NLP Project: Fake news challenge

Maxime Reynouard, Lucas Gnecco and Aurèle Goetz

IASD 2020-2021

Introduction

This report gives an overview of the work we carried on for the project about fake news detection as part of the course Natural Language Processing. In a first part, some contextual elements along with the plan of our work are given. Then, a baseline from the fake news challenge is implemented. Lastly, we present our personal contributions and trials to get better results with some more complex and recent approaches.

1 Description of the challenge and work plan

1.1 The task

The Fake news challenge stage 1 (FNC-1) is a public challenge from 2017. The goal of this first step is not to detect whether a piece of news is verified or fake, but to make stance detection which is an important first step towards fake news detection. The challenge proposes a 4-stance classification task between a headline and a text body. The different classes that could be predicted are the following:

- Agree, if the article supports the headline.
- Discuss, if the same topic is covered but without taking side.
- Disagree, if the article contradicts the headline.
- Unrelated.

As mentioned before, the task of stance detection is viewed by the organizers of the challenge as a first good step towards future autonomous fake news detection. More importantly, it is considered a much simpler task and therefore much more feasible. Since the end of the challenge, numerous teams have proposed new models to fulfill the task of stance detection using all sorts of techniques, from pure feature engineering [3] to transformers [9].

1.2 The data

The dataset is composed of 2587 headlines and bodies. These were extracted from articles about 300 topics. 200 topics were selected for the training set and 100 for the test set. To generate examples of the unrelated class, some headlines and bodies were randomly associated. It has to be underlined that the dataset is highly imbalanced. Among the 75385 headline-body pairs, 72.8% are unrelated, 17.7% of the "discuss" class, 7.4% of the "agree" class and only 2% of the "disagree" class. As discussed in [2], this imbalance makes it necessary to carefully choose the metric used to evaluate classifiers. According to them, the scoring metric used in the challenge was not well suited and favorised classifiers that performed poorly on the minority classes that are the ones we care about classifying. They propose a weighted F1 score as a better metric to evaluate performance.

Concerning the structure of the data, the bodies are composed of 2 to 5000 words, while the headlines range from 2 to 40 words. This is one of the points that make this task challenging. With such a variability in the size of the input and the possibility for the body to be a long piece of text, having a model that captures well the dependencies between the words and the sentences is difficult. In 2017, the attention based models like transformers were not popular yet and the LSTM were just starting to be massively developed. Among the best teams of the challenge, nobody used a LSTM model.

1.3 The results of the challenge

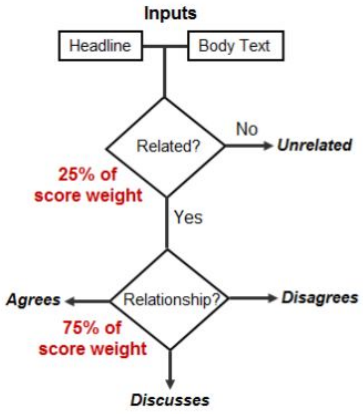

Figure 1: Scoring method

A quick word about the scoring: Figure 1 describes roughly how the score is computed. If we correctly label the input as *unrelated* we get 0.25 points, if we correctly label it within the *related* set (*agree*, *discuss* or *disagree*) we also get 0.25 points. Lastly, if we correctly label it inside this set, we get an additional 0.75 points.

Surprisingly enough, the models used by the three best teams ranked for the FNC-1 are very simple ones. The first team used an average model of a deep CNN operating on word2vec embeddings and a gradient-boosted decision-tree model. The two other teams used simple MLP, putting more efforts in the input pre-processing. The most interesting models is the third one, called "a simple but tough-to-beat baseline" [8], which consists in a MLP with only one hidden layer! The input is

a simple frequency vector (TF-IDF) counting the occurrences of the 5000 most frequent words in the headlines and bodies. More details about this architecture and the results it gives are given in the next part of this report.

1.4 Plan of our work

The extreme simplicity of the previously mentioned solution that ranked third for the challenge intrigued us. We decided to firstly take some time to implement their solution and test it on our side. This is presented in the second part of this report. Then, some more personal results are given in the third part. There, we tried to improve over the baseline solution by providing richer embeddings of the headline-body pair (not only the number of occurencies of the words). The NLP lecture about word embeddings inspired us and we thought combining it with LSTMs in order to obtain complex sentence/text embeddings that would provide a more powerful input for this stance classification task. We were hoping these embeddings would carry much more semantic meaning and thus be helpful for solving stance detection. Different word embeddings will be tested and provided as input to a bidirectional LSTM, which will then provide the embeddings of the body and the headline to an MLP similar to the baseline that will be presented in the next part.

2 Reproducing a strong baseline

In this section we will present the model proposed by University College of London NLP team (UCLNLP). We will explain the differences in our implementation and compare our results with the ones presented in UCLNLP's article [8].

2.1 UCLNLP's model

Only two simple bag-of-words (BOW) representations are used for the text inputs: Term Frequency (TF) and Term Frequency Inverse Document Frequency (TFIDF). Then to get the features of a headline-body pair we concatenate :

- the TF representation of the headline
- the TF representation of the body
- the normalised TFIDF vectors' cosine similarity

The vocabularies used to calculate the vectors are 5000 most frequent words with some stop words. We use only the training set for the TF vectors, and both sets for the TFIDF vectors.

This bring us to inputs of size 10001, which are then fed into the model with one hidden layer of size 100, and an output layer of size 4 (for each stance). The ReLU activation function is used. A schematic of the model is presented in Figure 2

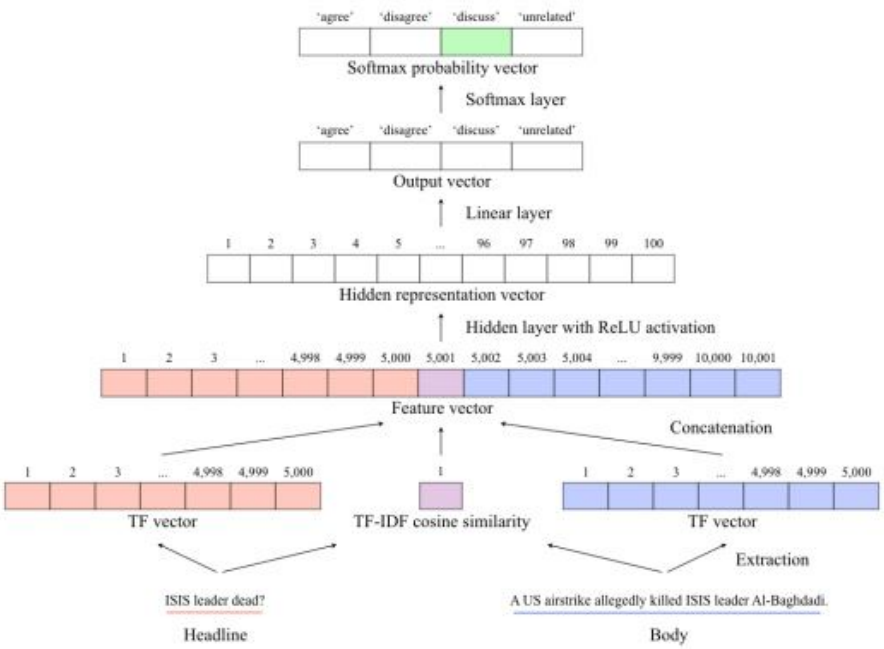

Figure 2: Schematic diagram of the baseline model

The training is then pretty standard: Adam optimizer, slight $L2$ regularization, drop-out. Note that even though the scoring system is not equivalent for each class 1, the loss function was not designed to optimize the score and treats each class as equivalent in terms of scoring.

2.2 Difference in our reproduction of the baseline

As our project took place long after the challenge was complete, we could use the answers for the test set as they were published at the end of the challenge. We therefore did not need to employ any cross validation methods. We just used the test set as a validation set. We also did not need to optimize hyper parameters as we just used those of the article [8].

Another difference was the early stopping, as the criterion's description used by UCLNLP is vague we were not sure how to reproduce it, so we just did not.

Finally a third difference was the modernity of the libraries we used. Quite early in the project, we decided to transpose the code to up to date libraries in case we wanted to re-use part of the work in the rest of our project.

2.3 Results comparison

With this method we managed to get scores around 9600 which is even better than the first place of the challenge. The top-3 scores of the challenge were:

1. 9556.50 - Talos Intelligence
2. 9550.75 - TU Darmstadt
3. 9521.50 - UCLNLP

So we are sensibly ahead of the first place, and quite far ahead of the results UCLNLP had. This is probably explained by the differences we just mentioned. As we could train on the whole training set without cross validation, we had a bigger equivalent data set. The early stopping criterion might make things go both ways, if it was not well designed maybe some over fitting is better for the score. And finally, even though it should not change anything, there might still be improvements hidden in the more recent version of the tensorflow and keras libraries.

3 Personal approach: use embeddings and LSTM networks to improve over the baseline

3.1 General ideas and architecture presentation

Using only TF-IDF vectors for this task makes it very simple to implement (as for the previously mentioned solution) but appeared too low-level compared to the possibilities that exists today in natural language processing. We imagine easily that the TF-IDF vectors help the model to look for common words that would indicate whether the headline and body are covering the same topic. This already makes a good model due to the imbalance of the dataset (at least with respect to the metric of the challenge). However, we hoped that the accuracy for the minority classes could be improved by capturing some more semantics of the data. That is why we decided to improve the headline and body embeddings that we would feed into the MLP with some more complex ones. The different word embeddings that we used are listed in the next part. With these pre-trained word embeddings, we fed a bi-directional LSTM model and used the concatenation of both outputs as the input of a fully connected layer. This is shown on Figure 3.

The embedding process described on Figure 3 is applied to both the headline and the body with shared weights in the LSTM part. However, the fully-connected layer is not shared. Then the two text embeddings are concatenated and given as input to the MLP that was described in the previous part. Our intention was to directly compare the quality of the embeddings by keeping the rest identical.

3.2 Preliminaries

3.2.1 Preprocessing and vocabulary building

To pass from raw text to the input of our model we relied on two libraries: `torchtext` and `gensim`. We also used some features of other libraries such as `spacey`. In the end, our preprocessing is just a list of functions to be applied to a raw text input, mainly containing specialized functions from `gensim.parsing.preprocessing`.

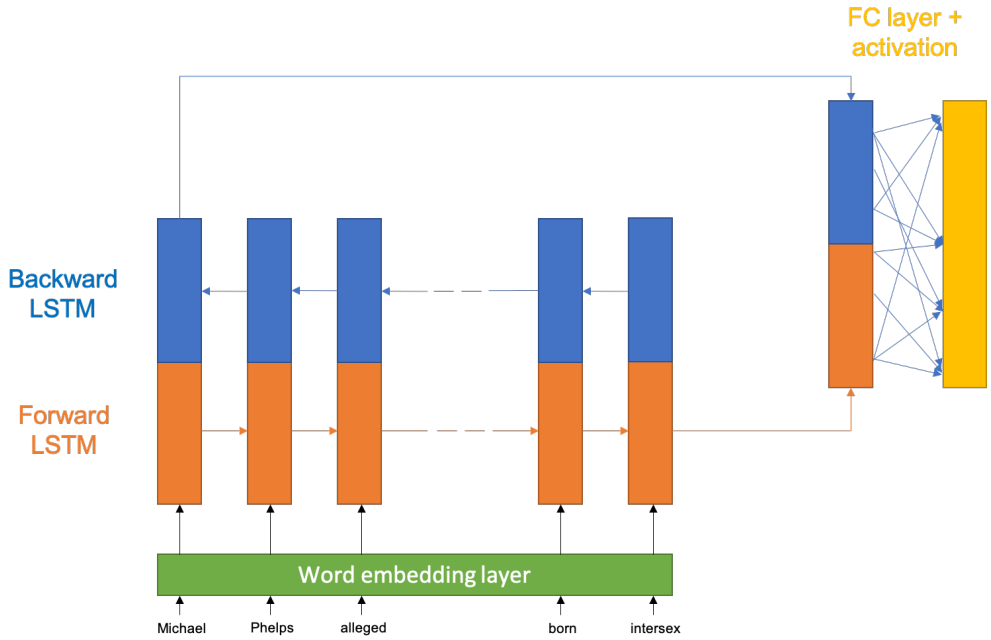

Figure 3: Embedding of the headlines and bodies (the size of the latent representation in the LSTM is 128 and the number of neurons for the fully-connected layer is 100)

Figures 4 and 5 show examples of the processing done on pieces of text. To create the whole pipeline from raw text to indices that feed an embedding with a look-up table, we relied on `torchtext`. The `Field` object now available in `torchtext.legacy` allows the creation of such pipeline, adding padding when necessary and start and end of sentence tags if needed.

```
ISIS might be harvesting organs, Iraq tells UN
------------------------------------------------
['isis', 'harvesting', 'organs', 'iraq', 'tells']
```

Figure 4: Example of a processed sentence

```
Found! Missing Afghan Soldiers Spotted Trying to Enter Canada
-------------------------------------------------------------------------
['missing', 'afghan', 'soldiers', 'spotted', 'trying', 'enter', 'canada']
```

Figure 5: Example of a processed sentence (2)

3.2.2 Word embeddings

We focused our experimentation on trying out different word embeddings. Our hypothesis was that the more complex LSTM model would be already more powerful for the stance detection task, and that by changing its initial word embedding input we could obtain different levels of performance. We considered training our own word embeddings using the FNC data, but the corpus was too small and had a lot of misspelled words (or words in other languages that use other alphabets). We also considered fine tuning pre-trained word embedding models, but this was not possible with

all of them. This happened particularly with the model that most interested us, the `Google News` word2vec. As we were not going to be able to fine tune all the embeddings, we preferred to leave them all as they were.

We worked for a while with some of these pre-trained word embeddings in `torchtext` and `gensim`, and we noticed that some of them like the `Google News` Word2Vec were able to capture the semantic information we wanted, while others such as `fastText` were better at finding similar words depending on the syntax. As we saw in section 2.3, the vectors used by the model submitted to the FNC were already giving some valuable information to the used model, so syntax-like embeddings such as `fastText` could be helpful even if they don't capture the information we thought would be more beneficial. Some examples of the difference between the `Google News` Word2Vec and `fastText` can be seen in the notebook we wrote, where some similarity queries are performed to evaluate what the two models capture in their representations.

In our final tests we had a grand variety of pre-trained models offered in `torchtext` (See the documentation for more details) and the two other models we mentioned before, loaded via `gensim` The notebook where we implemented the LSTM model allows the use of 8 different word embeddings. More are available but were left out for brevity. Here is a brief description of the available embeddings.

- **word2vec**: On the first models for creating word embeddings. The two original training methods are detailed in [6] and [5]. The `Google News` word2vec was trained using this method, and it is also worth noting that other methods such as `fastText` are based on this one.
- **fastText**: based on `word2vec`, `fastText` can be trained taking into accounts n-grams, which makes the embedding more aware of morphological information about words. The idea is detailed in [1]. See also [4]. A simple example of the difference between plain `word2vec` and `fastText` using `gensim` is available here.
- **charngram**: Originally proposed in [10], this embedding is also subword-aware as it embeds n-grams instead of whole worlds. The representation of a sequence of characters (a word for example) is the sum of its n-gram embeddings, passed over a non-linear function. In [10] authors even use it to produce sentence embeddings, showing it performed well on sentence similarity tasks.
- **GloVe**: Developed at Stanford, `GloVe` is trained on aggregated global word-word co-occurrence statistics from a corpus. The method is detailed in [7], and a very helpful resource for a quick overview is the Standford page, from were we cite the following intuition behind `Glove`: "*The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning*".

3.3 Results and discussion

The results we obtained were surprisingly bad. Having been able to replicate the challenge baselines, we thought adding the LSTM with pre-trained word embeddings before the MLP would be beneficial, but it proved to be the complete opposite. Moreover, this bad performance was the same for every embedding we used. Figures 6 and 7 show the evolution of the training and validation

loss over the epochs. We see a very particular behaviour: validation loss is not only not improving as training goes on, but is actually getting worse. It mirrors the evolution of the training loss.

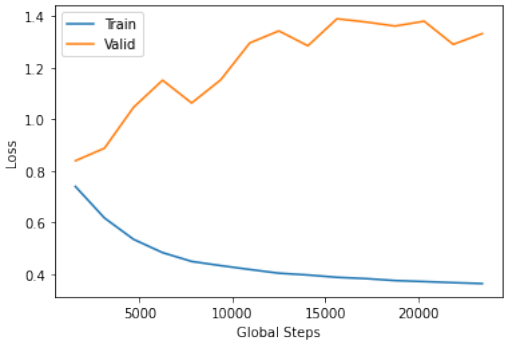

Figure 6: Loss using the `Google News` word2vec pre-trained word embedding

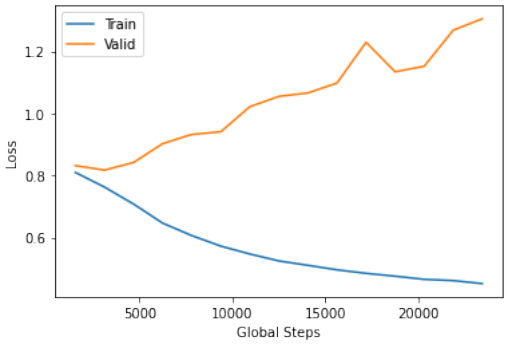

Figure 7: Loss using the `fastText` pre-trained word embedding

When looking at the accuracy in each class (Figures 8 and 9), we see that the model is clearly over fitting on the train set. The accuracy of the majority class is preserved (the model starts predicting almost everything as in the majority class) but then starts to better classify the other three classes. At the same time, on the validation test, the improvement on the minority classes is marginal. The wrong classifications that emerge on the majority class make the general performance much worse over all. We observed the same behaviour no matter the embedding.

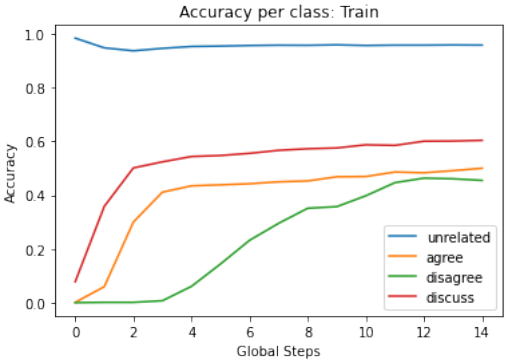

Figure 8: Accuracy per class using the `Google News` word2vec pre-trained word embedding on the **training set**

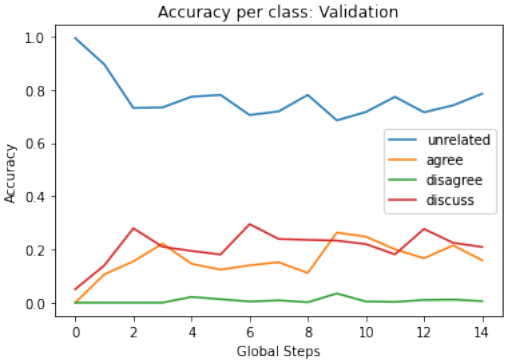

Figure 9: Accuracy per class using the `Google News` word2vec pre-trained word embedding on the **validation set**

We were very intrigued after this shocking discovery. Not knowing what was the exact reason behind this, we tried some changes hoping the model would be able to at least show a decreasing validation loss. First we tried to decrease the size of the model, believing that the great number of parameters could contribute to the negative behaviour through over-fitting starting at the first epoch. This change had little to no effect whatsoever.

Another change we made searching for better results was to eliminate the LSTM and just feed average word embeddings into the last MLP part. This idea was considered in [10], where the n-gram embeddings were used to create sentence embeddings for sentence similarity tasks. It also felt similar to the reproduced baseline: creating some features to feed an MLP. Nevertheless, the

features fabricated by the team we reproduced the baseline from proved to be more appropriate, and the sole word embeddings were not enough to obtain good results. At least the validation loss was not increasing, but it was not decreasing in a good manner either.

The only thing that we tried and actually gave us a more normal decaying validation loss was to merge the train and test set, re-shuffle and re-split. Nevertheless, we have a hard time justifying this action, as it can be seen as feeding the model during training with information used on validation. We thought of this merging and re-splitting as mixing topics more that testing on train data, but as we didn't completely separate the bodies and headlines then the model ends up seeing the same text during both phases (but never the same pair!). This is because we have no access to the stance of pairs that do not appear on the data, neither to the original topics of each headline/body.

Looking back at our model, we have some new ideas of what could have been different. For instance, the headlines and bodies were processed using the same LSTM. We did this because of the small amount of data, thinking that using all of it with the same LSTM could translate into more training and better performance. Now we wonder if this made the model unable to learn useful representations because of the difference in length between the headlines and bodies. Indeed, many of the articles we read that dealt with the FNC considered some sort of trimming of the bodies. They fixed a maximum number of words to consider. Maybe we could either use two separate LSTM models, or consider this trimming of the bodies to make them more comparable in length with the headlines.

4 Conclusion

During this project we were able to replicate the results of one of the finalist teams in the FNC-1. We tried to improve from there by proposing the use of a LSTM model to embed sentences testing with many different pre-trained word embeddings. This sentence embeddings would be the input of an MLP, replicating the idea behind the followed baseline. This idea that seemed promising to us showed a bad performance. Training of the model was complicated, as validation loss and accuracy followed completely different patterns than those on the train set.

References

[1] Piotr Bojanowski et al. "Enriching word vectors with subword information". In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146.

[2] Andreas Hanselowski et al. *A Retrospective Analysis of the Fake News Challenge Stance Detection Task*. 2018. arXiv: 1806.05180 [cs.IR].

[3] Razan Masood and Ahmet Aker. "The Fake News Challenge: Stance Detection using Traditional Machine Learning Approaches." In: *KMIS*. 2018, pp. 126–133.

[4] Tomas Mikolov et al. "Advances in Pre-Training Distributed Word Representations". In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*. 2018.

[5] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *arXiv preprint arXiv:1310.4546* (2013).

[6] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[7] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[8] Benjamin Riedel et al. *A simple but tough-to-beat baseline for the Fake News Challenge stance detection task*. 2018. arXiv: 1707.03264 [cs.CL].

[9] Valeriya Slovikovskaya. "Transfer learning from transformers to fake news challenge stance detection (FNC-1) task". In: *arXiv preprint arXiv:1910.14353* (2019).

[10] John Wieting et al. "Charagram: Embedding words and sentences via character n-grams". In: *arXiv preprint arXiv:1607.02789* (2016).