RESCIENCE C

Replication / ML Reproducibility Challenge 2022

Reproducibility Report - Natural Language Processing

Authors of the reproducibility report:
Andrea Carotti - University of Illinois Chicago
Emilio Ingenito - University of Illinois Chicago

Paper: Imputing Out of Vocabulary Embeddings with LOVE Makes Language Models Robust with Little Cost
Authors: Lihu Chen, Gaël Varoquaux, Fabian M. Suchanek

Date
28 November 2022

1 Introduction

The paper we tried to reproduce is *Imputing Out-of-Vocabulary Embeddings with LOVE Makes Language Models Robust with Little Cost*, that tries to address the problem of extending the word representation of an existing pre-trained language model like FastText and makes it robust to Out Of Vocabulary (OOV) words. The paper uses a simple contrastive learning framework called LOVE that follows the principle of mimick-like models to generate vectors for unseen words. It is demonstrated that the model achieves similar and better performances than prior competitors, both on original datasets and on corrupted variants. In addition, the performance of the model used in a plug-and-play fashion with FastText (see 4) and BERT (that we were unable to reproduce) shows the increase in robustness of the model.

2 Scope of reproducibility

As stated by the authors, there are two main methods to evaluate word representations: Intrinsic and Extrinsic. Intrinsic evaluations measure syntactic or semantic relationships between words directly, while extrinsic evaluations measure the performance of word embedding as input features to a downstream task. Our goal was to reproduce the results obtained by the authors through both, intrinsic and extrinsic tasks. Delving into details, the scope of our study was to reproduce the following claims:

- with regard to intrinsic tasks, the LOVE model outperforms MIMICK-like models (MIMICK, BoS and KVQ-FH) for the following datasets: SimLex (accuracy of 35%), MTurk(accuracy of 62%), MEN(accuracy of 68.8%), WordSim(accuracy of 55.1%), SimVerb (accuracy of 29.4%) and the avgerage accuracy on all the model tested is greater than the best performing MIMICK-like model by 6 percent points (49.7%);
- considering the performance on the extrinsic tasks, LOVE have a slightly higher accuracy than MIMICK-like models on the SST2 and MR datasets (it achieves an accuracy of 81.4%, 74.4%, respectively). On CoNNL-03 and BC2GM it achieves an accuracy of 78.6% and 64.7%, respectively. When the datasets are corrupted by 70% (typos are generated by simulated errors of an OCR engine), the model achieves the following accuracies: 73.2% (SST2), 66.7% (MR), 69.7% (CoNNL-03), 63.8% (BC2GM);
- LOVE can be used in a plug-and-play fashion with FastText, where it improves its robustness: the declared increase in robustness, on the SST2 dataset, is reported in the following table.

Model	original	10% typo	30% typo	50% typo	70% typo	90% typo
FastText	82.3	68.2	59.8	56.7	57.8	60.3
FastText+LOVE	82.1	79.8	74.9	74.2	68.8	67.2

3 Methodology

In order to reproduce the results, the author's code was used. It was provided in the original paper and and can be found in this repository.
Parts of the pipeline needed to be re-implemented: indeed, the author did not include in the original repository some minor scripts to perform basic editing on the dataset. Furthermore, not all the datasets were provided by the authors, and some of them required special privilege to be obtained. Nonetheless, after some research most of them (MR and BC2GM) were obtained (section 3.6.2).
We have provided a Jupiter Notebook which contains a step-by-step guide through our analysis.

3.1 Computational Requirements

In order to reproduce the author's code, 4 different machines were used:

- Personal Computers:
  - Macbook Pro 2017 with 2,5 GHz Dual-Core Intel Core i7 and 16 GB of RAM
  - Macbook Pro 2017 with Intel Core i5 and 8 GB of RAM
- A remote Google Cloud GPU instance:
  Intel Skylake, 1 x NVIDIA T4, Debian 10 based Deep Learning VM with M100, OS: Deep Learning on Linux with 50 GB
- The CPU available from Google Colab PRO:
  NVIDIA T4, CUDA Version 11.2, Driver Version 460.32.03

3.2 Missing datasets

To perform the intrinsic and extrinsic evaluations a total of 12 datasets were used. Some of them though were missing:

- AP (intrinsic cluster evaluation) - Not found
- BLESS (intrinsic cluster evaluation) - Not found
- MR (extrinsic text classification) - Found and tested
- BC2GM (extrinsic NER) - Found and tested

3.3 Missing corrupted datasets

All the corrupted datasets for extrinsic tasks were also missing. The problem was solved by creating custom scripts in order to corrupt them.

This was done using the function *get_random_attack()* located in the *attacks.py* file in the root of the project: this function augments a specific provided word, and modifies it by applying with a certain probability (passed as a parameter) one out of the five augmentation - 'swap', 'delete', 'insert', 'keyboard', 'synonym', 'unchange' (this one doesn't modify the word). We want to emphasize that it is not specified by the author the probability

used for every augmentation of the dataset (only 0.07, 0.07, 0.07, 0.07, 0.36, 0.36 was provided for the general case in the extrinsic evaluation).
We assumed the same probability for 'swap', 'delete', 'insert', 'keyboard' and the same also for 'synonym', 'unchange'. In addition, the synonym file, as provided by the author, is a small file of 13 words, so, every time the function to augment a word is called, it doesn't find in this small file the synonym of the word considered, and as a result the target word to change remains unchanged.

3.4 Reproducing the results

The results were replicated using our personal machines (using PyCharm Edu): to do so we removed all the calls to CUDA. Eventually we moved to Google Colab to reproduce the entirety of our results and to provide proofs of it. Google Colab was set up using the upgraded version because the computation would have required too long (1 hour and 50 minutes for each extrinsic task).
On our personal machines the intrinsic tasks were fast to compute and did not require to wait much to get results (see section 3.8 for further details). Instead, for the extrinsic tasks, the text classification analysis (on SST2 and MR datasets) required approximately 2 minutes (10 epochs) for both datasets. For NER (CoNLL-03 and BC2GM) the waiting time was longer, approximately 1 hour and 50 minutes (100 epochs), for both datasets. The training time was approximately the same using both the corrupted and the original datasets.

3.5 Model descriptions

For the entirety of our analysis we used the pre-trained model provided by the author, even if we tested the training of the model on remote machines hosted by Google. The model, LOVE (Learning Out-of-Vocabulary Embeddings) can be used both alone (original in the paper) or in a plug-and-play fashion with FastText or BERT.

The original love uses 6.5M parameters and FastText as target vector. The model can also be used with BERT, but we were unable to reproduce the results even after contacting the author.

3.6 Datasets

Several datasets were used by the author to test the model, here we provide a carachterization of them:

Intrinsic evaluation: — For the intrinsic evaluation we did not perform any preprocessing on the dataset, we provided the links to the original datasets, even if the author already preprocessed them and provided them in the repo.

Dataset	Pairs	Scale	link
RareWord	2034	0-10	[download]
SimLex	999	0-10	[download]
MTurk	771	1-5	[download]
MEN	3000	1-7	[download]
WordSim	252	0-10	[download]
SimVerb	3500	0-10	[download]
AP	na	na	na
BLESS	na	na	na

Extrinsic evaluation: — For the extrinsic evaluation was necessary to perform preprocessing on the text get the corrupted dataset with typos (10%, 30%, 50%, 70%, 90%), and specific scripts to do so were implemented, one for the text classification task and one for the NER task.

We had to split the MR dataset into train, test and development sets: in order to complete this task, we empirically found how the already given datasets were splitted (SST2 and CoNLL-03): 72% train, 19% test, 9% developer, and splitted the unkownn dataset manually.

Instead, BC2GM was found on the web already splitted, so no splitting was performed on this one. We were able to retrieve no further information about this dataset.

Dataset	Train Samples (72%)	Dev Samples (9%)	Test Samples (19%)	Balanced Labels
SST2	6920	872	1821	Yes
MR	7676	2024	962	Yes

CoNLL-03	Articles	Sentences	Tokens
Train (72%)	946	14,987	203,621
Dev (9%)	216	3,466	51,362
Test (19%)	231	3,684	46,435

3.7 Hyperparameters

The hyperparameters in the code provided were already set as stated in the paper. The only exception is for the extrinsic evaluation of SST2 and MR. While on the github repository it is clear that the author set 10 epochs to evaluate the performances, in the paper - as described in Table A2 - only 5 epochs were used. Here we report the hyperparameter for the extrinsic task.

Hyperparam	SST2	MR	CONLL-03. 2	BC2GM
model	CNN	CNN	BiLSTM+CRF	BiLSTM+CRF
layer	1	1	1	1
kernel	[3,4,5]	[3,4,5]	-	-
filter	100	100	-	-
hidden size	300	300	300	300
optimizer	Adam	Adam	SGD	SGD
dropout	0.5	0.5	0.5	0.5
batch size	50	50	768	768
epoch	**5**	**5**	100	100

Then, we used different learning rates (already provided by the author after finetuning for FastText), and we used all of them to run every extrinsic task. These learning rates are: {5e−3, 3e−3, 1e−3, 8e− 4, 5e − 4}.

The hyperparameters for the intrinsic task were already provided in the code, and were used for all the datasets, as can be seen in Table 2 of the original paper.

3.8 Experimental setup and code

Initial setup — To get the results first we tested the author code on PyCharm EDU on our **personal machines**. We installed the missing dependencies (nlpaug dependency was missing also from the requirements.txt file) and removed the calls to CUDA inside the

Hyperparam	Value
model	pam
batch size	32
drop rate	0.1
layer	1
epochs	20
gamma	0.97
hard negatives number	3
learning rate	0.002
loader type	hard
loss type	ntx
lowercase	True
merge	True
shuffle	True

code. To fix the additional problems due to removing cuda calls it was necessary to substitute in the code:

```
model.load_state_dict(torch.load(model_path))
```

with

```
model.load_state_dict(torch.load(model_path, map_location=torch.device(
    'cpu')))
```

Final setup — Eventually after the results were obtained we reproduced them on **Google Colab**. This because the computation was faster and in order to make clear to the reader how we obtained the results step-by-step.

The author's github repository was uploaded on google drive, and then accessed from Colab.

Here you can find the step-by-step to get the results.
The measure used to evaluate the experiments were:

- Spearman's ρ for word similarity on intrinsic tasks
- Accuracy for text classification on extrinsic tasks
- F1 for NER on extrinsic tasks

Average runtime to compute the intrinsic tasks

— For every dataset the compute time was almost immediate (less than a minute)

Average runtime to compute the extrinsic tasks

— We report the time for each epoch (the batch size is the same as stated in the hyperparameter section above), for 5 epochs and the total time to get the best accuracy among the different learning rates tested:

Dataset	Avg Epoch time	Total for 5 epochs	Total for 5 learning rates
SST2 original	10s	50s	4m 10s
SST2 + typo	11s	55s	4m 35s
MR orginal	12s	1m	5m
MR + typo	12s	1m	5m

Dataset	Avg Epoch time	Total for 100 epochs	Total for 5 different learning rates
CoNLL-03 original	20s	27m	138m
CoNLL-03 + typo	19s	31m 40s	2h 40m
BC2GM original	30	50m	4h 10m
BC2GM + typo	31	51m 40s	4h 18m

4 Results

Our results support the main claim of the original paper. In all the three task we tried to reproduce, i.e. intrinsic evaluation, extrinsic evaluation and robust evaluation, even though we did not always obtain the same results as the authors, we were able to make the same overall conclusions.
Hence, for each claim of section **2**, here we provide a corresponding section, where we show the results of our analysis and compare them with the ones provided by the authors.
Depending on the task, these are the metrics used to compare the performances:

- Intrinsic task: Spearman's ρ similarity
- Extrinsic task: Accuracy and F1

4.1 Intrinsic Evaluation

Here are summarized the results we obtained running the intrinsic tasks. As you can see, by comparing them with the accuracies reported by the authors (please check section 2), we achieved the same results, with a difference *error* of at most 1.7 percentage points (on the dataset MTurk). Hence, the first claim is verified, since the model still outperforms MIMICK-like models.

RareWord	SimLex	MTurk	MEN	WordSim	SimVerb
42.65	35.02	63.77	68.4	55.89	28.72

4.2 Extrinsic Evaluation

Here are summarized the result we obtained running the extrinsic tasks on differet datasets. With regard to the *original* datasets, we achieved an accuracy which is at most 10 percentage points different from the one achieved by the authors (most significant difference on the BC2GM dataset). On the other hand, dealing with corrupted datasets, we achieved an accuracy which is at most 37.85 percentage points different from the one achieved by the authors (most significant differece on the BC2GM dataset). Hence, the second claims is verified, since the model still has a slightly higher accuracy than MIMICK-like models on the SST2 and MR datasets.

SST2	SST2+typo	MR	MR+typo	CoNLL-03	CoNLL-03+typo	BC2GM	BC2GM+typo
79.96	71.21	73.92	66.17	83.41	66.17	54.09	25.95

4.3 Robust evaluation: FastText + LOVE

As stated by the authors, LOVE can we used in a plug and play fashion, combined with FastText. In this section we analyze the results of the *robust evaluation*, meaning that we evaluated the performance of the model on a specific dataset (SST2 and CoNLL-03),

which was corrupted with different typo probabilities (from 10% to 90%). As you can observe, comparing these results with the ones of section 2, we get an accuracy on the SST2 dataset which is at most 3 percentage point different from the one claimed by the authors (most significant difference on the dataset with 50% of typo probability). Similar analysis for the CoNLL-03 dataset, with the most significant difference of 3.75 percentage points (on the dataset with 50% of typo probability). Hence, the third claim is verified, since FastText+LOVE outperforms FastText.

Extrinsic Evaluation Love+FastText on SST2

original	10%	30%	50%	70%	90%
83.41	-	-	73.75	71.21	69.67

Extrinsic Evaluation Love+FastText on CoNLL-03

original	10%	30%	50%	70%	90%
-			72.19	66.17	63.1

5 Discussion

The results that we obtained strongly support the original claims of the author. The strength of our approach is the fact that we were able to reproduce the results both on our personal machines (removing CUDA calls) and on the remote machines. Furthermore, the author suggestion to use the get_random_attack() function to corrupt the dataset was essential to perform most of the experiments. We were able to demonstrate all the results obtained by running the commands on a Google Colab notebook and can be easily verified. Unfortunately we were not able to verify the results on the model trained by us .

5.1 What was easy

Not all the claims were easy to verify. Training the model and the evaluation of the intrinsic tasks were straightforward, since these tasks required to execute commands already provided on the github readme. Furthermore it was easy to use the already provided datasets, since the authors already processed them. In a nutshell, the easiest tasks were the following:

- train the model (although it required several hours);
- perform the intrinsic evaluation;
- generate the embeddings for already processed datasets (for the extrinsic tasks, SST2 and CoNLL-03).

5.2 What was difficult

It was instead harder to reproduce the extrinsic tasks. The first reason is that not all the datasets were provided. MR and BC2GM were found on the web to reproduce results, and as a consequence, we had to process it with custom made scripts. The second reason is that it is not completely clear how the author corrupted the datasets. Some function to perform augmentation were provided but how to run those on each dataset was not.

Furthermore we think that in extrinsic evaluation it was not clear that the scripts provided by the authors were meant **not for the specific datasets** but for the specific task. So at the beginning we wrongly expected a script for each dataset. As a consequence we mistakenly thought that MR text classification couldn't be performed. The same reasoning was done for BC2GM: it was crucial the communication with the authors of the paper, who made this clear to us.
Finally, it was hard to understand how to use the LOVE model in a plug-and-play fashion with FastText.

5.3 Communication with original authors

We had the opportunity to communicate with the original author of the paper, and exchange a couple of mails. The content of the question regards MR and BC2GM, two datasets that we were not able to find. And the second questions concerns the typo, and how to reproduce the result with the corrupted dataset (we asked how he added the typos in an incremental way). The author was very fast and replied accurately to our questions. He told us to use the get_random_attack() function in attacks.py .

The second question concerns how to modify "the corresponding path for loading pretrained parameters" as stated in the github repo. To us, it was not clear how to reproduce the results of Love+FastText and Love+BERT. The author told us that in order to reproduce the result of FastText+LOVE, we had to follow the following procedure:

- generate the embeddings of in-vocabulary words through FastText;
- generate the embeddings of out-of-vocabulary words through LOVE;
- load the LOVE model trained on FastText, the embeddings for all words (in and out of vocabulary) into the downstream task (e.g. classification).

6 Further observations

Probabilities of word augmentation — While reproducing the paper we found that some information regarding how the typo were obtained were not provided. Particularly it was not clear what probabilities were used for Love+FastText to corrupt each word of the dataset. We knew from the tables that the probability of a typo was 10% 30% 50% 70% and 90% but it was not specified the probability of the specific augmentation of a single word (e.g. 'swap', 'delete', 'insert', 'keyboard', 'synonym', 'unchange'). So we assumed equal probability for the actions that modify a single word (*swap*, delete, *insert*, *keyboard* and *synonym*) and 1 - (P(swap) + P(delete) + P(insert) + P(keyboard) + P(synonym)) for *unchange*.
For example if the typo probability is 70% the distribution assumed was:

- swap: 0.14
- delete: 0.14
- insert: 0.14
- keyboard: 0.14
- synonym: 0.14
- unchange: 0.30

Unfortunately we noticed that the synonym file provided by the author contained **only 13 words**. For this reason the synonym didn't actually change most of the time the words processed by the augmentation function.

Missing modules after installing repository — We noticed that these modules were not included in the *requirements.txt* file:

- nlpaug
- tensorboard

Conflict problems regarding the training on remote SSH — After running a remote instance on Google Cloud (Debian 10 based Deep Learning VM with M100, OS: Deep Learning on Linux with 100 GB). And cloning the original repository of the author, we tried to install the requirements with the standard *pip install -r requirements.txt* but unfortunately after doing so the following problem occurred:

```
ImportError: /lib/x86\_64-linux-gnu/libstdc++.so.6: version ‘GLIBCXX\_3
    .4.26’ not found
```

To solve this it was necessary **not** to install the requirements using the command *pip install -r requirements.txt* but by installing each requirement alone one-by-one because of conflicts inside the machine.

Extrinsic task evaluation problem — In order to run the *main.py* script to perform the evaluation task, initially a problem was encountered because the shell doesn't have permission to execute the file *conll.txt*.
Hence, we had to change the privileges of the file using the command:

```
chmod u+x conlleval
```