 others, 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.

The RNN-based models were trained over 70 epochs, with Adam optimizer (19), and with a number of neurons that amounts to half of the number of distinct items IDs. 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 the GRU, since it presents the best results and is simpler to train (there are fewer computations in a GRU than in an LSTM).

4.5.6 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 8, with the best penalization value p of 0.01 as seen on Table 4, 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 3, the best performing GRU version uses these two additional steps.

4.5.7 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 5, the results from the statistical models do not differ greatly from the ones on the first item prediction experience on Table 2. 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 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.

5 Conclusions

The underlying aim for this work was to predict the next action a customer is most likely to perform, after a sequence of actions in a browsing session from an e-commerce website.

This work decomposes an action and predicts every component separately. Given 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 research works in the scientific domain that already benchmark their results based on deep learning methods (20). In

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

this work, statistical models were initially explored and for a more advanced deep-learning approach, Recurrent Neural Networks (RNN) were used. To fairly assess the performance of the diverse approaches taken, the same experiments were executed for each created model, and in fact, the simple statistical approach proved to perform similarly to more modern deep learning approaches in most experiments, due primarily to the nature of the dataset used.

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 hierarchically relate between each other was found as relevant and incorporated in the RNN-based model, through modified training and evaluation processes of the network. In the end, this version of an RNN-based model proved to outperform all others by an interesting 5% margin in the results for all ranking metrics used, in both next category and next item prediction tasks.

Regarding future work, it would be interesting to see the impact of using the history of actions from the current user's session along with history from previous visiting sessions to the website, in order to make personalized recommendations to each user. To that end, a similar model to an H-RNN presented by (21) could be designed. In addition, a limitation to bear in mind in e-commerce settings is the heterogeneity of items popularity, which is not being modelled in a special way. For instance, attention mechanisms (22) could be used to highlight certain elements, just as in the work presented by Wang et al. (23), where depending on the time of the year, a different emphasis is given to certain elements.

References

[1] P. Resnick and H. R. Varian, "Recommender systems," *Commun. ACM*, vol. 40, no. 3, p. 56–58, Mar. 1997. [Online]. Available: https://doi.org/10.1145/245108.245121

[2] M. Quadrana, P. Cremonesi, and D. Jannach, "Sequence-aware recommender systems," *arXiv preprint arXiv:1802.08452*, 2018.

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

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

[5] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," 2007.

[6] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," 2003.

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

[8] B. Hidasi, M. Quadrana, A. Karatzoglou, and D. Tikk, "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*, 2016.

[9] A. Markov, "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]*, vol. 2, no. 1b, pp. Article 10, 12 p., electronic only–Article 10, 12 p., electronic only, 2006. [Online]. Available: http://eudml.org/doc/128778

[10] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2000.

[11] H. Jeffreys, *Theory of Probability*. Clarendon Press, Oxford, 1948.

[12] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Article No. csla.1999.0128*, 1999.

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

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

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

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

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

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

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980v9*, 2017.

[20] S. Wang, L. Cao, and Y. Wang, "A survey on session-based recommender systems," *arXiv preprint arXiv:1902.04864*, 2019.

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

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

[23] J. Wang, J. Caverlee, R. Louca, D. Hu, C. Cellier, and L. Hong, "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*, 2020.