Twitter Sentiment Analysis : A comparative approach

Mariem BOUHAHA Eden BELOUADAH

23 janvier 2018

1 Introduction

Text classification is a class of problems that finds application in many tasks, mainly sentiment analysis, predicting movie reviews as well as classifying emails as spam or not.

In recent literature, deep learning seems to be promising for text classification, achieving state-of-the-art results on a suite of standard benchmark problems.

In this project, we aim to compare and analyze the performance of several deep architectures that we use for sentiment analysis from tweets. For that, we studied the behavior of some deep neural networks as well as the influence of data representation on the models' performances.

In this intermediate report, we will present the data in hand as well as some statistics to describe it, then we will give a brief description of the models tested so far and finally present the preliminary results.

2 Presenting the data

For this project, we used the B-task SemEval-2013 [1] training and test data, which have the following characteristics before any preprocessing :

Set	Number of examples	Positive	Negative	Neutral
Train	5916	2179	816	2921
Test	876	324	158	394

Tableau 1. Classes distribution for train set and test set

If we visualize an extract (the first 5 for example) of the tweets in the training set, we get the text shown in figure 1.

We noticed that, in addition to words, tweets contain hashtags, URLs and user tags, which are less likely to be useful for sentiment prediction.

```
print(messages[:5])
```

```
["Gas by my house hit $3.39!!!! I'm going to Chapel Hill on Sat.
:)", "Iranian general says Israel's Iron Dome can't deal with the
ir missiles (keep talking like that and we may end up finding ou
t)", 'with J Davlar 11th. Main rivals are team Poland. Hopefully
we an make it a successful end to a tough week of training tomorr
ow.', "Talking about ACT's && SAT's, deciding where I want to go
to college, applying to colleges and everything about college str
esses me out.", "They may have a SuperBowl in Dallas, but Dallas
ain't winning a SuperBowl. Not with that quarterback and owner. @
S4NYC @RasmussenPoll"]
```

Figure 1. A small extract from Train

We also computed some statistics for the tweets, which we summarize in table 2.

Type	**Positive tweets**			**Negative tweets**			**Neutral tweets**		
Length	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
Train	4	44	23	6	38	23	5	40	23
Test	7	35	23	8	34	23	6	38	22

Tableau 2. Statstics about tweets length for train set and test set before data preprocessing

3 Data Preprocessing

Since we cannot use the "dirty" raw text tweets for classification, we did some preprocessing as following :

— Tokening tweets using *NLTK Tweet Tokenizer* which is more appropriate for this type of texts,

— Using regular expressions to remove digits (24, 12.400$...), punctuations (?, :, ;,",#, @...), tags (@user) and every sequence of special characters. However, the emoticons are kept,

— Using *NLTK Stop words* to remove all useless words,

— Preprocessing Hashtags : removing # and separating the hashtag parts (#VeryHappy -> Very + Happy)

— Transform all tweets to lower case.

These operations left us with a training vocabulary of size 17284, where preprocessed tweets now have the following statistics :

Type	**Positive tweets**			**Negative tweets**			**Neutral tweets**		
Length	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
Train	3	22	12	3	20	13	3	22	12

Tableau 3. Statistics about tweets length for train set and test set after data preprocessing

In figure 2, we show the distribution of word frequencies in our training set.

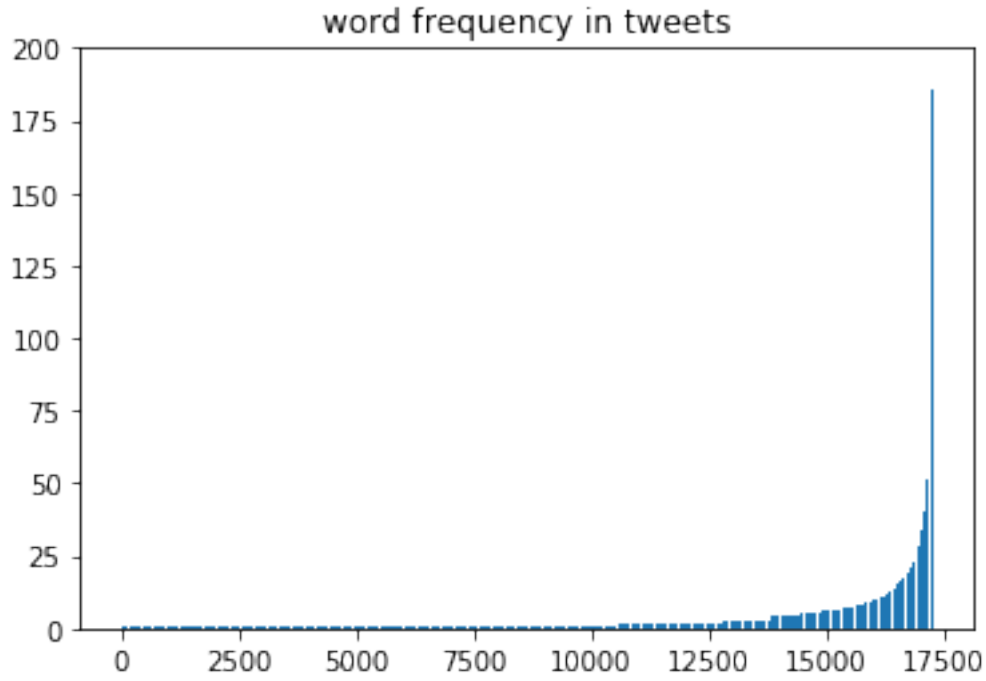

Figure 2. Word frequency

4 Building models

4.1 Multi-Layer Perceptron (MLP)

We tried to start with a simple model like MLP classifier, so we trained it on a word embedding-based representation of our tweets.

For that we used the Word2Vec embeddings for our Vocabulary and represented each tweet by the sum of the embeddings of its constituents. That is, if a tweet is "Happy New Year", the value of this observation would be the sum of the embeddings of "Happy", "New" and "Year". And we do this for all the elements in the training set. Thus, using an embedding dimension of 100, the shape of our training set now becomes (number of tweets, 100).

We then trained the MLP classifier with one hidden layer of size 100 neurons, using stratified 10-fold cross-validation. We obtained the following results :

Fold	0	1	2	3	4	5	6	7	8	9	**Mean**
Accuracy	0.513	0.540	0.513	0.522	0.5	0.530	0.521	0.496	0.487	0.493	**0.51**

Tableau 4. Accuracy obtained for each fold

As we can see, the MLP performance is slightly better than a random classifier which would give us an accuracy of 33% but it's not satisfying for our problem.

4.2 Long Short Term Memory (LSTM)

Since the MLP classifier didn't give any good results, we decided to use recurrent neural networks which are models specialized in sequential data like the natural text in which the position of the words plays a significant role in the meaning. More precisely, we have chosen LSTM which is a robust model of the RNNs that can keep record of the long term and the short term dependencies between the parts of the data.

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell is responsible for learning long-term dependencies through arbitrary time intervals ; this allows LSTM model to handle the exploding and vanishing gradient problems that traditional RNNs suffer from.

4.2.1 Data matrix

In order to train LSTM model, an input matrix must be constructed with 3 dimensions : batch size × sequence length × embeddings length, where :

— Batch size = number of tweets,

— Sequence length = length of the longest tweet (in the train set),

— Embedding length : size of embeddings fixed at 100.

As we notice, tweets have different lengths. To handle this, we take the size of the longest one and use it as a reference. The tweets having the same number of words are kept as they are, and the shorter ones are duplicated until their sizes reach that of the longest tweet.

Words embeddings are calculated using *word2vec* model which has been trained on the training set. Words from test set that don't exist in training set are replaced with the <unknown> word.

4.2.2 Model architecture

The architecture of our model is as following :

— **LSTM layer** with a hidden state size of 200. The output of this layer is a 3 dimensional matrix in which the first 2 dimensions are as always the *batch size* and the *sequence length*, but the last one will be hidden state size, which is equal, in our case, to 200.

— **Dropout layer** with a probability of dropping equals to 0.2 (20% of the output of the LSTM layer is zeroed at random locations in each training iteration).

— **Fully connected layer** having 2 dimensions, the first one is sequence length × LSTM hidden size, and this is caused by the output of the LSTM layer which is 3 dimensional while the input of the Dropout layer is only 2 dimensional, so the last 2 dimensions of the first layer's output are flattened into a single dimension.

The optimizer used for training is **Adam**. It has been chosen instead of the simple *Stochastic Gradient Descent (SGD)* since it is robust against local optima.

The loss function used is the **Cross Entropy Loss**, because we are dealing with a classification problem.

After performing the forward propagation, we backward the error gradient and update the parameters of the model. The network's memory is emptied after each iteration. The dropout layer is used as a regularization trick because it has a big impact on the model's performance. More details will be discussed later.

4.2.3 Implementation and Results

We used **PyTorch** to implement our LSTM model. The program shows real-time plots for accuracy and the Cross Entropy Loss for both train and test sets.

The next figure shows the plots for the whole training process :

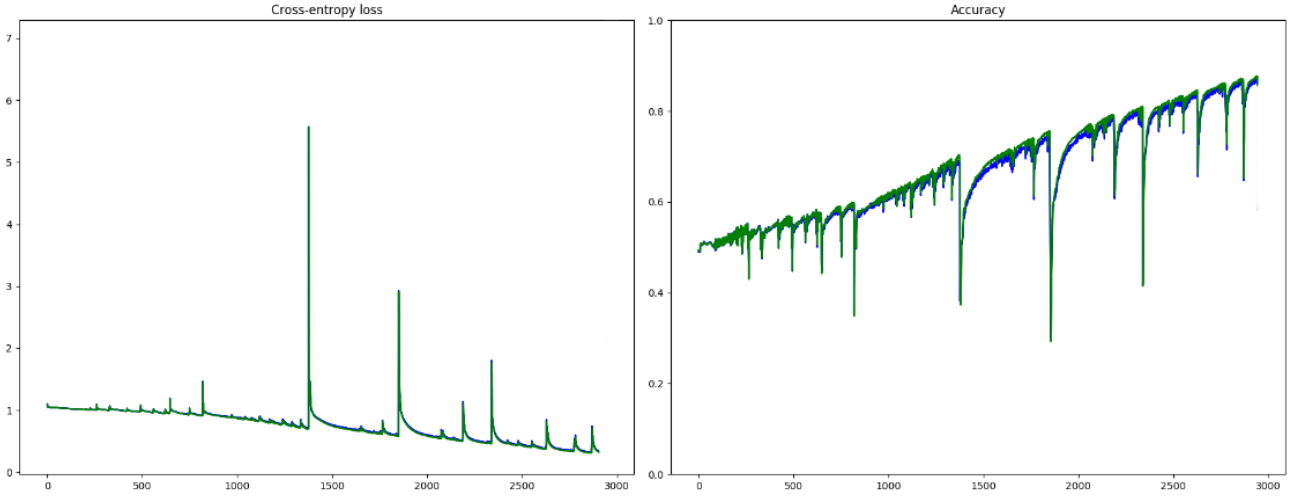

Figure 3. Evolution of Accuracy and Cross Entropy Loss through the whole training process (Blue : Train, Green : Test)

The program also allow interrupting the training process (if under/over-fitting are observed for example or if the training process takes too long) and then saves the model's learned parameters in a file so that it will be possible to continue the learning process from where it was stopped last time.

The results of training the LSTM model after 2941 epochs (running it for 9 hours on Intel Core i5 processor) are the following :

Set	Train	Test
Accuracy	86.76%	87.54%
Cross Entropy Loss	0.3094	0.2975

Tableau 5. Accuracy and Cross Entropy Loss obtained with LSTM

Why dropout layer is so important ? Before adding the dropout layer, the model was over-fitting the data, and the error on test set was diverging from that of training set since the first iterations ! Thus, the model was performing very well on the training data while it was very bad on the test data.

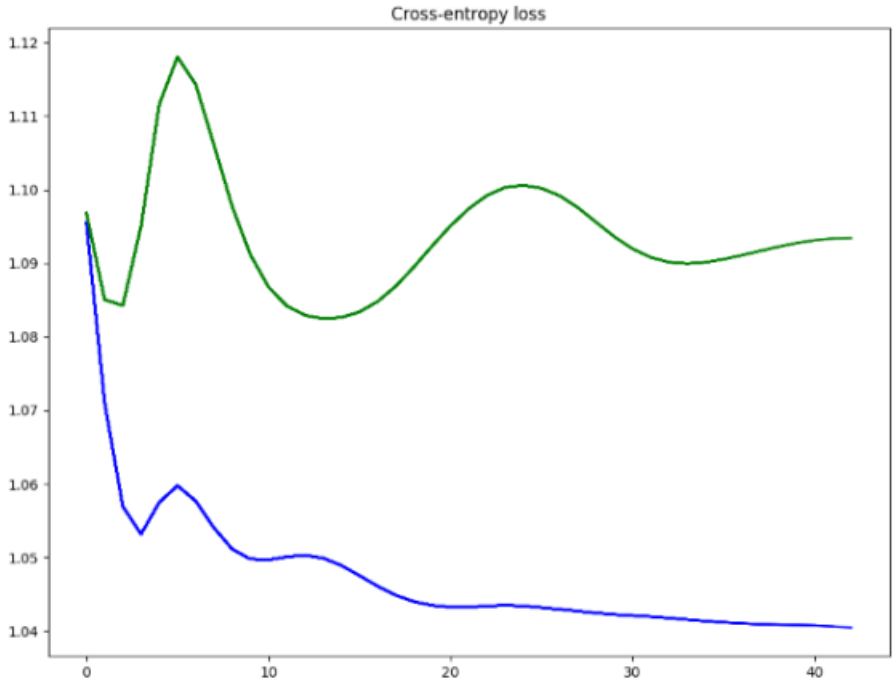

Figure 4. Cross Entropy Loss before adding dropout

4.2.4 Discussion of results

Comparing our LSTM model with the work done by Ye Yuan, You Zhou in [2], we find that their One-hidden-layer RNN has reached only **63.71%** of accuracy on the same dataset that we used. Therefore, our LSTM model is clearly better.

We also tried the mini-batch mode but the program was very slow compared to batch mode. The latter gave us satisfying results so we decided to use it in the final version of the program.

5 Conclusion

In this work, we compared the performance of two neural networks : MLP and LSTM classifiers. The performance of the latter was better on tweets classification task. Therefore, LSTM is more suitable for sentiment analysis from the text thanks to its long-term memory that is not available in general models like MLP.

In addition, we found that many factors can control the efficiency of our model, especially data preprocessing quality, regularization and network architecture.

Text classification remains a large field of research and the subject of many real-world problems.

6 References

[1] Semeval 2013 Task 2 Data. http ://www.cs.york.ac.uk/semeval-2013/task2/. Retrieved on January 4th 2018.

[2] Yuan, Y. and Zhou, Y., 2015. Twitter Sentiment Analysis with Recursive Neural Networks. CS224D Course Projects.