Recommendation Systems -
Final Project

Shir Cohen & Dana Gorovici

Based on ‘Session-Based Recommendation with Recurrent Neural Networks’ by Balazs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, Domonkos Tikk

1. Anchor paper

Our anchor paper is ‘Session-Based Recommendation with Recurrent Neural Networks’, published as a conference paper at ICLR 2016.
The main objective of the paper is to build a recommendation model with limited user data. Although the article was published in 2016, It explores a recent issue developing in the field of recommendation theory is the use of limited availability of user data, such as user demographic, cookies, and even previous browsing history. Often, we are tasked with building a recommendation model for items in such situations. Previous recommendation models either do not take into account temporal association of items in browsing history (such as KNN) or are either statistically complex (such as Markov chains). This article introduces a new type of recommendation model - a RNN deep learning based model. The authors present a GRU RNN on session-based item data.

2. Improvement

The original RNN only requires item id as an input. Our proposal, in order to improve the performance of the model, is to add an additional feature that represents the users browsing time spent on each item.

Our hypothesis is that this additional data point will help the model learn and better predict the next item in a session. For example, if a user was browsing a LEGO Star Wars complete set for 50 seconds and then moved on to browse a different LEGO Star Wars ship for 70 seconds and then moved on to browse a Barbie doll for 3 seconds, it could be deduced that the Barbie doll selection was a deviation from the users preference. We would expect that the model should predict a similar LEGO set for the

user, and ignore the Barbie category, although it was closest in the browsing time. Alternatively, if the user would have spent 60 seconds browsing the Barbie doll, it could be said that the user is interested in purchasing either LEGO or Barbie as a present for his daughter and either item categories should be taken into account by the model as the next recommended item.

We want to use the memory aspect of the model to “remember” relevant previous information from the input stream. We hypothesize that adding time spent will enable the model to better remember the important information in the stream and forget the less important memory. The more significant information being the items that were browsed for longer periods of time.

To some extent we can even consider an item in a sequence that was viewed for less than 1 second to be a negative sample. We can deduce that since the user saw the item and clicked out to another item very fast, the item is not at all relevant for the user. Specifically, it should be part of the “not recommended” items. As a future improvement we would add items viewed less than 1 sec as negative samples in the loss calculation. Currently, the negative samples that are used are just other items in the population that do not appear in the given session.

In order to create this additional feature, time_spent, a calculation was performed on the time column between succeeding items. In any given session of length n, we define the feature time_spent for item i as the time of item i - time of item i+1.

In order to integrate this feature into the RNN input the feature needs to be combined with the item id encoding. After the one hot encoding of the item id, which is used in the original model, we concern an additional dimension to the tensor representing the time_spent. If for example there are only 10 items in the training set, it would require a tensor of size 10 to represent each item differently. Then to represent item 1 that was browsed for 23 seconds, the final tensor would be: 1,0,0,0,0,0,0,0,0,0,23

This idea was taken from the hybrid approaches we saw in lecture where for example an item vector would be a concatenation of item id one hot encoding with category one hot encoding. We thought this would be a successful way to represent the additional data. Since the majority of items had time_spent between 1 sec and 60 sec, we decided to use the integer value as is. But a further improvement would be to put the time spent in buckets of 10 based on predefined time ranges and then only add a digit between 0 and 9 to the one hot encoding.

3. **Baseline Comparison**

We are using the KNN model as the baseline comparison. This was also the baseline model that had the best performance compared to all other baseline models in the anchor paper.

The k-nearest neighbors algorithm is a non-parametric classification and regression. In our case, we will use KNN as a baseline model to check item similarity to the actual last item on each session in the training dataset and similarity is defined as cosine similarity between the vectors of their sessions. The number of co-occurrences is two items in sessions divided by the square root of the product of the numbers of sessions in which the individual items occurred. Regularization is also included to avoid coincidental high similarities of rarely visited items. This baseline is one of the most common item-to-item solutions in practical systems, and provides recommendations in the “others who viewed this item also viewed these ones” setting. Despite its simplicity it is usually a strong baseline.

4. **Evaluation**

a. Dataset - We used the yoochoose dataset. This dataset was taken from the RecSys Challenge 2015 (RSC15) and contains click-streams of an ecommerce site that sometimes end in purchase events. We work with the training set of the challenge and keep only the click events. We filter out sessions of length 1. The original model and the baseline were trained on 6 months of data, containing 7,966,257 sessions of 31,637,239 clicks on 37,483 items and 15,324 sessions of 71,222 events for the test set as implemented in the anchor paper. We use the sessions of the subsequent day for testing. Each session is assigned to either the training or the test set. We filter out clicks from the test set where the item clicked is not in the train set. Sessions of length one are also removed from the test set.

   Due to memory restrictions, we also created a sample smaller dataset for training and validation of the improvements compared to the original model. The preprocessing includes adding the new feature of time_spent, we are left with 2,117,132 sessions of 8,530,594 clicks on 30,324 items and 27,694 sessions of 100,414 clicks on 9,341 items.

b. Hyper parameter optimization - we trained the models with different combinations of parameters and found that for the GRU the best params in the paper are the

best performed also in our implementation. In addition, we found different parameters for the improved model, below are the best performing parameters:

Model	Loss	Batch Size	Dropout	Learning Rate	Momentum
GRU	TOP 1	50	0.5	0.01	0
	BPR	50	0.2	0.05	0.2
	Cross entropy	500	0	0.01	0
Improved GRU	TOP 1	20/100	0	0.01	0
	BPR	100	0	0.05	0.1

c. Evaluation Metrics - first, we trained the original GRU RNN model and compared it to the KNN model on the full dataset as mentioned in the anchor paper. We received similar results of the metrics Recall@20 and MRR@20 for different types of single layers of GRU as used in the paper.

Results for full dataset and original model:

Model	Recall@20	MRR@20
KNN Baseline Model	0.506566	0.20477
GRU RNN Model	0.595563	0.276150

Secondly, in order to evaluate our improved version of the model, we trained the three models: KNN baseline model, original model and improved model on the same sample of the dataset and calculated the same metrics of Recall@20 and MRR@20 for the best performing models.

Results for sample dataset and improved model:

Model	Recall@20	MRR@20
KNN Baseline Model	0.507659	0.231720
GRU RNN Model	0.614256	0.290315
Improved Model	0.601363	0.280375

The best model that we could find with the limited amount of time for long runs and high memory restrictions is by using the loss function TOP1-max and 1000 hidden layers, trained on 1 epoch. However, we still could not outperform the results of the original model from the paper. All the results appear in the attached files of the project 'original_metrics_for_sample', 'improved_metrics_for_sample'. We assume that with more time and resources to run heavy models, we could find a better model by hyper parameters optimizations.

5. Conclusions

Overall, the best results were achieved from the original RNN model. The RNN model performed best both in recall and MRR metrics. Our goal was to find an improvement of this model. However, given the runtime and memory constraints, we were not able to outperform the original model. Surprisingly, the original RNN model - trained on only two million sessions - achieved similar results to those of the model trained in the original article (using eight million sessions). This shows that in order to build this RNN model, smaller datasets can be as effective as larger datasets. When training on the original RNN, memory constraints occurred. The one hundred hidden unit models were only trained on one epoch due to memory constraints. This model may be trained for more epochs if memory constraints were resolved. More optimal results would be expected.

Our improved RNN was improved by adding an additional feature to the input.

Pseudo code:

```
For each session {i}n, 1<=i<=n:
  Arrange items in order of time ascending
  For item i, 1<=i<n:
    Item i time_spent := int(time item I+1 - time item i)
```

Thus, all items in a session will receive the additional feature besides the nth (last) item.

Although the article states that one hot encoding performed more optimally than embedding, this may not be the case when adding an additional feature. However, using embedding of the item id with the additional feature may perform better than the tested one hot encoding with an additional feature. This is a hypothesis for future testing.