UNDER REVIEW

Replication / Natural Language Processing

ReDynamic Routing Transformer Network for Multimodal Sarcasm Detection

Andrea Infantino[1] and Eleonora Quaranta[1]
[1]University of Illinois at Chicago, Chicago, IL

A replication of tian-etal-2023-dynamic.

Reproducibility Summary

Template and style guide for Option A, based on materials from the ML Reproducibility Challenge 2022. The following section of Reproducibility Summary is **mandatory**. *This summary* ***must fit*** *in the first page.*

Scope of Reproducibility — State the main claim(s) of the original paper you are trying to reproduce (typically the main claim(s) of the paper). This is meant to place the work in context, and to tell a reader the objective of the reproduction.

Methodology — Briefly describe what you did and which resources you used. For example, did you use author's code? Did you re-implement parts of the pipeline? You can use this space to list the hardware and total budget (e.g. GPU hours) for the experiments.

Results — Start with your overall conclusion — where did your results reproduce the original paper, and where did your results differ? Be specific and use precise language, e.g. "we reproduced the accuracy to within 1% of reported value, which supports the paper's conclusion that it outperforms the baselines". Getting exactly the same number is in most cases infeasible, so you'll need to use your judgement to decide if your results support the original claim of the paper.

What was easy — Describe which parts of your reproduction study were easy. For example, was it easy to run the author's code, or easy to re-implement their method based on the description in the paper? The goal of this section is to summarize to a reader which parts of the original paper they could easily apply to their problem.

What was difficult — Describe which parts of your reproduction study were difficult or took much more time than you expected. Perhaps the data was not available and you couldn't verify some experiments, or the author's code was broken and had to be debugged first. Or, perhaps some experiments just take too much time/resources to run and you couldn't verify them. The purpose of this section is to indicate to the reader which parts of the original paper are either difficult to re-use, or require a significant amount of work and resources to verify.

Communication with original authors — Briefly describe how much contact you had with the original authors (if any).

1 Introduction

In this work we attempt the reproduction of the results presented in the paper ”Dynamic Routing Transformer Network for Multimodal Sarcasm Detection” [1], which proposes DynRT, a new architecture to tackle the sarcasm detection task. The authors of the paper leverage multimodal NLP to join information obtained from pictures and their corresponding captions.

2 Scope of reproducibility

The authors of the paper claim that their model outperforms the state-of-the-art models in the sarcasm detection task. They also claim that their model is able to leverage multimodal information to improve the overall performance in terms of both accuracy ad F1 score. The novel DynRT architecture is used to capture incongruity between the picture and the caption, which is a key aspect of sarcasm, and is tested on a multimodal sarcasm detection dataset to compare the results with other state-of-the-art models.

3 Methodology

To reproduce the results stated in the original paper we mainly used the code made available by the authors in their public repository (TODO: link). The instructions provided were partially incomplete, and the code needed some small adjustments. We used a Colab notebook to run the experiments, which allowed us to leverage the NVIDIA V100 GPU.

3.1 Model description

The DynRT model is composed of three main parts: two encoders and a dynamic routing transformer. The two encoders are used to encode the input text and the image features, and are respectively RoBERTa (TODO: cite) for the text and ViT (TODO: cite) for the images. The dynamic routing transformer is used to capture the incongruity between the two modalities. It is composed of a stack of N layers, each of which is composed of a multi-head co-attention routing module, a multi-head self-attention module and a feed-forward network; each module is followed by a residual connection and a normalization layer. The transformer leverages hierarchical co-attention to module the incongruity between texts and images. The output of the encoders is fed to the dynamic routing transformer, which outputs a vector of *routed features* that are used for the classification together with the *image features*. The output of the classification is a vector containing the probabilities of the input being sarcastic or not.

3.2 Datasets

The dataset used to evaluate the architecture is the Multimodal Sarcasm Detection (MSD) dataset (TODO: cite), whose composition and split information are shown in table 1. The dataset is not perfectly balanced, but it reflects the real-world distribution of sarcastic and non-sarcastic sentences. The dataset is composed of sentences and their corresponding images. The preprocessing operations performed on the dataset before feeding it to the model are the following:

- Discard tweets with regular worlds, such as *sarcasm, sarcastic, irony, jokes, humor, reposting, ironic, exgag*, and URLs
- replace mentions with the token <*USER*>

Table 1. MSD dataset composition and split information

	Train	Development	Test
Sarcastic	8,642	959	959
Non-sarcastic	11,174	1,451	1,450
Total	19,816	2,410	2,409

The dataset can be downloaded from https://github.com/headacheboy/data-of-multimodal-sarcasm-detection.

3.3 Hyperparameters

The configuration used to run the experiments is pretty similar to the one used by the authors (i.e. the hyperparameters that could have influenced the performance of the model, such as the learning rate, the batch size, etc., were left unchanged) in order to perform a fair comparison of the results. The only difference in our runs was in the parallelization of the training process, which was not configured by default. Initially we tried without it, but the training process was excessively long. Therefore, we increased the number of workers to 10. In this way we could run the whole training process (15 epochs) in less than 4 hours.

3.4 Experimental setup and code

Since we used a Colab notebook to run the experiments, we created a copy of our repository in a Google Drive folder, and we mounted the drive in the notebook. Then we tried installing the requirements in the way described in the readme.md of the original repository (TODO: link) but we encountered some problems due to incompatible versions of some libraries. Therefore, we manually changed the installation process by updating the problematic libraries to their latest version. We then proceeded to the preprocessing phase as shown by the authors, which consisted in performing the cleaning operations on the dataset and in converting the images to tensor. This last operation was performed locally, since we encountered some issues when trying to run it on the Colab notebook. We then uploaded the preprocessed dataset to the notebook and we started the training process. Here, we encountered other issues. Firstly, we had no feedback about how the training process was proceeding, therefore we decided to increase the verbosity by adding a code line that printed the advancement of the number of batches processed (with the default batch size of 32 we had to elaborate 611 batches for each epoch). Secondly, we got an exception due to the loading of some .npy files containing the tensors (previosly created starting from the images in the preprocessing phase). Specificaly, some files couldn't be serialized in the way the algorithm was originally written. To solve this, we added an exception handling code block to manage these files. Finally, since Colab's stability is not complety reliable, we added a simple script in our notebook that leverages the checkpoint system implemented by the original authors: it checks whether a checkpoint file already exists or not, and if it does it loads the model from it (in case of multiple checkpoints models, it load the last one). After these changes, we were able to run the training process without any other issues.

3.5 Computational requirements

As already mentioned, we used the NVIDIA V100 GPU provided by Colab Pro to run the experiments. The whole training and evaluation process took about 4 hours to complete, and we experienced a GPU memory consumption of about 4 GB. It is interesting to note how with no parallelization a single training epoch took about 3 hours to complete, which would have led to an excessively long training process of 45 hours for 15

Table 2. Results obtained by the original authors and by us

	Original	Reproduced
Accuracy	93.49	93.59
F1	93.21	91.93

epochs. This would have been an issue since Colab only allows for 12 hours of continuous execution.

4 Results

4.1 Results reproducing original paper

The results obtained are in line with the ones provided in the paper. The authors report accuracy and F1 metrics obtained by averaging the performance of five different runs, showing how the newly introducted architecture outperforms other state-of-the-art models. The results that we obtained, shown in table (TODO: table) confirm those claims for accuracy results, but the F1 score obtained by us is slightly lower.

5 Discussion

TODO: talk about how the learning process was increasingly getting better (it had a spike) Give your judgement on if your experimental results support the claims of the paper. Discuss the strengths and weaknesses of your approach - perhaps you didn't have time to run all the experiments, or perhaps you did additional experiments that further strengthened the claims in the paper.

5.1 What was easy

Give your judgement of what was easy to reproduce. Perhaps the author's code is clearly written and easy to run, so it was easy to verify the majority of original claims. Or, the explanation in the paper was really easy to follow and put into code.
Be careful not to give sweeping generalizations. Something that is easy for you might be difficult to others. Put what was easy in context and explain why it was easy (e.g. code had extensive API documentation and a lot of examples that matched experiments in papers).

5.2 What was difficult

List part of the reproduction study that took more time than you anticipated or you felt were difficult.
Be careful to put your discussion in context. For example, don't say "the maths was difficult to follow", say "the math requires advanced knowledge of calculus to follow".

5.3 Communication with original authors

Document the extent of (or lack of) communication with the original authors. To make sure the reproducibility report is a fair assessment of the original research we recommend getting in touch with the original authors. You can ask authors specific questions, or if you don't have any questions you can send them the full report to get their feedback before it gets published.

References

1. Y. Tian, N. Xu, R. Zhang, and W. Mao. "Dynamic Routing Transformer Network for Multimodal Sarcasm Detection." In: **Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**. Ed. by A. Rogers, J. Boyd-Graber, and N. Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 2468–2480.