Textual sentiment Analysis via two different neural networks and SentiBabelNet.

Emanuele Alessi

Summary

1. Introduction

Sentiment Analysis is the process of determining whether a piece of writing is positive, negative or neutral. A sentiment analysis system for text analysis combines natural language processing (NLP) and machine learning techniques to assign weighted sentiment scores to the entities, topics, themes and categories within a sentence or phrase.

Sentiment analysis can help data analysts within large enterprises gauge public opinion, conduct nuanced market research, monitor brand and product reputation, and understand customer experiences. In addition, data analytics companies often integrate third-party sentiment analysis APIs into their own customer experience management, social media monitoring, or workforce analytics platform, in order to deliver useful insights to their own customers.

In this project I will propose two frameworks:

- SentimentAnalyzer: this model combines word tokenization, word lemmatisation, Part-Of-Speech (POS) tagging, word-sense disambiguation (WSD) using BabelNet synsets and, finally, sentiment analysis;
- SentimentClassifier: this model consists of a model inspired by state of the art architecture and, therefore, more performance-oriented.

2. Problem definition

The objective is to design a model, totally from scratch, that is able to do sentiment analysis of english text documents through a straightforward process:

- Break each text document down into its component parts (sentences, phrases, tokens and parts of speech)
- Identify each sentiment-bearing phrase and component
- Assign a sentiment score to each phrase and component (0 to +1)
- Combine scores for multi-layered sentiment analysis

More precisely, the SentimentAnalyzer's model implementation will be explained sequentially in the next chapters, from tokenization to sentiment classification, while the SentimentClassifier's computational process will be represented in a more straughtforward way because of its more immediate computational process.

3. Project setup

The technologies that I used while working on this project (in order of importance):

- Python 3.7
- TensorFlow 1.15 (backup of the models are compatible also with the newer releases)
- NumPy
- TQDM

3.1 TensorFlow

TensorFlow is the most famous framework for working out any large-scale Machine Learning: originally created by the Google Brain Team, it is an open-source library which bundles mainly Deep Learning models and algorithms. The library can train and run Deep Neural Networks for many tasks, ranging from digit classification to image recognition.

But how does it work?

TensorFlow allows the creation of so-called"dataflow graphs"; structures that describe how data moves through a graph. Here:

- A node represents a mathematical operation;
- An edge between two nodes symbolize a "Tensor" (short for multidimensional array).

The nodes, though, are not executed in Python: to ensure a higher speed of computation, in fact, the library executes these operations in C++, so that they can be worked out at low-level. Another great advantage is that the developer can choose to execute calculations either on the CPU or the GPU, to ensure more computational power to the program.

Figure 1 - Tensorflow logo

What's new in TensorFlow 2.0 version?

TensorFlow 2.0 will focus on simplicity and ease of use, featuring updates like:

- Easy model building with Keras and eager execution.
- Robust model deployment in production on any platform.
- Powerful experimentation for research.
- Simplifying the API by cleaning up deprecated APIs and reducing duplication.

Recently, Google announced that Keras, a user-friendly API standard for machine learning, will be the central high-level API used to build and train models. The Keras API makes it easy to get started with TensorFlow. Importantly, Keras provides several model-building APIs (Sequential, Functional, and Subclassing), so you can choose the right level of abstraction for your project.

In other words, the training of the neural networks will be more compact and interactive thanks also to the integration of Keras within it and the computational power seems to be more performant, probably because of the compatibility with the new NVidia CUDA versions.

4. SentimentAnalyzer's description

The following are the chapters that describe in detail its implementation.

4.1 Training and test datasets

The SemCor corpus is an English corpus with semantically annotated texts. The semantic analysis was done manually with WordNet 1.6 senses (SemCor version 1.6) and later automatically mapped to WordNet 3.0 (SemCor version 3.0). I used a SemCor version which is mapped on BabelNet synsets.

This sense-tagged corpus SemCor 3.0 was automatically created from SemCor 1.6 by mapping WordNet 1.6 to WordNet 3.0 senses. SemCor 1.6 was created and is property of Princeton University.

The corpus has also multi-word expressions (MWE) marked with underscore (_), e.g. manor_house. These multi-word units were annotated by Siva Reddy.

SemCor was tagged by TreeTagger using Penn TreeBank tagset.

The following are the test datasets (these too are mapped with babelnet synsets):

- senseval2
- senseval3
- semeval2007
- semeval2013
- semeval2015

4.2 Text tokenization

Text pre-processing plays essential role in text mining analysis, especially in case of analysis of social media posts. The application of text pre-processing steps has one main reason and it is reduction of feature set, which might otherwise result in sparsity.

First of all, given a sentence, the text tokenization procedure normalizes all letters in lower case, then blank spaces will be added between all characters that are not letters or numbers.

Finally, the text tokenization function returns a list of words splitted by blank spaces.

Just to make an example, given a (dirty) sentence *s*, where:

s = " Hi! How're you???I ' m Emanuele, this is the AFC's project "

and a tokenization function *tokenize*, then:

tokenize(s) = ['hi', '!', 'how', "'re", 'you', '?', '?', '?', 'i', "'m", 'emanuele', ',', 'this', 'is', 'the', 'afc', "'s", 'project']

4.3 Lemmas and Part-Of-Speech tags prediction

Lemmatisation in linguistics is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma. In computational linguistics, lemmatization is the algorithmic process of determining the lemma of a word based on its intended meaning. Unlike stemming, lemmatisation depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document. As a result, developing efficient lemmatisation algorithms is an open area of research.

Word	Lemmatisation	Stemming
was	be	Wa
studies	study	studi
studying	study	study
troubling	trouble	troubl
troubles	trouble	troubl

Figure 2 – Lemmatisation vs stemming

The second phase consists of training a deep learning neural network that, given a tokenized sentence, can recognize, for each word, both its lemma and POS tag.

After several searches, I decided to train a bidirectional Long-Short Term Memory (LSTM) model, using the multitask learning technique, in order to make it capable of simultaneously making predictions for different tasks.

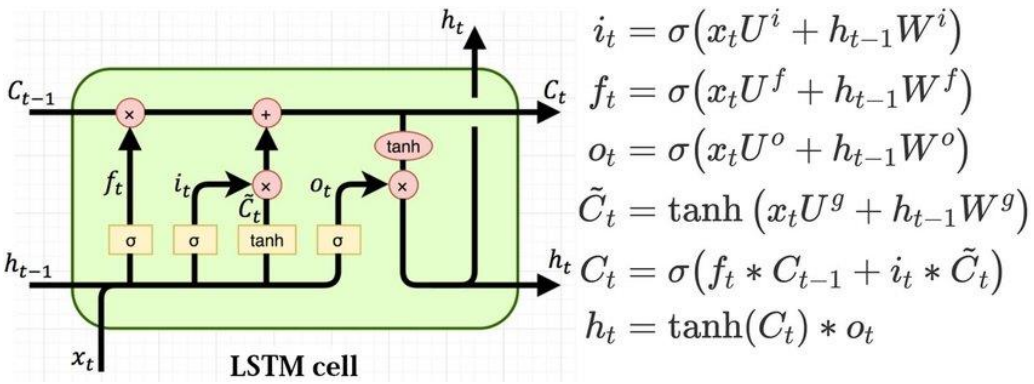

Figure 3 - Algebraic operations in a single LSTM cell

Briefly, the model is composed using many layers, such as:

1. **Input**: the preprocessed sentence (list of words mapped to integer ids);
2. **Embedding lookup**: in this layer, I used pretrained Glove embeddings (100-dimensions) in order to create an association between input (word ids) and its word embeddings. I also fixed this layer as not trainable;
3. **LSTM (forward)**;
4. **LSTM (backward)**;
5. **Concatenation layer**: between forward (3.) and backward (4.);
6. **Dense layer (lemmas):** layer of *l* neurons, where *l* is the number of possible lemmas;
7. **Dense layer (POS tags):** layer of *p* neurons, where *p* is the number of possible POS tags.

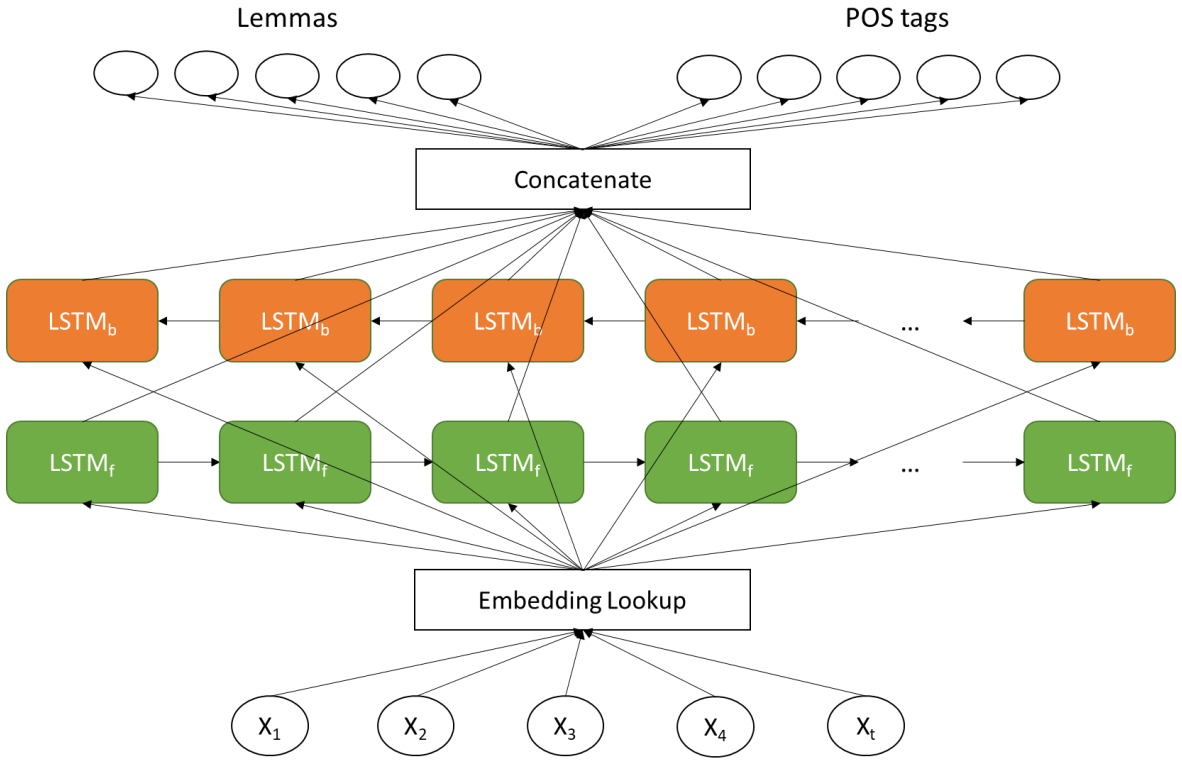

Figure 4 - Multitask model architecture (from bottom to the top)

Regarding the loss function, sparse softmax cross entropy has been used to optimize the classification tasks.

$$H(p,q) = -\frac{1}{N}\sum_{i=1}^{N} p(x_i)\log_2 q(x_i)$$

Figure 5 – Cross entropy formula applied on p (labels) and q (prediction scores). N=batch size

After many attempts of hyperparameter tuning, i have noticed that the optimal performances are obtained fixing 100 LSTM cells, both for the forward LSTM and the backward LSTM, and batch size set to 16 sentences.

The final model has been trained for 5 epochs. The following are the performance, in terms of accuracy, obtained for each test dataset.

Dataset	Lemmas accuracy	POS tags accuracy
senseval2	91.99%	84.59%
senseval3	91.66%	84.42%
semeval2007	90.19%	82.88%
semeval2013	90.21%	82.96%
semeval2015	88.59%	84.52%

Figure 6 - Multitask model performances

4.4 Word sense disambiguation

In computational linguistics, word-sense disambiguation (WSD) is an open problem concerned with identifying which sense of a word is used in a sentence. The solution to this problem impacts other computer-related writing, such as discourse, improving relevance of search engines, anaphora resolution, coherence, and inference. The human brain is quite proficient at word-sense disambiguation. That natural language is formed in a way that requires so much of it is a reflection of that neurologic reality. In other words, human language developed in a way that reflects the innate ability provided by the brain's neural networks.

In English, on fine-grained sense distinctions, top accuracies from 59.1% to 69.0% have been reported in evaluation datasets (SemEval-2007, Senseval-2), where the baseline accuracy of the simplest possible algorithm of always choosing the most frequent sense was 51.4% and 57%, respectively.

In this task, I tried to designed an efficient neural architecture able to outperform the performances of the baselines.

The proposed model is quite similar to the previous one. Its layers are:

1. **Input**: combination of preprocessed sentence (list of words mapped to integer ids), POS tags of the sentence (list of tags mapped to integer ids), and a list of flags composed by 0 or 1 values in order to add informative content to the sentence: for each word *w*, if *w* has an associated synset, and therefore it is ambiguous, then its value will be 1, otherwhise 0;
2. **Embedding lookup**: as in the previous model, I used pretrained Glove embeddings (100-dimensions) in order to create an association between input (word ids) and its word embeddings. I also fixed this layer as not trainable;
3. **LSTM (forward)**;
4. **LSTM (backward)**;
5. **Concatenation layer**: between forward (3.) and backward (4.);
6. **Dense layer (synsets):** layer of *s* neurons, where *s* is the number of possible BabelNet synsets (about 30000).

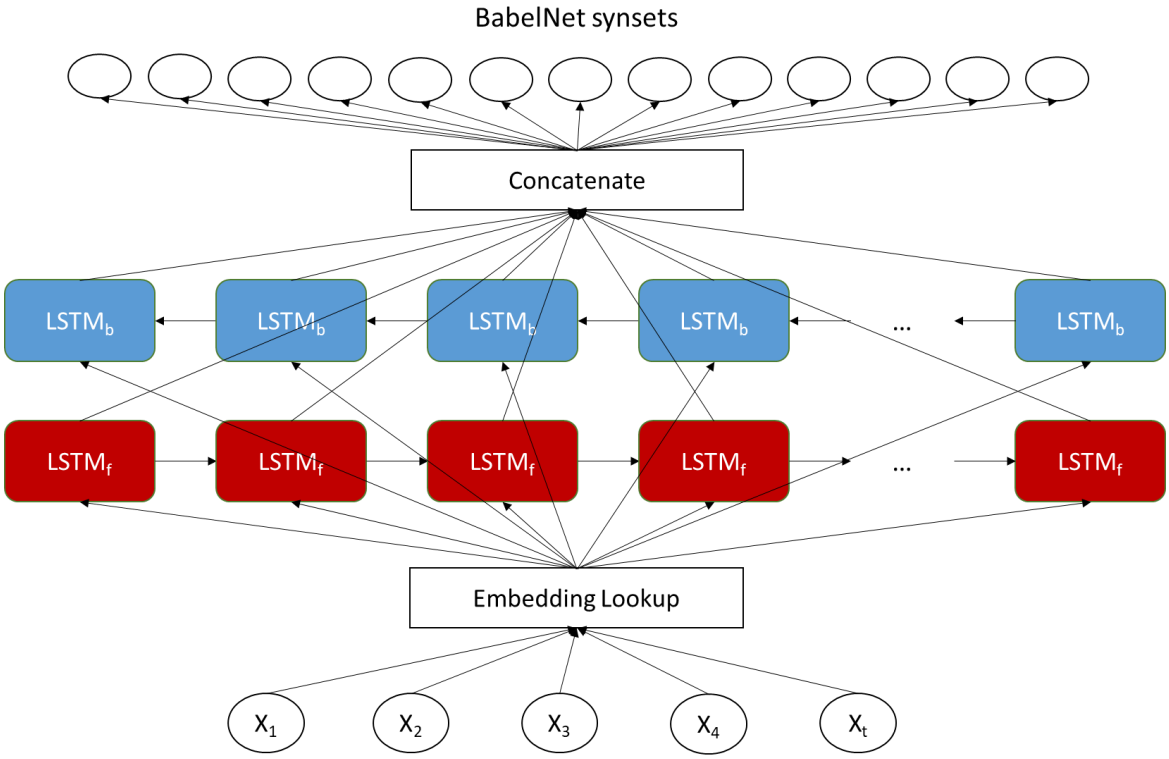

Figure 7 - WSD model architecture (from bottom to the top)

Also in this case, the model was trained using the sparse softmax cross entropy calculated on a batch of 16 sentences.

After 5 training epochs, the following are the obtained performance, in terms of F1, precision and recall.

Dataset	F-measure	Precision	Recall
senseval2	75.80%	61.04%	100%
senseval3	74.66%	59.57%	100%
semeval2007	69.16%	52.86%	100%
semeval2013	71.46%	55.59%	100%
semeval2015	73.65%	58.29%	100%

Figure 8 - WSD model performances

4.5 SentiBabelNet

SentiBabelNet is the result of the automatic annotation of the synsets of BabelNet according to the notions of "positivity" and "negativity". Eachsynset *s* is associated to three numerical scores *Pos(s)*, and *Obj(s)* (the latter expressed as *1 – (Pos(s) + Neg(s))*) which indicate how positive, negative, and "objective" (i.e., neutral) the terms contained inthe synset are.

Different senses of the same term may thus have different opinion-related properties.

In this project, I used the english parsed synsets of SentiBabelNet, which are almost 83000.

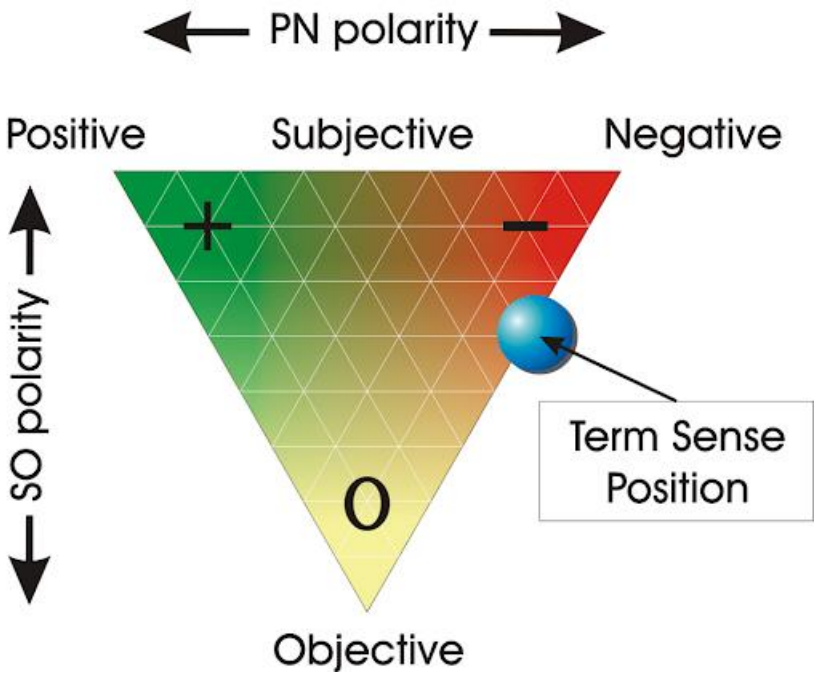

Figure 9 - Example of SentiBabelNet graphical representation

4.6 Sentiment prediction

Once trained the two neural networks, they will be part of the computational process to the sentiment calculation.

The final model returns, for a given sentence *s*, two probability values:

- *P(sentiment="Positive" | s)*
- *P(sentiment="Negative" | s)*

The resulting values are included within a vector and the corresponding vector will be returned.

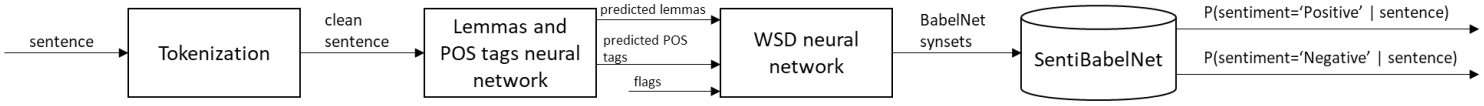

Figure 10 - Computational flow for sentiment classification

The implementation described above is represented in a python class named SentimentAnalyzer, whose methods will be described in the next sixth paragraph.

5. SentimentClassifier's description

In the very next chapters I will describe the implementation of the state of the art inspired model.

5.1 Train and test dataset

IMDB dataset having 50K movie reviews for natural language processing or Text analytics. This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. They provide a set of 25,000 positive movie reviews for and 25,000 negative movie reviews. So, predict the number of positive and negative reviews using either classification or deep learning algorithms.
Regarding the download link, the IMDB dataset can be downloaded from these links:

http://ai.stanford.edu/~amaas/data/sentiment/

Or

https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

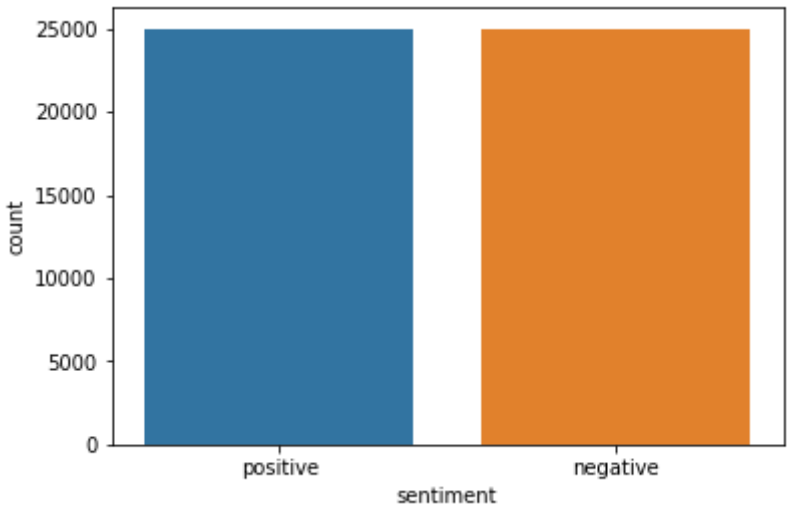

Figure 11 - Labels distribution of IMDB dataset

5.2 Text preprocessing

I saw that the dataset contained punctuations and HTML tags. In this section I defined a function, named *tokenize*, that takes a text string as a parameter and then performs preprocessing on the string to remove special characters and HTML tags from the string. Finally, the resulting value is returned to the calling function as a list of strings splitted by blank spaces.

The preprocessing function is quite similar to the one described in the chapter 4.2.

5.3 Model architecture

The classifier's architecture is made in such a way that guarantees higher accuracy performances. As mentioned before, the given model is trained on IMDB reviews dataset, more precisely, 70% of its instances are used for training, and the remaining 30% for testing the performances.

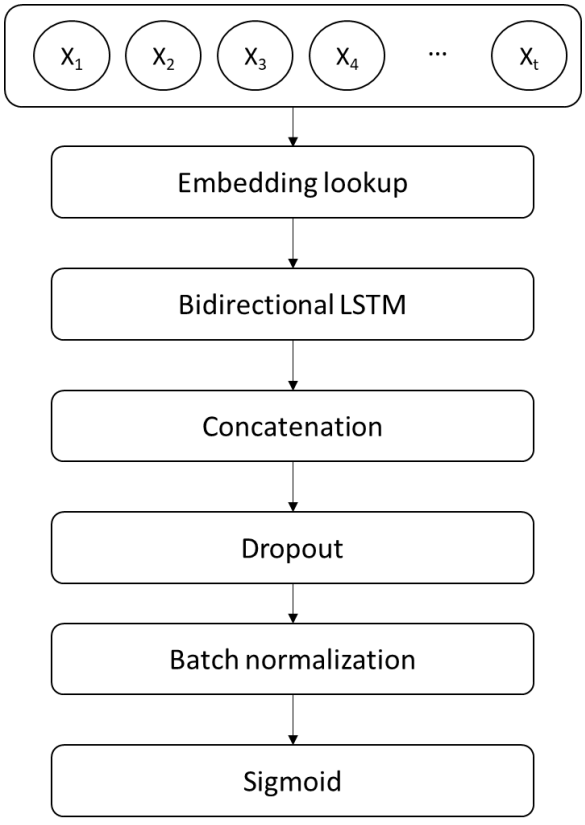

Figure 12 - Sentiment classifier's architecture from input (top) to output (bottom)

Regarding the bidirectional LSTM layer, 256 cells are used (128 for forward and 128 for backward), then the resulting tensors are concatenated and the following tricks have been applied:

- **Dropout**: It can be used with most types of layers, such as dense fully connected layers, convolutional layers, and recurrent layers such as the long short-term memory network layer. Dropout may be implemented on any or all hidden layers in the network as well as the visible or input layer. It is not used on the output layer. The dropout value was set to 0.5, in other words, neurons are dropped with the probability of 0.5.
- **Batch normalization**: it normalizes the activation of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. It addresses the problem of internal covariate shift. It also acts as a regularizer, in some cases eliminating the need forDropout. It helps in speeding up the training process.

The output is represented by only one neuron with sigmoid activation. If the output is less than 0.5, then the result will be positive; negative otherwise.

$$output(x) = \begin{cases} positive\ sentiment & if\ output(x) \leq 0.5 \\ negative\ sentiment & if\ output(x) > 0.5 \end{cases}$$

Figure 13 - Output formula

As regards the loss, the model is trained with the binary cross entropy.

$$loss = \frac{1}{N}\sum_{i=1}^{N} -Y \log Y_{pred} - (1 - Y) \log Y_{pred}$$

Figure 14 – Cross entropy formula applied on Y (labels) and Y_{pred} (prediction scores). N=batch size

5.4 SentimentAnalyzer vs SentimentClassifier

After traning both models, they are tested on the remaining 30% of the IMDB reviews dataset. The following are the results, in terms of accuracy:

Model	Accuracy
SentimentAnalyzer	58,31%
SentimentClassifier	88,81%

7. SentimentAnalyzer API

7.1 sa.tokenize

```
sa.tokenize(sentence)
```

It transforms the *sentence* (string) into a list of words applying many text preprocessing techniques based on regular expressions.

7.2 sa.predict_lemmas_and_pos_tags

```
sa.predict_lemmas_and_pos_tags(sentence)
```

Given a *sentence* (string), it returns, through the lemmas and pos tags neural network, a list of lemmas and a list of POS tags.

7.3 sa.predict_synsets

```
sa.predict_synsets(sentence=None, lemmas=None, pos_tags=None)
```

- If *sentence* is not *None*, it starts the computational flow through the lemmas and pos tags neural network, and then through the WSD neural network, returning a list of BabelNet synsets.
- If *lemmas* and *pos_tags* are not *None*, it starts the computational flow through the wsd neural network and returns the list of BabelNet synsets.

7.4 sa.predict_proba

```
sa.predict_proba(sentence, from_synsets=False)
```

- If *from_synsets* is *False*, *sentence* (string) will be computed through the lemmas and pos tags neural network, and then through the WSD neural network, returning a vector of probabilities.
- If *from_synsets* is *True*, then *sentence* is represented as a list of BabelNet synsets, thus it starts the computational flow through the WSD neural network, returning a vector of probabilities.

7.5 sa.predict

```
sa.predict(sentence)
```

Given a *sentence* (string), it returns, through the *argmax* function applied on the probabilities vector *v*:

- "Positive" if *v[0]* > *v[1]*;
- "Negative" if *v[1]* > *v[0]*;
- "Neutral" if *v[0]* = *v[1]*

8. SentimentClassifier API

8.1 sc.tokenize

```
sc.tokenize(sentence)
```

It transforms the *sentence* (string) into a list of words applying many text preprocessing techniques based on regular expressions.

8.2 sc.predict_proba

```
sc.predict_proba(sentence)
```

It returns the prediction score based on the *sentence* (string) given in input.

8.3 sc.predict

```
sc.predict(sentence)
```

Given a *sentence* (string), it returns:

- *'positive'* if *0 <= predict_proba(sentence) <=0.5*;
- 'negative' if *0.5 < predict_proba(sentence) <= 1*.