# Reproducing Experimental Results from the Paper *Collaborative Memory Network for Recommendation Systems*

Sabrina Kuhrn
e0926421@student.tuwien.ac.at
Technical University of Vienna
*Institute for Business Informatics*
Vienna, Austria

Judith Louis-Alexandre
e12024728@student.tuwien.ac.at
Technical University of Vienna
*Institute for Business Informatics*
Vienna, Austria

Balázs Tószegi
e12034936@student.tuwien.ac.at
Technical University of Vienna
*Institute for Business Informatics*
Vienna, Austria

## ABSTRACT

The goal of this paper is to try – based on careful experimental design – to confirm the numbers and findings reported in the *Collaborative Memory Network for Recommendation Systems* paper from the 2018 SIGIR conference [2] or, alternatively, uncover inconsistencies and therefore challenge the conclusions drawn. This is done by, first, identifying the experimental setup and strategy taken in the paper and afterwards, outlining the steps necessary to reproduce the results. Overall, our work should focus on potential problems in today's published papers in terms of insufficient information given to reproduce the results or/and statistically significant differences between the different settings.

We reproduced the results for the *citeulike-a* dataset with the three CMN models and challenged the outcome with the KNN baseline model. The 1-hop CMN model had different scores in our experiments. Based on the statistical significance testing we can conclude that we could reproduce similar results with the 2-hop and 3-hop CMN models. For the KNN model, we got better results than in the original paper but not significantly. But when adding run time as metric, the p-value indicates that KNN outperforms the CMN model and therefore, we are challenging the claim that CMN on *citeulike-a* outperforms the competitive baselines.

## KEYWORDS

Experimental Design; Reproducibility; Collaborative Memory Network; Recommender Systems

## 1 INTRODUCTION

Collaborative Filtering (CF) can generally be grouped into three categories: memory or neighborhood-based approaches, latent factor models and hybrid models. The Collaborative Memory Network (CMN) method was presented at SIGIR '18 and proposes a unified hybrid model combining a memory/neighborhood-based and latent factor approach and therefore, capitalizing on the strengths of the global structure of latent factor model and local neighborhood-based structure. It, therefore, maintains three memory states: an internal user-specific memory, an item-specific memory, and a collective neighborhood state. The memory component allows read and write operations to encode complex user and item relations in the internal memory. An associative addressing scheme acts as a nearest neighborhood model finding semantically similar users based on an adaptive user-item state. The neural attention mechanism places higher weights on specific subsets of users who share similar preferences forming a collective neighborhood summary. Finally, a nonlinear interaction between the local neighborhood summary and the global latent factors derives the ranking score [2] [1].

*Experimental Design '20, January 2020, Vienna, AUT*

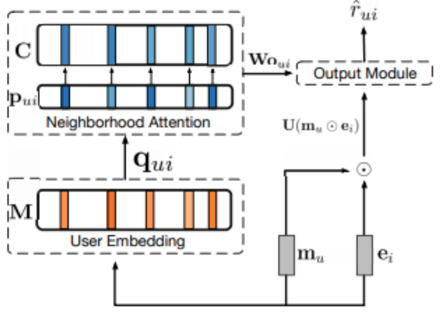

**Figure 1: Model overview**

## 2 EXPERIMENTAL SETUP

The model and what has been done is explained in great detail in the paper. The corresponding code as well and two of the three datasets used were provided in the project GitHub repository. Therefore, the information given in the paper was sufficient to reproduce most of the results reported.

### 2.1 Datasets

Originally the paper studied the effectiveness of their approach on three publicly available datasets - the *citeulike-a*, the *Pinterest* and the *Epionons* dataset.
The later is not provided in their GitHub repository, but sharing the dataset link of the dataset source in their paper [2], made it possible to find the dataset and download it as a text file. The other two datasets are provided in npz format which is a simple file format for *NumPy* arrays.
With the explanations in the paper and the data format description on GitHub, we managed to create a npz format file out of the *Epinions* text file as well. During this creation, some items are chosen randomly, therefore our npz file is for sure not the same as the one they used to get their experimental results. Also important to mention is that, due to the huge amount of data and the computing power of our private machines, creating the npz file took a lot of time. Nevertheless, in the end the *Epinions* as well as *Pinterest*

dataset were to large to perform the evaluations on our private machines using CPU.
Therefore, we just used the remaining *citeulike-a* dataset for the evaluation, which is originally collected from *CiteULike* - an online service which provides users with a digital catalog to save and share academic papers. User preferences are encoded as 1 if the user has saved the paper (item) in their library [2].

## 2.2 Evaluation Methodology

For evaluation, the same metrics as in the original papers were used, namely the Hit Ratio (HR) and the Normalized Discounted Cumulative Gain (NDCG). HR can be defined as the measurement of the existence of the positive item regarding the N best results. The NDCG measurement is based on the position of the items in the graded list and adds penalties to the score if the item is under evaluated [2]. We used $N \in \{5, 10\}$ and compared our outcome to the original paper outcomes.
Regarding parameter tuning, we used the optimal hyper-parameters that were reported in the original paper. This is appropriate because we used the same dataset, implementation of algorithm as well as evaluation procedure.

## 2.3 Code

The link to the GitHub repository [1] of the authors is given in the paper. There the data files for two out of the three datasets, the code, some information about how to run the code and some package information are available.

The code is fully written in Python and composes of two main files (pretrain.py and train.py). These two scripts import functions and classes defined in other files stored in the folder *util*. Furthermore, the main files take hyper-parameters as arguments in order to, on the one hand to pretrain the model for initialization and on the other hand afterwards run the model to create the different evaluation metrics.

The hyper-parameters used to pretrain the model are:

- the GPU device number
- the maximum number of iterations
- the batch size
- the embedding size
- the dataset npz file to use
- the number of negatives samples
- the l2 regularization
- the file name for the pretrained data output

To train the model the function takes the same arguments as for pretraining and, in addition:

- the number of hops/memory layers (iterations over user embedding and neighborhood attention)[2]
- the name of log directory, where all the log files will be saved
- the name of the file containing the pretrained data

Regarding the package requirements it is also important to mention, that the *Python* and *TensorFlow* versions are given, the *dm-sonnet* package version is missing even though the package is listed. The other packages needed to run the code are not mentioned neither in the paper, nor in the GitHub repository.

During our research, we found a paper from the 2019 ACM conference on Recommender Systems [1] that reproduced on result table from the original CMN paper - namely the comparison to other baseline models. The GitHub repository [3] for this paper was also available and gives more information about the packages needed to run the code. To be able to run the code from the CMN paper, someone can therefore use the package versions provided in the *requirements.txt* file from their GitHub repository.

Finally, regarding reproducibility it is also worth mentioning, that there is no clue about the system they used to perform their code, neither in their paper nor in their GitHub repository.

## 2.4 Our reproducing strategy

In the paper, five experiments with the introduced CMN algorithm have been made.

The first one consists of the comparison of the CMN algorithm with 1, 2 and 3 hops with different baselines models based on the before described HR and NDCG metrics [4]. The code for these baseline algorithms is not provided, neither information about the hyper-parameters used for each model is given in the paper. Thus, we were just able to compare the reported CMN results with the results we got when running their code.
As no information about the operating system as well as the exact version for some packages were given someone can guess that reproducing the exact same results is therefore impossible.
We overcame the absent information about the used baseline models and used the code provided by the RecSys '19 paper [1]. Again, due to computational power we evaluated the model comparison results for just the KNN model, which is a neighborhood-based approach computing the cosine item-item similarity to provide recommendations.

The second experiment consists of the variation of the embedding size and number of hops.

The third experiment shows the effects of attention and linearity on CMN. The effects of these parameters are explained in the paper, but not the way how they are implemented in their code. Attention and linearity are not part of the hyper-parameters list that we can easily tune. Therefore, we had a deep look into their code to try to figure out where and how attention and linearity are used and how we can change it, but we didn't manage to find a way to do so.

The fourth experiment consists of the variation of the number of negative samples used for CMN with 1, 2 and 3 hops.

[1]https://github.com/tebesu/CollaborativeMemoryNetwork
[2]Each hop queries the internal user memory and item memory followed by the attention mechanism to derive the next collective neighborhood state vector[2].

[3]https://github.com/MaurizioFD/RecSys2019_DeepLearning_Evaluation
[4]This is the experiment which was reproduced by [1]

The last experiment about attention weights for users in the neighborhood of a specific user *u*. But as the ID of the randomly selected user was not provided, we were not able to reproduce this experiment.

To reproduced these experiments, we executed the code by changing the mentioned parameters and keeping all the other parameters to their default value. We used our private machines, having different operating systems. We can see the effect of the operating system used in some of the results.

## 3 REPRODUCED RESULTS

### 3.1 Variation of number of hops

**Table 1: Experimental results compared to the original results for the CMN method using the metrics and cutoffs reported in the original paper.**

| citeulike-a | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **Our Results** | | | | **Original Results** | | | |
| | *HR@5* | *HR@10* | *NDCG@5* | *NDCG@10* | *HR@5* | *HR@10* | *NDCG@5* | *NDCG@10* |
| CMN-1 | 0.4806 | 0.6293 | 0.4031 | 0.4401 | 0.6692 | 0.7809 | 0.5213 | 0.5575 |
| CMN-2 | **0.8033** | 0.8888 | **0.6407** | **0.6686** | **0.7959** | **0.8921** | 0.6185 | 0.6500 |
| CMN-3 | 0.7970 | **0.8944** | 0.6279 | 0.6599 | 0.7932 | 0.8901 | **0.6234** | **0.6551** |

The results for 1 hop are quite far from the paper results, although, for 2 and 3 hops, the almost identical to paper results.
The results for 1 hop were computed on a different machine than the results for 2 and 3 hops. Thus, we first thought that the difference of the operating systems could explain the different results that we got. Therefore, we ran CMN 1 hop again but on a different operating systems and still ended up with worse results than in the paper. Thus, we are not able to explain the reason of such a difference in the reproduced results for 1 hop.

### 3.2 Comparison to ItemKNN

As already mentioned, they are not explaining how they are using, tuning etc. the baseline models. Therefore, we took the code provided by the RecSys '19 paper [1] and ran the code for just KNN, which computes the cosine item-item similarity to provide recommendations.
We had to change in the default hyper-parameters used in their python file (run_SIGIR_18_CMN.py) to the default hyper-parameter used in the CMN paper to be able to make a correct comparison. Therefore, we adopted in the section *DL ALGORITHM* the part *CMN_article_hyperparameters*, changing the hop parameter to 2 instead of 3 and both epoch parameters to 30 instead of 100. We are not sure if the authors of the RecSys '19 paper [1] made a mistake here, as the name *CMN_article_hyperparameters* would suggest that they are using the default parameters of the CMN paper.

Some other adaptions have been made to run the code faster, for example dropping all other models except the KNN as well as evaluating the metrics just on the citeulike-a dataset.

**Table 2: Item KNN results compared to the original results for the CMN method using the metrics and cutoffs reported in the original paper.**

| citeulike-a | | | | |
|---|---|---|---|---|
| | HR@ 5 | NDCG@ 5 | HR@ 10 | NDCG@ 10 |
| ItemKNN CF cosine | **0.8209** | **0.7012** | 0.8925 | **0.7246** |
| CMN | 0.8089 | 0.6655 | **0.8932** | 0.6930 |

When comparing the given results from Table 2 with the ones from the original paper, someone can see that ItemKNN outperforms the CMN. Just for HR@10 the CMN performs better.

As for statistical significance testing sometimes it is also good to add some satisficing metric, we thought it would be good to evaluate the train and recommendation time results as well.

**Table 3: Item KNN and CMN train as well as recommendation time results using the metrics and cutoffs reported in the original paper.**

| | Train Time | Recommendation Time |
|---|---|---|
| ItemKNN CF cosine | 28.59 ± 6.81 [sec] | 16.40 ± 0.54 [sec] |
| CMN | 9878.38 [sec] / 2.74 [hour] | 30.89 [sec] |

As someone can see, the runtime is significantly lower for the KNN, especially for training the model.

### 3.3 Variation of embedding size

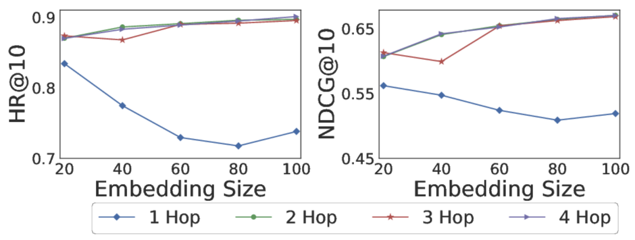

**Figure 2: Original results for CMN varying the embedding size from 20-100 and hops from 1-4.**

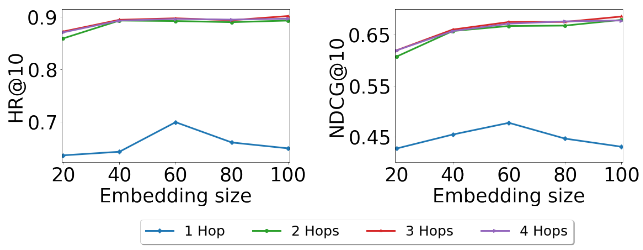

**Figure 3: Reproduced results for CMN varying the embedding size from 20-100 and hops from 1-4.**

For the variation of the embedding size, the results differs quite a lot between the original results and our results regarding 1 hop. Whereas their evaluation shows the best performance results for an embedding size of 20 followed by a decrease in performance when the embedding size increases, our results are almost equal for all the embedding sizes.
For the other numbers of hops, the results are very similar, except for 3 hops and embedding size 40. In their paper, they have a drop of the performance for the combination of these parameters. That could be, according to them, linked to poor minima. We do not have such a drop in our performance.

## 3.4 Variation of number of negative samples

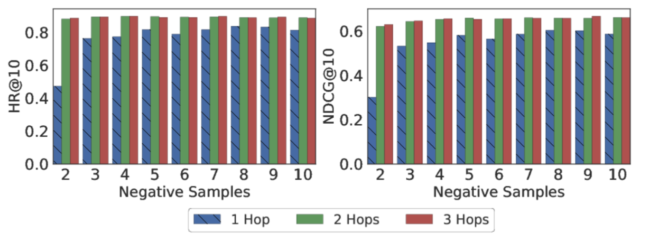

**Figure 4: Original results for CMN varying the number of negative samples from 2-10 and hops from 1-3.**

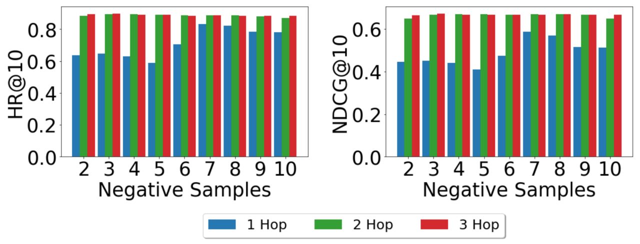

**Figure 5: Reproduced results for CMN varying the number of negative samples from 2-10 and hops from 1-3.**

Our results are very similar to the paper results regarding CMN with 2 and 3 hops. The results differ again a lot for 1 hop. We got better results for 1 hop CMN using 2 negative samples and worse for all other values.
Thus, although in the paper the performances increase for 3 negatives samples and are almost identical for all the others negative sample numbers, in our results there is a stagnation of the performance for 2, 3, 4 and 5 negatives samples, followed by an increase.

## 3.5 Statistical significance testing

For the first experiment (comparisons between CMN and baselines algorithms), a paired t-test has been performed to support their conclusion, that CMN algorithm with 2 and 3 hops outperform the baseline algorithms, but the precise results of these significance tests are not provided.

Table 4 shows the results of the significance tests. Our null hypothesis was that the reproduced results are similar to the original results and since the p values are high we can conclude that the null hypothesis is true. In the first row, one can see that the p-value is much smaller than for the others and this is due to the fact that our CMN-1 results are a bit different.

Testing the CMN against the KNN is not statistically significant. Especially, when adding the recommendation time as satisficing metric, we get a very low p-value, indicating that the KNN outperforms the CMN model.

For the negative samples, there were not provided exact original results so we could not do significance tests.

| *Compared results* | | *p-value* |
|---|---|---|
| original CMN - 1 | reproduced CMN - 1 | 0.11 |
| original CMN - 2 | reproduced CMN - 2 | 0.97 |
| original CMN - 3 | reproduced CMN - 3 | 0.96 |
| our KNN | our CMN | 0.79 |
| our KNN (with time-correction) | our CMN (with time-correction) | 6.5e-11 |

**Table 4: Results of paired t-tests**

## 4 CONCLUSION

The paper and the code are described in detail. This allows to easily have an overview on what is the purpose of the paper, what and how it is done. The major flaw that we found is that almost no information is given about the environmental specifications needed to run the code. Information about used Python packages is provided in the GitHub repository, but still incomplete: a lot of packages or the version necessary for running the exact same code are not mentioned.
Furthermore, there is no indication about the system used. During our work, we discovered that the operating system can affect the evaluation metrics. Some advice about the computational power needed would have been useful too. On our personal machines, the runtime execution was high (sometimes more than one day) and prevented us from trying to reproduce the results for the other datasets.
We were not able to reproduce all their experiments as some information were missing (e.g. the code and parameters for the baselines models, the user selected for their last experiment). For the experiments that we managed to execute, we found a difference depending on the number of hops used. Using 1 hop, we mostly got worse results than in the paper, whereas for 2 and 3 hops the results were almost identical (as the significance tests show). When comparing the performance against the baseline model, as we got better results for the KNN algorithm than for CMN, but we were not able to find a significant improvement of CMN over KNN using statistical significance testing. But when adding the recommendation time as satisficing metric we achieved a p-value lower than 0.01. Therefore, we are challenging the claim that CMN on *citeulike-a* outperforms the competitive baselines.

## REFERENCES

[1] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches *(RecSys '19)*. Association for Computing Machinery, New York, NY, USA, 101–109.
[2] Travis Ebesu, Bin Shen, and Yi Fang. 2018. Collaborative Memory Network for Recommendation Systems *(SIGIR '18)*. ACM, New York, NY, USA, 515–524.