Exploring Attention mechanisms for LSTM Networks applied to sentiment analysis

Benjamin GALMICHE & Matthis MANTHE & Yann POURCENOUX

May 17, 2020

Abstract

After building a basic LSTM network to check that *Xavier* initialization and using *GloVe* embedding was improving the performance, it has been verified that using an attention block and using aspect embedding was improving the performance for classifying the sentiment of a sentence given an aspect. Using another pre-trained embedding weights such as *FastText* is not improving the performance however using a combination of *GloVe* for the common words and *FastText* for the rare words could be a way of improvement.

Introduction

Among Natural Language Processing (NLP) tasks, one of the most known and studied these years is Sentiment Analysis. The goal is to extract a sentiment (positive, neutral, or negative) from a sentence, or a review. A lot of scientific work has already been led on this field, and some even succeeded to build Aspect-level sentiment classification (Wang et al. [1]). This is a finer work, where some particulars "aspects" are classified in a sentence. Indeed, it can be hard to classify a sentence as 'positive' or 'negative' when it contains two contradictory aspects such as in the sentence : "The food is really good, but it is too expensive.". In this example, the aspects "food" and "price" can be classified respectively as 'positive' and 'negative' but it would be hard to classify the whole sentence.

In this project, we try to reach state-of-the-art results from our baseline paper (Wang et al. [1]), comparing a simple Long Short Term Memory (LSTM) network and different versions of an Attention-based LSTM, supposed to achieve this task of aspect-level classification. The next step is to improve the obtained network with regularization techniques based on another work (Bao et al. [2]). Finally, we also explore different ways for words embeddings, in order to find the best network.

1 The Dataset

Before diving into the network and all its technical and interesting characteristics, let's have a look at the data to get a better understanding of the task.

1.1 Architecture

The data is contained into xml files such as the example 1 below. Each data has a text attribute, which is nothing else that the sentence of the review. There are five aspect categories: anecdotes, ambience, food, price, service. Each data has one or more corresponding aspect category with an assigned polarity.

Listing 1: data example

```
<sentence id="2846">
    <text>Not only was the food outstanding, but the little
                                ’perks’ were great.</text>
    <aspectCategories>
        <aspectCategory category="food" polarity="positive"/>
        <aspectCategory category="service" polarity="positive"/>
    </aspectCategories>
</sentence>
```

1.2 Isolating the relevant informations

This previously stated polarity is what will be used as label. In the first part of the project ie section 2, only basic and low level improvement have been added to the network. It can not have a perception of the aspect so the polarities of the aspect categories have been averaged for each sentence to get labels the network can work with. As said previously, there are five different categories so has been chosen to label a sentence as *negative* if more aspects for that sentence were negative than positive and vice-versa for the *positive* class. If there are as much *negative* and *positive* labeled category for one sentence (or only *neutral* ones), the sentence is classified as *neutral*.

For the second part of the project ie section 3 and onwards, each sentence-aspect pair will be an input and the polarity for the given aspect of this sentence will be the label.

To sum up, in the first part, the example above will result in one input sentence labeled as positive. For the second part it will result in two sentences-category pair, one with the sentence and the aspect *food* labeled as *positive* and one with the sentence and the aspect *service* also labeled as *positive*.

2 Base LSTM as reference mark

Since we had no prior experience with LSTM networks, it has been chosen to create a basic LSTM network which will be used as a reference mark for the

incoming improved models. In this section, it will be explained what is a LSTM, how this basic LSTM was constructed and how the training process has been thought and improved.

2.1 LSTM-network architecture

Firstly, let's take a look at a sentence. One sentence from the data and will be used as example is *Their wine list is excellent.*. Let's assume that this sentence uses all the words we have in all our data to ease the explanations. A vocabulary is created assigning to each word an *index*. Since computations are done in batches, it is important that all the sentences in one batch have the same size and this is why an additional character is used and is called padding.

The vocabulary here would then be {<pad>: 0, 'Their': 1, 'wine': 2, 'list': 3, 'is': 4, 'excellent': 5, '.': 6}. Let assume that the longest sentence in the batch has one more word than our example. One padding character will then be used. This vocabulary is then used to convert those words into indexes (cf bottom row on figure 1).

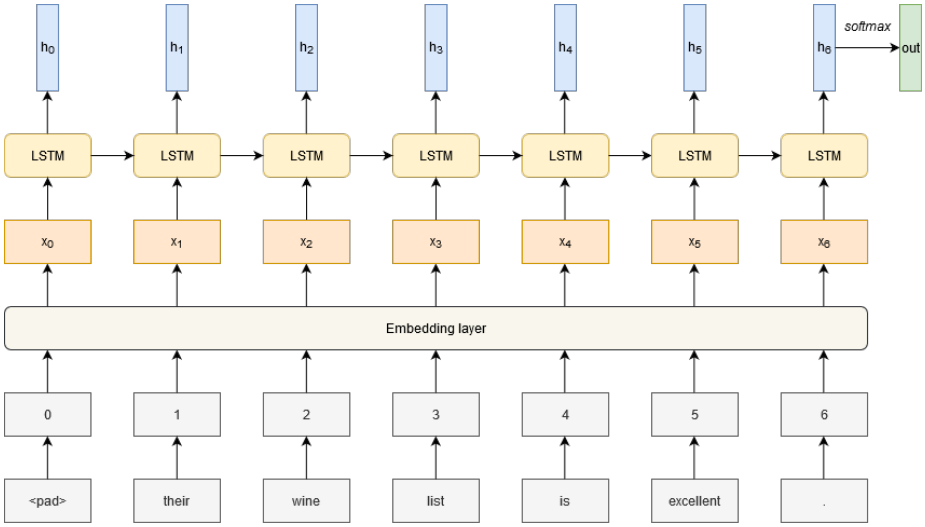

Figure 1: Architecture of a basic LSTM classifier.

The embedding layer is a trainable layer that assigns a representation in a chosen d dimension of the words. It can be seen as a matrix where the first row of the matrix corresponds to the embedding of the word of index 0 which would correspond to the embedding of the padding character.

This embedded sentence composed of embedded words is then feed in a LSTM network as shown on figure 1. Each cell can be computed as described in

the paper [1].

$$X = \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} \tag{1}$$

$$f_t = \sigma(W_f \cdot X + b_f) \tag{2}$$

$$i_t = \sigma(W_i \cdot X + b_i) \tag{3}$$

$$o_t = \sigma(W_o \cdot X + b_o) \tag{4}$$

$$c_t = f_t \odot c_t - 1 + i_t \odot tanh(W_c \cdot X + b_c) \tag{5}$$

$$h_t = o_t \odot tanh(c_t) \tag{6}$$

Since the dimensions of the hidden vectors (the h_i on figure 1) are also of dimension d, $W_f, W_i, W_o \in \mathbb{R}^{d \times 2d}$ and $b_f, b_i, b_o \in \mathbb{R}^d$. σ stands for the sigmoid function and $\odot$ for the element-wise multiplication. In our project this d-dimension is equals to 300. The last hidden vector (h_6 in our example) is then seen as the representation of the whole sentence. This is also why we chose to add padding on the left and not on the right of the sentence.

This final hidden vector goes then through a linear layer with softmax activation function as follows (seen as the green block on figure 1).

$$out = softmax(W_l \cdot X + b_l) \tag{7}$$

Since the idea is to classify the sentences as *negative/neutral/positive*, $W_l \in \mathbb{R}^3 \times d$ and $b_l \in \mathbb{R}^3$. The softmax functions transforms the output in a probabilistic distributions over the three classes.

2.2 Global training features

To monitor the training of the model, the training data has been split into a train set and a validation set. The validation set represents 30% of the global data. It has been chosen to make the validation set balanced between the classes. With the same way of thinking the yielded training batches to the network are balanced.

Since the accuracy does not represent how a classification model performs on the data. It has been chosen to add an extra metric: the *f1_score*.

The *f1_score* is the harmonic mean of precision and recall.

$$f1\_score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{8}$$

Using these parameters the model achieves a mean accuracy on the test of 0.686 and a mean f1-score of 0.687. Those results are the mean over 20 training/testing cycles.

2.3 Adding Initializer

A good way of improving the final accuracy and stabilizing the learning is to add an initialization that takes into account the dimensions of input and ouptut of a weight. To do so, we used the *Glorot Normal* initialization also known as *Xavier* initialization derived from the publication from Glorot and Bengio [3]. For a weight $W \in \mathbb{R}^{d_{out} \times d_{in}}$ it will be initialized with values drawn from a normal distribution $\mathcal{N}(0, \sigma^2)$ with $\sigma = \sqrt{\frac{2}{d_{in}+d_{out}}}$.

Adding this initializer resulted in having a mean accuracy of 0.689 and a mean f1-score of 0.688.

2.4 Using GloVe pre-trained weights

A common way of boosting the performance of a NLP related model is to use pre-trained weights for the embedding layer (as shown on the figure 1). One of the most trending and used in both papers from Wang et al. [1] and from Bao et al. [2] is *GloVe* embedding from the paper from Pennington et al. [4].

The idea of *GloVe* is to embed the words in a way that the dot product of two embedded vectors equals the log of the number of times the two words occur near each other in the training corpus. We used the pre-trained version on 840B tokens with a vocabulary size of 2.2M and encoded with a dimension d=300.

Our model improved its accuracy on average on the test set up to 0.727 and 0.731 for the f1-score.

2.5 Sum up

Adding *Xavier* initialization barely improved the accuracy but made the learning more consistent. Adding the pre-trained *GloVe* word embeddings improved both the accuracy and the f1-score by 4%.

Table 1: Base-LSTM configurations

Configuration	mean accuracy	mean f1-score
Base	0.686	0.687
Xavier init.	0.689	0.688
Xavier init. & GloVe	0.727	0.731

We've seen that *Xavier* initialization and *GloVe* improve the performance of the network. We will then construct the models to classify a sentence given an aspect from this best performing LSTM.

3 AE-LSTM

The basic LSTM implemented can achieve good classification results on sentiment analysis, however it is not suitable for aspect-level classification. Indeed, if the

goal is to classify a specific aspect in a sentence, the simple LSTM cannot detect the important parts to emphasize in the sentence. This is why an improved version is implemented : an LSTM with Aspect-Embedding (AE-LSTM). Here, an embedding vector is learnt for each aspect category (*food, price...*).

As explained in the Dataset section, the inputs of this network are slightly different. Indeed, the aspects have to be included. Each sentence gives now one input for each aspect it contains. These inputs contain the sentence and the considered aspect, and its corresponding polarity is the label.

In this network, for each aspect category i, there is an embedding vector v_{a_i}. In the LSTM, v_{a_i} is appended to each word input vector. Then, as before, the whole embedded sequence composed of embedded words and aspect is feed in a LSTM.

The results from this AE-LSTM are a mean accuracy of 0.548 and a mean f1-score 0.539.

4 AT-LSTM

The AE-LSTM structure adds the knowledge of the considered aspect as an input of the network, which enables a regular LSTM to analyse the sentiments of a sentence focusing on a particular aspect. But usually, one aspect of a sentence is not represented in the whole sentence, only a small part of it is actually about the considered aspect. This gives us the intuition to build a system such that some parts of the sentence have more importance than others, and this is exactly the description of an attention mechanism built in an AT-LSTM.

Using the same type of dataset as in an AE-LSTM, the structure of an AT-LSTM starts with a regular LSTM. We write $H = [h_1, ..., h_N]$ the matrix of N hidden vectors, like in part 2.1, with N the size of the input sentence. We write $v_a \in \mathbb{R}^{d_a}$ the embedding of the input aspect, with d_a the dimension of the embedding. We compute an attention vector α as follows.

$$M = tanh(\begin{bmatrix} W_h H \\ W_v v_a \otimes e_N \end{bmatrix}) \tag{9}$$

$$\alpha = softmax(w^T M) \tag{10}$$

$$r = H\alpha^T \tag{11}$$

where $M \in \mathbb{R}^{(d+d_a)\times N}, \alpha \in \mathbb{R}^N, r \in \mathbb{R}^d, W_h \in \mathbb{R}^{d\times d}, W_v \in \mathbb{R}^{d_a \times d_a}$ and $w \in \mathbb{R}^{d+d_a}$. Thus, r is a weighted representation of the input sentence. The operation $\otimes e_N$ means that we concatenate the vector N times. A final representation of the sentence is computed as follows.

$$h* = tanh(W_p r + W_x h_N) \tag{12}$$

where $h* \in \mathbb{R}^d$ and $W_p, W_x \in \mathbb{R}^{d\times d}$. This representation is then used as input of a fully connected linear-softmax layer like in the regular LSTM version in 2.1.

The results from this AT-LSTM are a mean accuracy of 0.652 and a mean f1-score 0.650 which is an increased of performance of 10% on the accuracy and and 11% for the f1-score from the AE-LSTM network.

5 ATAE-LSTM

By adding the aspects embedding to the input to the AT-LSTM like we did for the AE-LSTM we get the ATAE-LSTM from Wang et al. [1]. This network is expected to give the best results.

The results from this ATAE-LSTM are a mean accuracy of 0.656 and a mean f1-score 0.653 which is an increased of performance of 0.4% on the accuracy and and 0.3% for the f1-score from the AT-LSTM. This improvement is barely noticeable but is coherent regarding the result from the paper from Wang et al. [1].

6 Attention Regularization

Adding more weights and parameters to a model would also usually imply more chances of overfitting. Some experiments of regularization of the attention mechanisms described above have been developed in [2], we implement them. Since the goal of the attention mechanism is to focus the computation of the network on a small part of the sentence, it could be interesting to limit its capacity to do so, such that instead of having one word with almost the whole attention, the regularization tends to push the attention towards multiple words. To do so, the authors proposed 2 different regularizers. To the cross entropy loss with L2-regularization we add a third term $\epsilon * R(\alpha)$, with ϵ the regularization factor and $R(\alpha)$ a regularizer. We implement the two proposed regularizers : the standard deviation of the attention vector (13) and its negative entropy (14).

$$R(\alpha) = \sigma(\alpha) \tag{13}$$

$$R(\alpha) = \sum_{i}^{N} \alpha_i * log(\alpha_i) \tag{14}$$

Using a the standard deviation regularization on an ATAE-LSTM enables us to reach a mean accuracy of 65.96% and a mean f1-score of 65.47%, a slight improvement its score without regularization. The negative entropy regularization didn't give convincing results.

7 Exploring other pre-trained word embeddings

To go beyond the paper we explored a way of using another trending pre-trained embedding weights method which is *fastText* from the paper from Mikolov et al. [5]. *Fasttext* represents each word as an n-gram and uses a *skip-grams* to learn

the embeddings. It works well on rare words which are not necesarily in the training corpus. This resulted in a drop of performances on the ATAE-LSTM by 2.3%.

A way of improvement could be to explore the idea of using the glove embedded vectors for the words both in our data and in the vocabulary of *GloVe* and using *fastText* for those not in *GloVe* vocabulary.

Conclusion

In this project, we implemented attention-based LSTMs for aspect sentiment classification. This gives a better analysis of a sentence than usual classification LSTMs. The results showed that AE, AT, and ATAE-LSTM give performances we expected. Different ways of improving our models have been experimented, such as Xavier initialization, pre-trained word embeddings techniques, and attention regularization.

For a future work, it could be interesting to improve the word embeddings, by combining *Glove* and *fastText*.

References

[1] Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. Attention-based LSTM for aspect-level sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 606–615, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1058. URL `https://www.aclweb.org/anthology/D16-1058`.

[2] Lingxian Bao, Patrik Lambert, and Toni Badia. Attention and lexicon regularized LSTM for aspect-based sentiment analysis. In Fernando Emilio Alva-Manchego, Eunsol Choi, and Daniel Khashabi, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28 - August 2, 2019, Volume 2: Student Research Workshop*, pages 253–259. Association for Computational Linguistics, 2019. doi: 10.18653/v1/p19-2035. URL `https://doi.org/10.18653/v1/p19-2035`.

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `http://proceedings.mlr.press/v9/glorot10a.html`.

[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

[5] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.