Université de Liège

Web and text analysis

INFO2049-1

Sentiment Analysis

Auteures :

de la Brassinne Bonardeaux Ophélie (s)
Navez Lucie (s180703)

Master ingénieur civil

Année académique 2022-2023

1 Git repository

The code for our project is available here

2 Investigation

Sentiment analysis consists in classifying pieces of text into different kinds of categories. Those categories can range from "positive" or "negative", extra classes can also express different degrees/emotions, related to a positive or negative sentiment. Adding more classification categories is called fine-grained sentiment analysis.
Of course, this kind of process can be useful in many areas, e.g. for community management on social medias (automatic deletion of mean comments, for instance).
Traditionally, the most used architectures for sentiment analysis are RNNs, but CNNs are sometimes used too.
The goal of this project us to implement an RNN with attention for sentiment analysis. As inspiration, two approaches were advised.

2.1 Letarte et al. - SANs

Letarte et al.[5] explore the way models interpret relations between words by using their Self-Attention network (SANet), which is according to them a flexible and interpretable model. Interpretability of model results is apparently easier as long as attention mechanism are used, because the attention mechanism allows the model to understand which words are important in a sentence, and on which it should focus its *attention*. Self-attention thus allows each word of the sentence to pay attention to all other words of the sentence and capture the ones that are important in a certain context.
The proposed architecture is described as a text-classification architecture called SANet, that models the interaction between all input word pairs and is depicted on figure 1

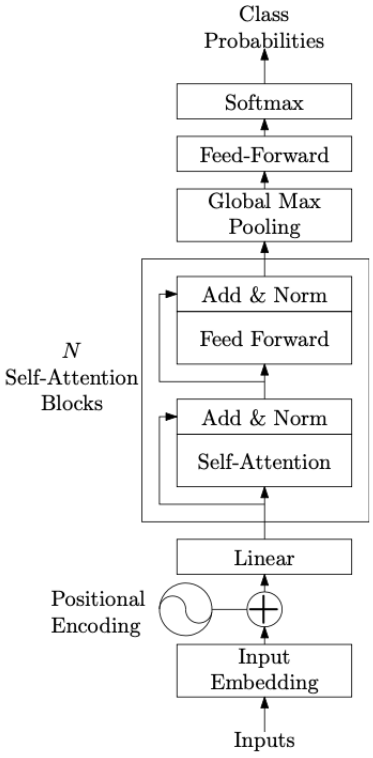

FIGURE 1 – SANet architecture

2.2 Ambartsoumian & Popowich

The approach studied by Ambartsoumian and Popowich [1] proposes to use Self-attention networks (SANs) to do sentiment analysis. Their basic building block is the attention mechanism. Those networks are supposed to show better results, without using any convolutions or recurrences. Such models are called Transformers and can achieve state-of-the-art performances for sentiment analysis among other things.
Attention[4] is popular for its ability to capture context in sequential data.
The following architecture is described in the paper :

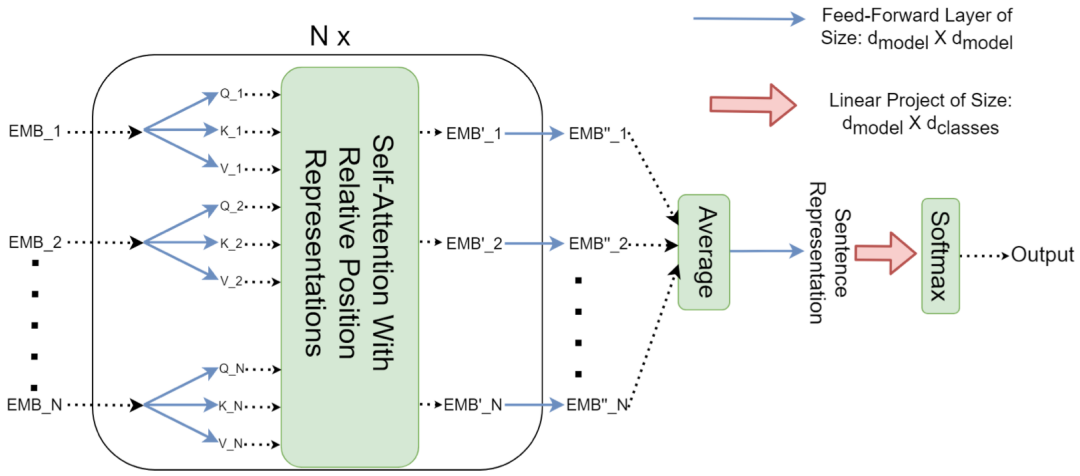

FIGURE 2 – SAN architecture

One of our attempts is based completely on this paper and focuses on using only attention mechanisms, without using any kind of convolutional or recurrent-based blocks. It will be discussed in the following sections.

2.3 Wu et al.

In this paper, Wu et al [8] develop a new method for document embedding. It explores the possibilities of using the word embedding Word2Vec along with Word Mover's Distance (WMD) in order to build the Word Mover's embedding. Word2Vec is a famous word embedding method that we used in our models and we will develop later in this report. WMD is a distance between two text documents that takes into account the alignments between words. In the figure below, we can see the algorithm of this new document embedding.

Algorithm 1 Word Mover's Embedding: An Unsupervised Feature Representation for Documents

Input: Texts $\{x_i\}_{i=1}^N$, $D_{\max}$, R.
Output: Matrix $Z_{N\times R}$, with rows corresponding to text embeddings.

Compute $v_{\max}$ and $v_{\min}$ as the maximum and minimum values, over all coordinates of the word vectors $\boldsymbol{v}$ of $\{x_i\}_{i=1}^N$, from any pre-trained word embeddings (e.g. Word2Vec, GloVe or PSL999).
for $j = 1, \ldots, R$ **do**
  Draw $D_j \sim \text{Uniform}[1, D_{\max}]$.
  Generate a random document ω_j consisting of D_j number of random words drawn as $\omega_{j\ell} \sim \text{Uniform}[v_{\min}, v_{\max}]^d$, $\ell = 1, \ldots, D_j$.
  Compute $\boldsymbol{f}_{x_i}$ and $\boldsymbol{f}_{\omega_j}$ using a popular weighting scheme (e.g. NBOW or TF-IDF).
  Compute the WME feature vector $Z_j = \phi_{\omega_j}(\{x_i\}_{i=1}^N)$ using WMD in Equation (2).
end for
Return $Z(\{x_i\}_{i=1}^N) = \frac{1}{\sqrt{R}}[Z_1\ Z_2\ \ldots\ Z_R]$

FIGURE 3 – Word Mover's embedding algorithm

The performance results of Word Mover's embedding were most of the time better than other document embeddings like doc2vec or SIF with pre-trained GloVe.

3 Method

3.1 Libraries

Apart from the basic libraries, we mainly used :

- **NLKT :** for the data pre-processing (removing stopwords, lemmatisation, ...)
- **gensim :** for the use of the word embedding word2vec
- **PyTorch :** for the implementation of the model

3.2 Datasets

At first, we used the Twitter sentiment analysis dataset available on the following link. This dataset contains more than 1.6 million tweets. We decided to split it in the following way :

- The training set : 1.280.000 tweets
- The validation set : 320.000 tweets
- The testing set : 498 tweets

It is indeed an interesting dataset to work with because of the challenge brought by the diversity in the tweets. Some of them are not in English, people are mentioning other Twitter users using handles @, there are URLs in some of the tweets and some users also misspell words, or use Internet slang...

However, due to very the high number of tweets in the train and validation set (which extends the training time of our model and doesn't necessarily give better performance), we decided to reduce these sets by only keeping 20% of their content (chosen at random), which still yields a dataset of around 320000 tweets.

Unfortunately, this first dataset proved to give poor performances due to the extensive use of words that don't exist in the embeddings by Tweeter users (ex : "looovvve", misspellings, ...). Because of this challenge, and after many attempts to overcome it, we decided to use the IMDB dataset instead, (to compare different approaches also), available here. Since this dataset is made of movie reviews and is globally written in a better English than the tweets, it is more unlikely to contain unknown words by the embedding and thus, would potentially give better performances. We split it in the following way :

- The training set : 36.000 reviews
- The validation set : 9.000 review
- The testing set : 5.000 reviews

3.2.1 Pre-processing

All the specific details are available in the notebook in the pre-processing section. For the pre-processing of the data, we performed the following steps :

(1) Put everything to lowercase

(2) Remove unwanted characters/words like HTML beacons, emojis, handles, ...

(3) Remove stop-words (words that do not bring any information to the semantic meaning of a sentence)

(4) Lemmatisation : reducing each word to its root : "lighter" and "lightning" both reduce to "light"

(5) Stemming : This functionality is implemented, but we do not use it in our final version, because it seems less safe than lemmatisation. It is kind of the same principle, but stemming does not take the context of the word into account, and thus yields less precise results.

(6) Remove non-English words

(7) Expand English contractions (he'll, can't, ...)

(8) Remove extensive repetitions of words (for instance, the Tweeter dataset contained entries as follows : "fuck, fuck, fuck, fuck, fuck, fuck, ..." or "shit, shit, shit, shit, shit, ..." and these were the longest tweets of the dataset at some point...

Also, note that the order of these step can be important, for instance, removing punctuation before tokenization can hinder the expansion of English contractions such as "he'll" as "he will", or "I'd like" to "I would like". All these steps are performed sequentially, among other less important ones, during the loading of the dataset. The goal of all this pre-processing is to eliminate all unuseful information for training for the model so that the data are not noisy, and that the model can learn by focusing on the parts that are really important for sentiment analysis.

3.2.2 Padding and encoding

The model can only be fed constant size inputs. That's when a problem arises : our inputs are non-constant length lists of tokens. Some sentences contain only a few tokens, and some contain more than hundreds in the case of the IMDB dataset.
To overcome this problem, we need to expand sentences that are too short (by adding a number of padding `<PAD>` tokens), and truncate those that are too big (by removing some words).
We chose as the input size of our model the value `sequence_length`, to which all inputs will be modified. The values we chose for both datasets can be seen in table 1.
These values were chosen based on some statistics on the length of the sentences in both dataset. To finish off the constituting of our dataset, we also insert one `<SOS>` as the first token of each sentence, and one `<EOS>` token as the final token (after the padding tokens if any). Then, the token sequence is converted into its encoding.

3.3 Vocabulary

In order to build the embedding matrix (discussed later), and to have an idea of the content of our datasets, the step of building a dictionary is crucial.
It simply consists in building a set of all the distinct words used in the training set and associate to them an ID.
Also, for the sake of learning, we manually add to the vocabulary some custom tokens that will help structure the dataset :

- `<PAD>` : to fill in sequences that are shorter than a fixed `sequence_length`. Discussed later.
- `<SOS>` : to mark the Start Of a Sentence fed to the model
- `<EOS>` : to mark the End Of a Sequence fed to the model (`<EOS>` comes before `<PAD>` tokens).
- `<UKN>` : to indicate that the word is unknown to the embedding. Discussed later.

Thus, for the pre-processing described here above, the obtained vocabulary sized for both datasets are written down in table 1. During the process of building the vocabulary, we can also get an idea of how long the entries are. The maximum sizes of sentences for both datasets can also be viewed in table 1. Note that in theory, the Tweeter dataset should contain way more different words than the IMDB one. Indeed,

	Tweeter sentiment analysis	IMDB
Vocabulary size	30982	30619
Max sentence length	20	692
Sequence length	12	100

TABLE 1 – Vocabulary statistics

the vocabulary employed by Tweeter user is vaster because it includes misspellings, Twitter slang, words that are not english, and so on. And indeed, before pre-processing, it is the case. But the pre-processing has a bigger impact on the Tweeter dataset than on the IMDB one, and that is why in the end we manage to obtain vocabularies whose sizes are similar for both datasets. In contrast, note that sentences of the Tweeter dataset are way shorter than the reviews of the IMDB dataset, which again, makes sense.

3.4 Embeddings

The embedding allows to transform word tokens into vectors that will be easily interpretable by the model. This first step is an object of investigation, because there are many ways in which embedding can be achieved. For this project, we used the following embedding techniques :

- GloVe
- Word2Vec
- FastText
- Doc2Vec
- Word2Vec averaged with TF-idf weights

The code is modular and allows to choose one of these embeddings easily. In our implementation, we also chose to go with the 300d embedding size of all of these embeddings.

3.4.1 GloVe

GloVe[7] uses some co-occurrence statistics (more specifically, GloVe learns the ratios of raw occurrence probabilities between pairs of words) in order to deduce correlations between words. The context of the word is important and influences the vector representation of the word.
Thus, the model captures the context of a word by using the frequency of words which appear often with the word.

3.4.2 Word2Vec

Word2Vec[6] uses co-occurrence statistics, more precisely, it tries to see which words are often employed together. It does so by using a window, whose size can be set, that surrounds each word of the sentence. The "central" word is called the *target*, and the words surrounding it are called the *context words*. A dataset is constituted with all of the target-context words pairs. Afterwards, Word2Vec is trained on these pairs, and becomes thus able to establish links between some words and the context in which they are employed.

3.4.3 FastText

FastText[3] embeds words in a (supposedly) quicker and faster manner. One of its strength compared to the previous methods is that it works very well with less-frequent words, because it is able to create vector representations for these unknown words.

3.4.4 Doc2Vec

Doc2Vec is a method extremely similar to Word2Vec. It used CBOW method, to which a new matrix, called the paragraph matrix, was added.

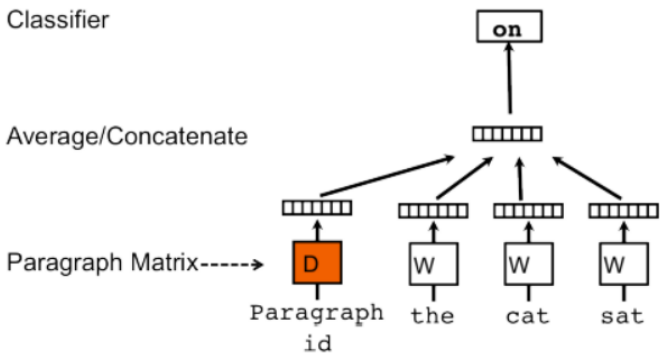

FIGURE 4 – Doc2Vec method (image from this link)

3.4.5 Word2Vec averaged with TF-idf weights

Finally, we decided to build our own document embedding based on the Word2Vec embedding and the TF-idf value. This was done in the following way :

(1) Load a pre-trained Word2Vev embedding

(2) Compute the Tf-idf matrix for the given set

(3) For each sentence in the set, we take the vectors of each word in that sentence and we average them, using as weights the normalized Tf-idf values

3.5 Model architectures

As explained above, we aimed to build a very modular code, and thus, our implementation allows to use three kind of models :

- RNN
- Attention only model
- RNN + Attention mechanism

The following architectures are described in the sections below. These architectures were implemented successively, in the following way : first we tried to implement the first fundamental block : the RNN. Afterwards, to explore the way attention works, we built a model using only attention, implemented based on the Ambartsoumian & Popowich paper[2].
Finally, the next step for us was to integrate an attention mechanism in our previous RNN architecture.

Note that we had to adapt the models for the document embedding. Indeed, instead of using an embedding layer, we directly gave as an input the already embedded vectors. The models will be the same except that the embedding layer will be removed.

3.5.1 RNN

The RNN was the most straightforward part of our implementation. RNNs are indeed one of the most popular ways to do sentiment analysis, and thus there are plenty of information on that subject.
Our RNN model comprises the following layers :

- Embedding layer (GloVE, Word2Vec or FastText)
- LSTM or GRU
- Fully connected layer

Parameters
The parameters of our RNN are the following :

- Embedding matrix : obtained from previous steps
- Hidden dimension : this parameter decides of the size of the hidden vectors used in the LSTMs or GRUs.
- Dropout : we added dropout to the embeddings and before the fully connected layer.

- Bidirectional : The LSTMs and GRUs can be made bi-directional. This allows to capture more context between words by taking into account not only the words appearing before the current word, but also the words appearing after, that also play a role to capture the context of the word in the sentence.
- Padding ID : this parameter allows to inform the embedding layer of the presence of the padding `<PAD>` token so that it avoids "learning" the padding tokens.
- Number of layers : this parameter defines the number of stacked GRUs or LSTMs.
- The parameter type has been added so that the user can choose between using LSTMs or GRUs.

3.5.2 Attention only

As explained above, the attention only model is completely based on the Ambartsoumian & Popowich paper[2], and our implementation of the architecture follows what is denoted on the figure 2.

3.5.3 RNN+Attention

This architecture, that we called *AttentiveRNN* is, as its name indicates it, an RNN with a layer of attention. It is structured as follows :

- Embedding layer (GloVE, Word2Vec or FastText)
- LSTM or GRU
- Self-attention layer (here we chose to implement it manually in our final version for our own information and to understand better the underlying principle of attention, but we also tried it using the `nn.MultiheadAttention` from PyTorch)
- Fully connected layer

A dropout is applied to the embeddings of the input vectors. The result is fed to either GRUs or LSTMS, and self-attention is then applied to the result. Self-attention is implemented as a separate class and applies to its input the same operations that are described in this paper[2]. Finally, the results of attention are averaged and fed to a fully connected layer and a linear projection. A Softmax is then applied so that the output is a vector of 2 elements corresponding to the probabilities for the tweet (or IMDB review) to belong to the positive or negative sentiment class.

Parameters
Obviously, all parameters are shared with the RNN architecture, since this new architecture is actually a combination of both RNN and attention mechanism. The attention mechanism does not require any additional parameter.

4 Results

4.1 Basic implementation

4.2 LSTM vs GRU

For the analysis of the performances of LSTM and GRU, we decided to use the following parametrization :

- Embedding layer : pre-trained Word2Vec
- Embedding dimension : 300
- Hidden layer dimension : 32
- Dropout : 20%
- Bidirectional : true
- Fully connected layer
- Dataset : IMBD

4.2.1 LSTM

The following metrics were computed on the test set.

- Accuracy : 0.8654

- Recall : 0.8836
- Precision : 0.8439

Confusion matrix :

	Classified positive	Classified negative
Actual Positive	2126	280
Actual Negative	393	2201

TABLE 2 – Confusion matrix LSTM

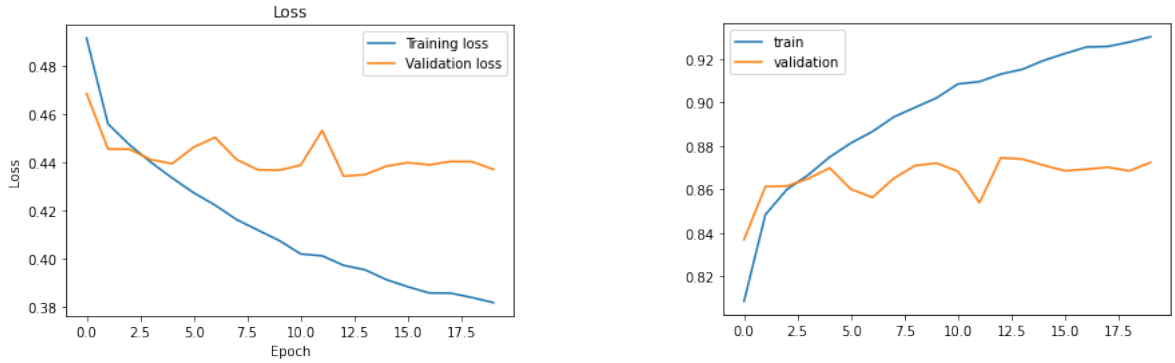

FIGURE 5 – Evolution of the loss and the accuracies during training

4.2.2 GRU

The following metrics were computed on the test set.

- Accuracy : 0.8598
- Recall : 0.88
- Precision : 0.8356

Confusion matrix :

	Classified positive	Classified negative
Actual Positive	2105	287
Actual Negative	414	2194

TABLE 3 – Confusion matrix GRU

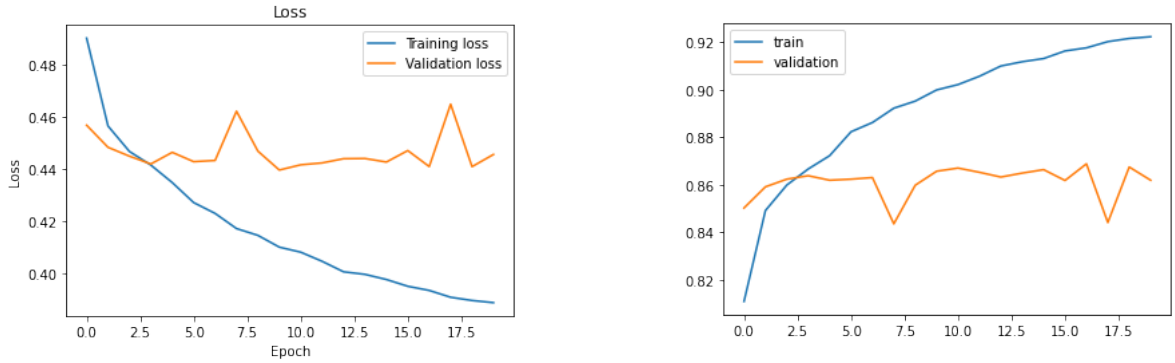

FIGURE 6 – Evolution of the loss and the accuracies during training

4.2.3 Comparison

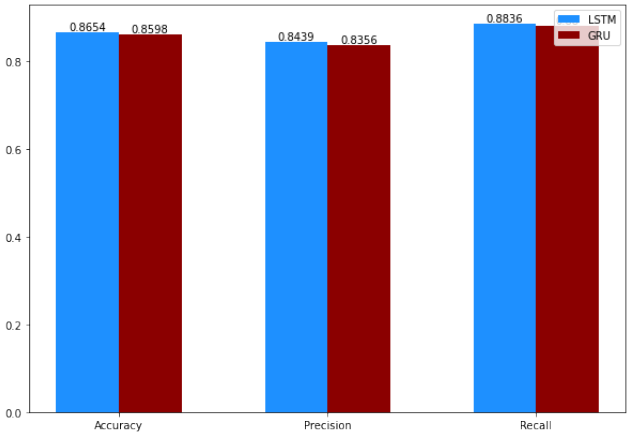

FIGURE 7 – LSTM vs GRU

When we have a look at the results, we can observe that the performance of LSTM and GRU are quite similar. However, if we look closely, it seems that LSTM has a slightly better performance. If we look at the figures 5 and 6, we can first observe for example that the loss on the validation set for LSTM is slightly lower than 0.44 while it is higher for GRU. We can also see that the accuracy is higher than 0.86 on the validation set for LSTM while it is a bit under that value for GRU.

4.3 Influence of word embeddings

For the analysis of the performances of different kind of embeddings, we decided to use the following parametrization :

- Training layer : we chose to use LSTMs as it demonstrated slightly better performance as discussed above.
- Embedding dimension : 300
- Hidden layer dimension : 32
- Dropout : 20%
- Bidirectional : true
- Fully connected layer
- Dataset : IMBD

4.3.1 Word2Vec

For Word2Vec, we made 2 different version : one with a pre-trained dataset and one that we trained on our training data (train set + validation set).

Our curves indicated an overfitting on the learning set. We could observe the typical scheme of the validation loss going back at the same rate as the training loss, but then increasing again, forming an U shape, which means that our model overfits the training data.
To avoid such problem, we added an early stopping mechanism to our implementation to allow the model to stop before it overfits the training data, thus conserving the best weights for the best accuracy on the validation set. The early stopping mechanism checks at each epoch if the validation loss keeps getting smaller and smaller. If at some point it does not decrease anymore, we wait for a few other epochs (this number is set through the `patience` parameter), and if it still does not decrease, then, we stop the training at that point and save the model as such, which allows to avoid overfitting by stopping the training early. The obtained curves with early stopping show indeed that overfitting does not happen anymore :

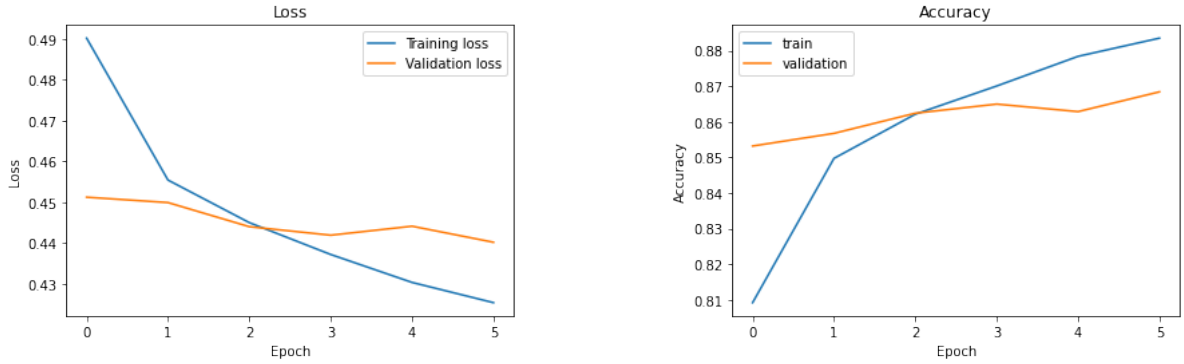

FIGURE 8 – Evolution of the loss and the accuracies during training

- Accuracy : 0.854
- Recall : 0.8884
- Precision : 0.8122

Confusion matrix :

	Classified positive	Classified negative
Actual Positive	2046	257
Actual Negative	473	2224

TABLE 4 – Confusion matrix pre-trained Word2Vec

We can thus already observe that using an Word2Vec embedding increases the performance of the model for every metrics. For the second version, using a Word2Vec embedding trained on our data, we found the following ;

- Accuracy : 0.8674
- Recall : 0.829545
- Precision : 0.9273

Confusion matrix :

	Classified positive	Classified negative
Actual Positive	2336	480
Actual Negative	183	2001

TABLE 5 – Confusion matrix Word2Vec

This time, to avoid the overfitting problem previously encountered, we used early stopping straightaway, and obtained the following learning curves :

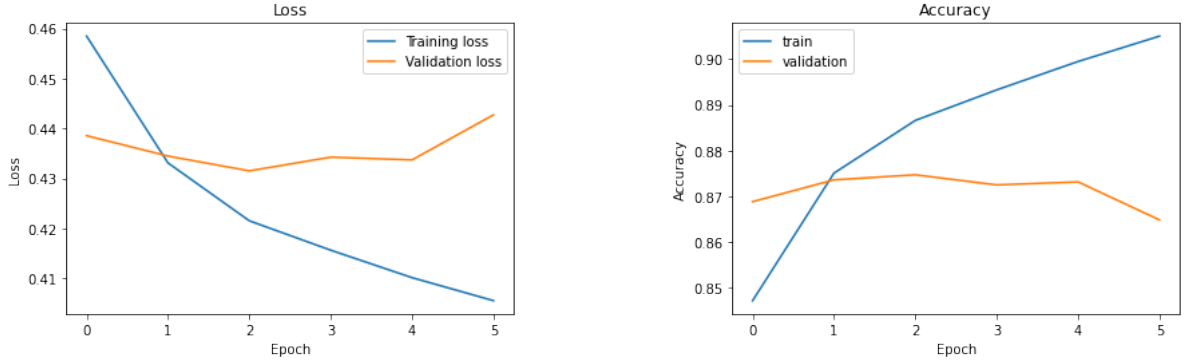

FIGURE 9 – Evolution of the loss and the accuracies during training

As expected, the results are even better when training the embedding directly on our vocabulary.

4.3.2 GloVe

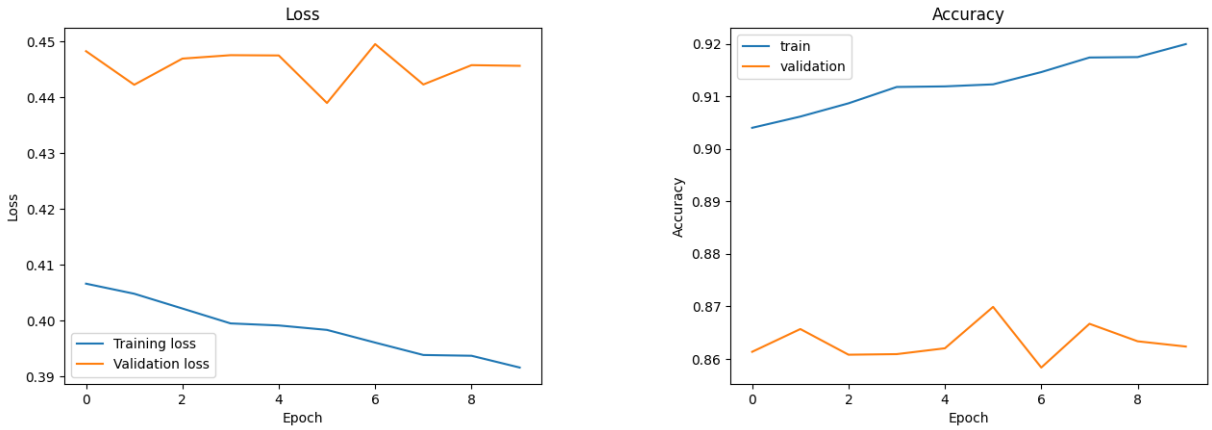

FIGURE 10 – Evolution of the loss and the accuracies during training with Word2Vec averaged with TF-idf weights

- Accuracy : 0.8588
- Recall : 0.880722385552289
- Precision : 0.832473

Confusion matrix :

	Classified positive	Classified negative
Actual Positive	2097	284
Actual Negative	422	2197

TABLE 6 – Confusion matrix without attention

The results obtained with GloVe are comparable to those previously obtained,

4.3.3 FastText

- Accuracy : 0.8616
- Precision : 0.8782
- Recall : 0.842

Confusion matrix :

	Classified positive	Classified negative
Actual Positive	2121	294
Actual Negative	398	2187

TABLE 7 – Confusion matrix FastText

And the following learning curves :

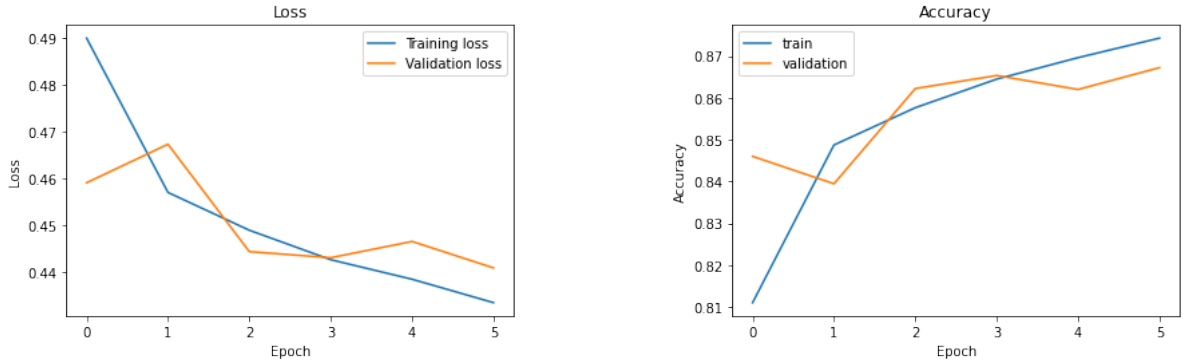

FIGURE 11 – Evolution of the loss and the accuracies during training

Once again, we obtain similar results than with the others. In is however interesting to note that FastText never gave the best result, we can thus suggest that it might be a bit less appropriate than the other ones.

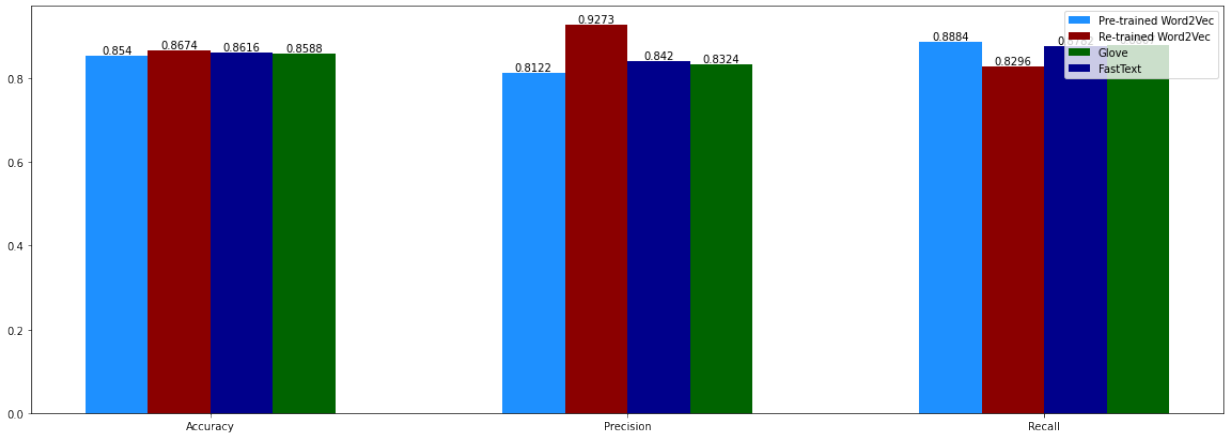

FIGURE 12 – Comparison word embeddings

4.4 Influence of document level embeddings

For the analysis of the performances of different kind of document embeddings, we decided to use the following parametrization :

- Training layer : we chose to use LSTMs as it demonstrated slightly better performance as discussed above.
- Embedding dimension : 300
- Hidden layer dimension : 32
- Dropout : 20%
- Bidirectional : true
- Fully connected layer
- Dataset : IMBD

We decided to not use the attention for this analyze. The main reason was that the interest of attention is to get more context. However, document embedding returns a vector based on the context of the sentence. We thus though that there was no use for attention. Furthermore, we trained the model with Doc2Vec with and without attention, and we got a better loss and accuracy without attention (see figures 13 and 14).

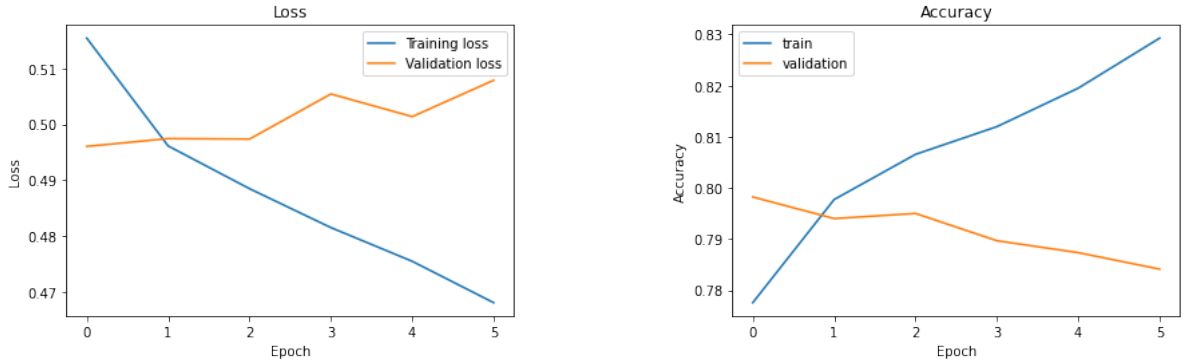

FIGURE 13 – Evolution of the loss and the accuracies during training with Doc2Vec with attention

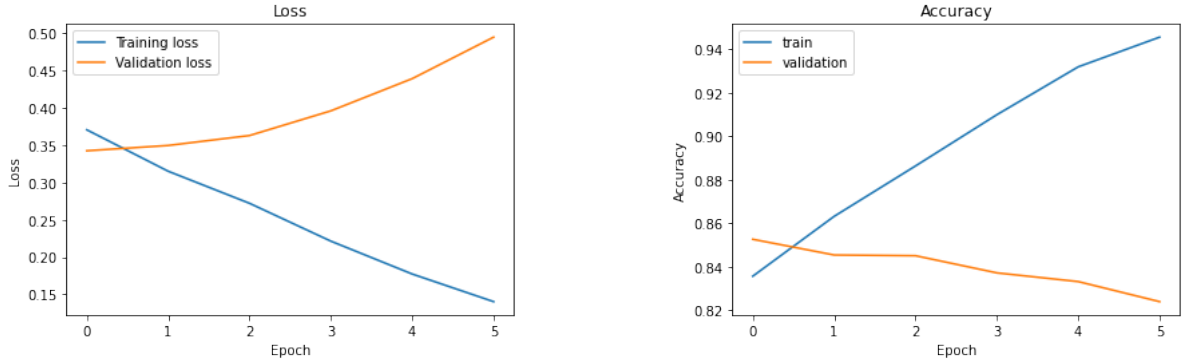

FIGURE 14 – Evolution of the loss and the accuracies during training with Doc2Vec without attention

4.4.1 Doc2vec

- Accuracy : 0.8236
- Recall : 0.8299
- Precision : 0.8174

Confusion matrix :

	Classified positive	Classified negative
Actual Positive	2059	422
Actual Negative	460	2059

TABLE 8 – Confusion matrix doc2vec

4.4.2 Word2Vec avaraged with TF-idf

- Accuracy : 0.8278
- Recall : 0.8513
- Precision : 0.7975

Confusion matrix :

	Classified positive	Classified negative
Actual Positive	2009	351
Actual Negative	510	2130

TABLE 9 – Confusion matrix word2vec averaged with TF-idf

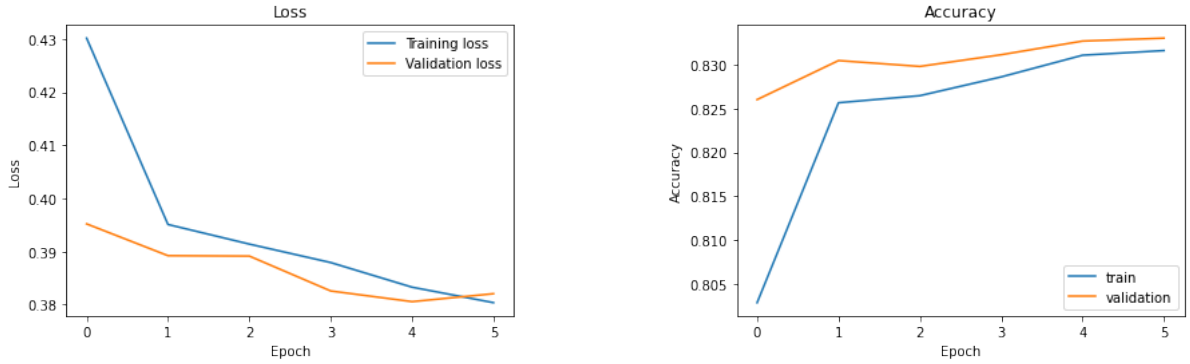

FIGURE 15 – Evolution of the loss and the accuracies during training with Word2Vec averaged with TF-idf weights

We can see that the performance scores are extremely between the 2 document embeddings. We can also observe that that these results are slightly lower than the ones observed with word embedding.

We can assume that the attention that was used with word embedding is partly responsible for these performances. Indeed, the interest of using document embedding is to make full use of the context. However, thanks to the attention layer, the RNN is able to make use of the context too (even though it is in a different way).

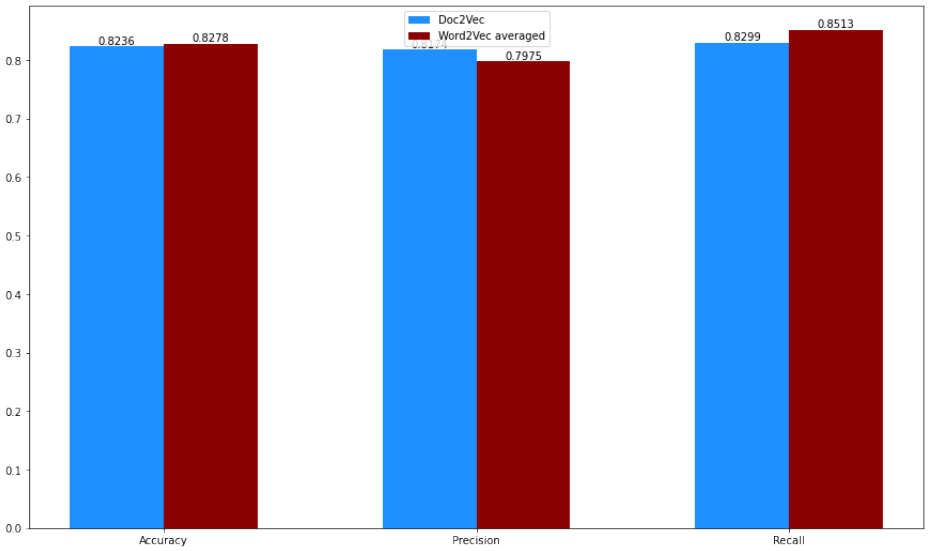

FIGURE 16 – Comparison document embeddings

4.5 Influence of attention

To study the influence of attention, we used the same parametrization as earlier, and decided to use the pre-trained GloVe embedding since it gave very nice performances with the RNN earlier. The following learning curves were obtained :

- Accuracy : 0.8578
- Recall : 0.8689795918367347
- Precision : 0.8451766574

Confusion matrix :

	Classified positive	Classified negative
Actual Positive	2129	321
Actual Negative	390	2160

TABLE 10 – Confusion matrix Glove

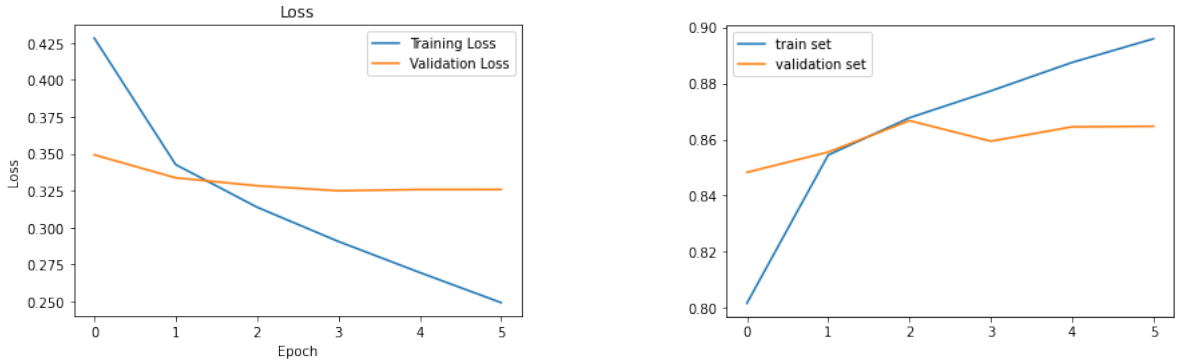

FIGURE 17 – Evolution of the loss and the accuracies during training

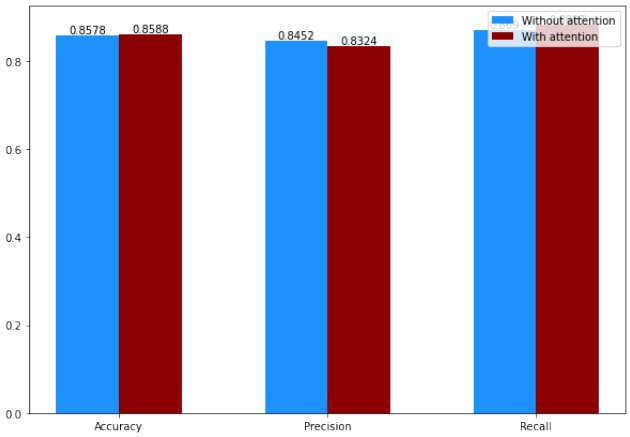

FIGURE 18 – Comparison attention

We observe that apart from the precision, the attention provides better results than without attention.

5 Conclusion

To conclude, even if the results could possibly be improved, we tried to implement as much points of comparisons as possible, by writing new architecture to test simple RNNs, RNNs with attention, and even more of a "Transformer" architecture, using only attention for our own information and whose performances are not depicted in this report. However, this architecture is available in our implementation files. Since the first results we obtained with the first dataset (Twitter sentiment analysis) were both not concluding because of the noise present in this kind of dataset, we even chose to shoot our shot with another dataset.

The embeddings also play an important role in the results obtained, and thus we tried many of them, with their pre-trained (or not) version. It seems like the kind of embedding does not make a huge difference, they all have their strengths and weaknesses, but using them allows to increase the performance of the model, except for FastText, which seems to be less adapted to this task and gives generally less good performances.

Adding an attention layer seems to be a good idea in general, since it allows to capture a bit more context in the sentences, and allows to increase the performance a bit more. It was an interesting project, allowing us to dive more into the practical aspect of sentiment analysis and text classification in general.

NOTE : Please note that some useful code snippets (the training loop, some utility functions...) were directly retrieved and slightly modified from a former project led by Victor Mangeleer, Axelle Schyns, and Lucie Navez (co-author of this report) for the course INFO8010-1 Deep learning, tought by LOUPPE Gilles, with their consent.

Références

[1] Artaches Ambartsoumian and Fred Popowich. Self-attention : A better building block for sentiment analysis neural network classifiers. arXiv preprint arXiv :1812.07860, 2018.

[2] Artaches Ambartsoumian and Fred Popowich. Self-attention : A better building block for sentiment analysis neural network classifiers. In Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, pages 130–139, Brussels, Belgium, October 2018. Association for Computational Linguistics.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. Transactions of the association for computational linguistics, 5 :135–146, 2017.

[4] Jan Chorowski, Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. End-to-end continuous speech recognition using attention-based recurrent nn : First results. arXiv preprint arXiv :1412.1602, 2014.

[5] Gaël Letarte, Frédérik Paradis, Philippe Giguère, and François Laviolette. Importance of self-attention for sentiment analysis. In Proceedings of the 2018 EMNLP Workshop BlackboxNLP : Analyzing and Interpreting Neural Networks for NLP, pages 267–275, 2018.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. 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), pages 1532–1543, 2014.

[8] Lingfei Wu, Ian En-Hsu Yen, Kun Xu, Fangli Xu, Avinash Balakrishnan, Pin-Yu Chen, Pradeep Ravikumar, and Michael J. Witbrock. Word mover's embedding : From Word2Vec to document embedding. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 4524–4534, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.