An Attentive RNN Model for Session-based and Context-aware Recommendations: A Solution to the RecSys Challenge 2019

Ricardo Gama*

School of Technology and Management of Lamego
Polytechnic Institute of Viseu
Lamego, Portugal
rgama@estgl.ipv.pt

Hugo Fernandes*

Rockets of Awesome
New York City, New York, USA
hugoguh@gmail.com

ABSTRACT

In the RecSys Challenge 2019 the participants were asked to predict which items, from a presented list of items/accommodations of a search result on trivago, had been clicked-on during the last part of a user's session. Here we present the 7th place solution¹. It consists of a neural network designed to learn interactions between session, context, sequence features, and the features of the displayed items at the time of a click. Our approach uses well established deep learning techniques, such as Recurrent Neural Networks, Attention and self-Attention mechanisms to deal with the different aspects of the information available, and it predicts a (categorical) probability distribution over the list of presented items. In addition to the model structure we also describe the somewhat heavy feature engineering, data augmentation and other decisions/observations made a long the way.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning approaches**; *Machine learning algorithms*.

KEYWORDS

Recommender systems, recurrent neural networks, attention, challenge, session-based, context-aware.

ACM Reference Format:

Ricardo Gama and Hugo Fernandes. 2019. An Attentive RNN Model for Session-based and Context-aware Recommendations: A Solution to the RecSys Challenge 2019. In *Proceedings of the ACM Recommender Systems Challenge 2019 Workshop (RecSys Challenge '19)*, September 20, 2019, Copenhagen, Denmark. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3359555.3359757>

1 INTRODUCTION

In recent years, the use of Deep Learning-based Recommender Systems has been growing [14]. Following the advances and successes

in other areas, some Deep Learning ideas have been incorporated and adjusted to the specificities of recommender systems. In session-based recommendation settings in particular, Recurrent Neural Networks (RNN) have been used to model the sequence of user actions and incorporate this information in the recommendation process. Recently, attention mechanisms have been also included as important algorithmic components to leverage the quality of the recommendations. In fact, many variations of attention and self-attention mechanisms, have been proposed to this account (e.g. [4, 7, 10]).

In the 2019 RecSys Challenge [8], we were asked to predict which item/accommodation, from a presented list, the *impressions*, of a search result on trivago, had been clicked-on. Our approach's main idea was to use the impressions as a central part the model, to which all other model elements would relate. In general terms, we used several techniques to process the provided session information and combine it with the items' features, having a final output layer that, following ideas presented in [13], predicts a probability vector over the items' positions in the list of impressions. Since the session data is processed simultaneously for all the items in the list, we could expect the model to not only extract the patterns in the sequences of user actions and iterations between the several features, but also to model competition between items in the displayed list.

2 FEATURES

We divide the input features into four categories: Session features that have a single value for each session; Sequence features that vary within session but not for each item and are associated to each step/timestamp; Impression features, which include both item properties that remain the same for each item, as well as others, like Click-through rate (CTR), that are associated with each item but change within session and timestamp; and current/activate Filters features.

All categorical features were encoded using an embedding layer. Numerical features, were standardized or normalized using min-max scaling.

2.1 Session features: x^{ses}

Session features are features that have a single value for each session: platform, device, day (we corrected sessions that included more than one day, by further slitting them), day of week, start hour, number steps in session. We have also included last user action and corresponding reference before clickout.

*Both authors contributed equally to this work.

¹Team Mustelideos. Final Score: 0.678838 MRR

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

RecSys Challenge '19, September 20, 2019, Copenhagen, Denmark

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-7667-9/19/09...\$15.00

<https://doi.org/10.1145/3359555.3359757>

2.2 Sequence features: $\{x_1^{seq}, x_2^{seq}, \dots, x_{n_{steps}}^{seq}\}$

Sequence features are features that change along the session, are associated with each timestamp/step but do not change for each item. These include some raw features like action type, reference (of action), country, current filters, click hour, as well as some slightly processed ones: city, is platform equal to country, last activated filter. It also includes the last reference associated to any specific kind of action or group of actions and the corresponding time since that action: previous reference, previous reference that is an item, previous action, previous action on an item, time since previous action, time since previous action on an item, etc.

As a preprocessing step for the x^{seq} features, we added a starting token and padded/truncated all sequences at 512 steps.

2.3 Impression features: $\{x_1^{imp}, x_2^{imp}, \dots, x_{n_{items}}^{imp}\}$

These are features associated with each item in a list of impressions. They include the item properties, which are fixed for each item, as well features that are specific of each session's step and item.

2.3.1 Item properties. To the list of item properties we added some derived ones, namely:

- rating: 'Satisfactory Rating', 'Good Rating', 'Very Good Rating', 'Excellent Rating'
- from stars: 'From 2 Stars', 'From 3 Stars', 'From 4 Stars';
- stars: '1 Star', '2 Star', '3 Star', '4 Star', '5 Star';
- total number of properties of each item.

These were encoded both numerically and as categorical variables using embeddings.

2.3.2 CTR. For each item in the list of impressions, we computed, at user-level and at session-level: 1.a. how many times it had been the subject of each item-actions (interaction item image, clickout item, interaction item info, interaction item deals, interaction item rating); 1.b. how many times it had been the subject of any item-action; 2. how many times it had appeared in a past impression list; 3. the CTRs: how many times it had been the subject of a specific item action or group of actions divided by how many times it had appeared in a past impression list, i.e. 1.a. or 1.b. divided by 2.

2.3.3 Dwell time. For each item in the list of impressions, we computed, at user-level and at session-level the total amount time that the user had interacted with that item. Here we did not get to separate by action type, and we do not know if that would have improved performance.

2.3.4 Past actions. Last action on each item (if any) and time since the last action on each item.

2.4 Filter features: x^{fil}

We considered the list of active filters separately. In addition to the list of active filters at the time of the last timestamp, we also considered the latest activated filter before the clickout within the session.

3 ARCHITECTURE

The overall architecture of the Neural Network can be divided into several functional blocks that process and combine the various kinds of features and representations.

Figure 1: General model architecture.

Concretely, we use a Recurrent Layer for the sequence-related features (3.1); Attention and self-Attention layers (3.2) to extract a feature vector representations whenever a list or set of features is present (e.g. active filters or item properties); a feature interaction layer (3.3), and a final dense block (3.4) that outputs a probability distribution over the items in the list of impressions.

3.1 Recurrent Layer

After concatenating all the categorical embedding vectors (e.g. action type) and continuous features (e.g. time since last action) along each time step, the sequence features list, x^{seq} , was embedded in a d_h space using a Recurrent Neural Network (RNN), resulting on $\{h_1, h_2, \dots, h_n\}$ hidden states, with $h_i \in \mathbb{R}^{d_h}$ and where n is the number of steps in the sequence. For this part of the model, we tried two commonly used RNN models: GRU and LSTM. The performance of these two options was similar but the GRU gave a slightly better performance (see Fig. 3).

We use a linear transformation of the session features/context, x^{ses} , as initialization, h_0 , of the hidden states:

$$h_0 = W_0 x^{ses},$$

where $W_0 \in \mathbb{R}^{d_h \times d_{ses}}$ and d_{ses} is the dimension of the session features. The idea was to allow the RNN to learn from x^{seq} while

taking into account the context of the session. We found that using this initialization improved performance over using a learnable vector (see Fig. 3).

When using the LSTM, in addition to the h_0 initialization, we set the initial cell state vector c_0 to be a learnable vector of our model.

3.2 Attention Layer

We were explicitly given, or computed, four sets of lists of features or representations: currently activated filters, list of items/impressions, each item's list of properties, and the list of hidden states obtained by the RNN. In order to pull out good NN representations from these sets of lists we applied the additive attention mechanism [2]. Concretely, given a list of features or representations $\{f_1, f_2, \dots, f_m\}$, with $f_i \in \mathbb{R}^{d_f}$, and a context vector s , we compute the attention vector as follows

$$u_i(f_i, s) = v_a^T \tanh(W_1 f_i + W_2 s + b), \quad \forall i \in (1, \dots, m) \quad (1)$$

where $W_1 \in \mathbb{R}^{d_i \times d_f}$, $W_2 \in \mathbb{R}^{d_i \times d_s}$, $b \in \mathbb{R}^{d_i}$ and $v_a \in \mathbb{R}^{d_i}$. Then we have,

$$\begin{aligned} \alpha_i &= \text{softmax}(u_i(f_i, s)), \quad \forall i \in (1, \dots, m) \\ c &= \sum_i \alpha_i f_i, \end{aligned} \quad (2)$$

which gives us the attention vector representation c of the feature list.

The choice of the context vector s depends on the set of vectors we want to apply attention to. For example, for the list of hidden states obtained from the RNN, $\{h_1, h_2, \dots, h_n\}$, we perform two different context attention operations: in the first operation we choose s to be the last hidden state h_n , and for the second operation we use the impression features as context, that is $s = x_j^{imp}$ for each $j \in (1, \dots, n_{it})$. This allows the network to focus on possibly different parts of the sequence features according to the various context vectors provided.

We also computed Self-Attention, allowing each vector on the list of impression features (the given properties concatenated with other engineered item features) to attend to the list itself. Concretely, considering $\{f_1, f_2, \dots\} = \{x_1^{imp}, x_2^{imp}, \dots\}$, for each $j \in (1, \dots, n_{it})$ we set the context vector to be $s = x_j^{imp}$, and thus obtaining for each j a particular weighed average, c_j , of the impressions' features.

3.3 Interaction Layer

In order to extract interactions between two distinct groups of representations computed in the intermediate layers of our network, we projected them into the same space and then added them. This is similar to what is done in the attention block/layer/step. Concretely:

$$u(f_1, f_2) = \tanh(W_1 f_1 + W_2 f_2 + b) \quad (3)$$

where $W_1 \in \mathbb{R}^{d \times d_{f_1}}$, $W_2 \in \mathbb{R}^{d \times d_{f_2}}$, $b \in \mathbb{R}^d$.

In total we applied three of these blocks in order to combine information between session, sequence and filters, and the embeddings of the items in the impressions.

3.4 Final Layers

At the end of the NN's main body we obtain one processed representation vector for each item. This vector is then feed into a dense layer block consisting of linear layers, layer normalization [1], and a non-linearity RELU activation. Finally, the resulting vectors are flattened into a unique representation vector that is feed to a final fully connected layer to generate a probability distribution over the items' positions in the list of impressions. Concretely, the probability vector is given by:

$$p = \text{softmax}(C \tanh(W_f x)) \quad (4)$$

where $x \in \mathbb{R}^{800}$ is the representation vector, $W_f \in \mathbb{R}^{25 \times 800}$ the weight matrix and C a hyper-parameter that controls the range of the logits [3]. We tried to set C as a parameter, but at the end chose to hard set it to $C = 10$ based on the validation score.

4 SETUP AND RESULTS

4.1 Validation

In order to build a validation set we started by determining how the missing values in the test set were chosen. We found that they corresponded to the last step of each user's largest session(s) on days 7 and 8. We replicated this for days 5 and 6 and used those sessions as our validation set. The Leaderboard's public score was reasonably well predicted and typically around 1.7% higher than the score that we observed on the validation set (see Fig. 3). During this process we also noticed that a few sessions were not well separated (they included data in different days) and corrected that.

4.2 Data Augmentation

We augmented the dataset by including all sub-sessions that start at step 1 and end in a clickout item. That is, a session with n clickout items would generate $n - 1$ new sub-sessions. Note that it is not a balanced data augmentation as it is biased towards smaller session sizes. This represented 92% increase in number of sessions, and lead to an almost 1% score improvement (see Fig. 3).

4.3 Training

We used the Adam optimizer and a learning rate schedule of 10^{-3} for the first two epochs, 7.5×10^{-4} for the third and 5×10^{-4} on the remaining epochs. We ran the model for 6 epochs and chose the epoch with best validation loss to decide which epoch to use for the submission (see Fig. 2 and Table 1). In order to prevent overfitting we used dropout on the final full connected layers and L_2 regularization on all network weights with $\lambda = 10^{-5}$. We initialized all model parameters using the Xavier normal initialization [5].

All hyper-parameters (e.g. dimensions of embeddings and RNN, learning rate and regularization values) were optimized via cross validation, using our best model as reference.

The challenge's metric was the Mean Reciprocal Rank (MRR). We tried to optimize directly for rank metrics such as BPR and TOP1 [6], either from the start of training or doing fine-tuning, but without success. We found MRR to be well correlated with the negative log-likelihood (NLL), and since it gave us the best results, we used it as our loss function.

Figure 2: Model train/validation sets losses values evolution during NN training. The shaded area represents 95% CI (bootstrap) of the mean value over 10 different runs.

Model	MRR (95% CI)
model	0.6621 (0.6619, 0.6623)
+future	0.6658 (0.6654, 0.6661)
without h_0 context initialization	0.6618 (0.6612, 0.6624)
LSTM instead of GRU	0.6618 (0.6616, 0.6621)
without data augmentation	0.6567 (0.6563, 0.6570)
without time and CTR	0.6474 (0.6473, 0.6476)

Table 1: Single Model Performance on the validation set: average MRR with 95% CI (bootstrap) over 10 different runs. Note that these show variability due to change in seed only.

4.4 Model (Self-)averaging

In order to reduce the variance of our neural network we tried bootstrap aggregation, but what worked best was to average different runs of the same model with different initialization seeds (see Figure 3 for improvement as a function of number of runs). Even though it is a single neural network model we found that it benefited a lot from averaging several runs of that same model.

4.5 Impact of Feature Engineering

Since a competition can be won by the 3rd or 4th decimal case and computing time is not an issue, feature engineering cannot hurt. Unlike other areas of successful applications of NN we found that, at least for our model, somewhat heavy feature engineering was very much needed (see model "without time and CTR" in Figure 3). Some of these features we could hope that a good NN structure would learn from raw or minimally processed data alone, like for instance, past within-session click-through rate (CTR) for each of the items in the list of impressions.

4.6 Questionable Data Augmentation

The rules of the Challenge did not restrict the use of data that would not be available in production, such as, time since last action: an algorithm would generate the list of impressions without knowing how long the user is going to take to act. And even if it knew, it

Figure 3: Mean Reciprocal Rank as function of the number of averaged models. LB indicates predicted public score on the Leaderboard.

would be limited to generate recommendations within that time. Definitely unavailable but also not explicitly restricted by the rules is future data. We tested including all sessions without missing values of the test set sessions as training data. This gave us an improvement of about 0.5% on the validation which, we predict, moved us to 7th place from a would have been 9th or 10th place.

5 FINAL REMARKS

In the last few years several variations of the attention mechanism have been proposed. In our model we use additive attention. We are curious in particular about whether the use of a multi-head attention mechanism [12] could help in improving the NN's performance.

One of the main challenges we faced was overfitting. Even with dropout and weight regularization to minimize this effect and enhance the network generalization capacity, in general the problem remained (Fig. 2). Whether due to our data augmentation strategy or the overall network training methodology, this issue was not overcome during the competition. We wonder if some recently introduced techniques (e.g. [9, 11]) could help improve the model's predictive power.

We observed (data not shown) that prediction was harder when the action preceding the clickout was not an item action. We did very little work in trying to incorporate the searches for a point of interest (POI) text data. We wonder how much improvement its inclusion could bring.

Finally, one aspect that is important for model performance and we did not get to explore is the initialization of the embeddings, in

particular for cold start, i.e. for items and users that did not appear on the training set. This is something that we think would have been worth exploring with some simple heuristic or an unsupervised technique.

The source code for the proposed model, as well as detailed instructions, can be found in GitHub².

ACKNOWLEDGMENTS

The first author is grateful to Polytechnic Institute of Viseu and CI&DETS for their support.

REFERENCES

- [1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. (jul 2016). arXiv:1607.06450
- [2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. (2014). arXiv:1409.0473
- [3] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2016. Neural Combinatorial Optimization with Reinforcement Learning. (2016), 1–15. <https://doi.org/10.1146/annurev.cellbio.15.1.81> arXiv:1611.09940
- [4] Tong Chen, Hongzhi Yin, Hongxu Chen, Rui Yan, Quoc Viet Hung Nguyen, and Xue Li. 2019. AIR: Attentional intention-aware recommender systems. In *Proceedings - International Conference on Data Engineering*, Vol. 2019-April. IEEE, 304–315. <https://doi.org/10.1109/ICDE.2019.00035>
- [5] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. *PMLR* 9 (2010), 249–256. <https://doi.org/10.1146/annurev.cellbio.15.1.81> arXiv:arXiv:1011.1669v3
- [6] Balázs Hidasi and Alexandros Karatzoglou. 2017. Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. (2017). <https://doi.org/10.1145/3269206.3271761> arXiv:1706.03847
- [7] Wang Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, Vol. 2018-Novem. 197–206. <https://doi.org/10.1109/ICDM.2018.00035> arXiv:1808.09781v1
- [8] Peter Knees, Yashar Deldjoo, Farshad Bakhshandegan Moghaddam, Jens Adamczak, Gerard-Paul Leyson, and Philipp Monreal. 2019. RecSys Challenge 2019: Session-based Hotel Recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems*. ACM.
- [9] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. *Regularizing and Optimizing LSTM Language Models*. Technical Report. arXiv:1708.02182v1 <https://arxiv.org/pdf/1708.02182.pdf>
- [10] Pengjie Ren, Zhumin Chen, Jing Li, Zhaochun Ren, Jun Ma, and Maarten De Rijke. [n. d.]. RepeatNet: A Repeat Aware Neural Recommendation Machine for Session-based Recommendation. www.aaai.org
- [11] Leslie N. Smith. 2018. A disciplined approach to neural network hyperparameters: Part 1 – learning rate, batch size, momentum, and weight decay. (mar 2018). arXiv:1803.09820 <http://arxiv.org/abs/1803.09820>
- [12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *Nips* (2017). <https://doi.org/10.1017/S0140525X16001837> arXiv:1706.03762
- [13] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 2692–2700.
- [14] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Comput. Surv.* 52, 1, Article 5 (Feb. 2019), 38 pages. <https://doi.org/10.1145/3285029>

²<https://github.com/mustelideos/recsys-challenge-2019>