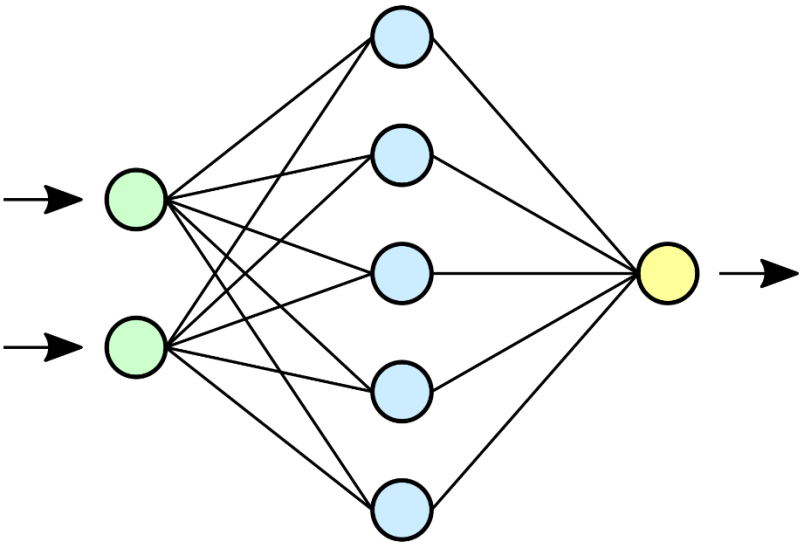
Sentiment Analysis:

Natural Language Processing



Roger Miranda Pérez

Contenido

1. Introduction

In this assignment we'll have a bunch of Amazon opinions with its score; the goal is to create something with the ability to understand the relation between the texts and score, thus being able to predict the score giving only the text.

We'll perform all the tests in a Jupyter Notebook, with all the dependencies managed by Conda (check the README.md for the installation process).

2. DNN

After the hands-on lab performed on class, we conclude that a Dense Neural Network (DNN) is the most accurate approach.

The problem of the solution proposed back there was the conversion between text to the input of the DNN. That solution used an inverse word frequency (TF-IDF) that generated (for each text) a vector of size <number of different words in the training data> with values between 0 and 1.

2.1. First approach

As a better[1] approach, the first proposed solution is to use *word2vec*[2] combined with Recurrent Neural Networks; more precisely to use a bidirectional Long-Short Term Memory (LSTM). By following this approach the text isn't immediately collapsed, but instead it process word by word while also sending the (hopefully) relevant information to the next unit. You can check the Figure 2.1, bidirectional LSTM, for reference.

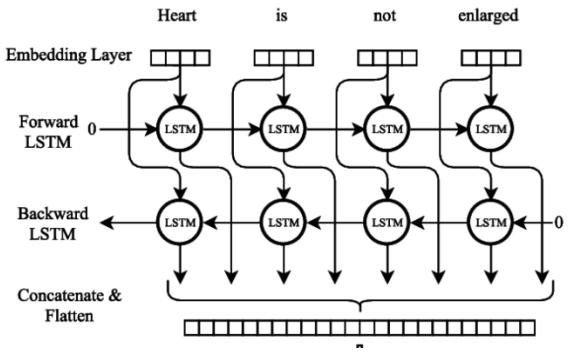


Figure 2.1: bidirectional LSTM

[1] One solution with score >0.5

[2] *word2vec* is a different approach on word representation. By representing a word by its context, you can estimate synonyms by being vectors that are really close to each other.

The LSTM approach was better than the original, but only for a bit (by pushing the DNN VRAM limits we got an accuracy of 0.62).

2.2. Second approach

After the low accuracy of the first approach, we tried to implement one different solution proposed by user neel[3]. The problem here is how we can send the high-density vector generated by the *word2vec* (<max words in review> x <*word2vec* vector size>) into the DNN. Neel propose to use the previous input approach (TF-IDF) not as the direct solution, but as an indicator to perform a mean with the *word2vec* vector.

Unfortunately, this second approach got the same accuracy (check Figure 2.2, Accuracy evolution), but with a significant lower VRAM usage, so we'll stick to that.

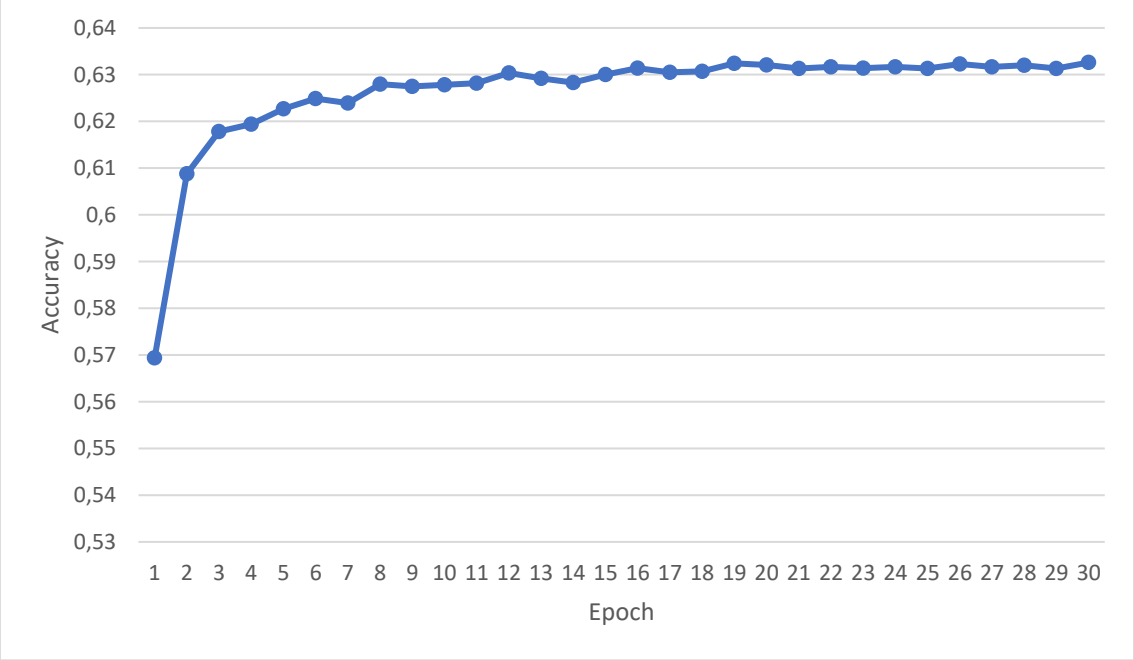


Figure 2.2: Accuracy evolution

[3] https://stackoverflow.com/a/31738627/9178470

3. Playing with *word2vec*

As word2vec represents a word by its context, it may be useful to check a bit further on what it can offer.

3.1. Making groups

To make groups out of the generated vectors we've tried a clustering algorithm (k-Means), and two density algorithms (DBSCAN and OPTICS). Unfortunately, the data was too clustered for the density algorithms to work, showing one single group.

k-Means on the other hand was able to extract some interesting groups, like words that represent "me" ("I", "me", "am"), numbers ("2", "two", "3", "5", "few"), time ("years", "year", "months", "month", "days", "weeks", "week", "seconds", "minutes"), and so on.

3.2. Looking for word correlation

Following the post made by Piotr Migdał "king- man + woman is queen; but why?"[4], thanks to how word2vec works it is possible to estimate a word by vector subtraction (as seen in Figure 3.1, *word2vec* vector subtraction).

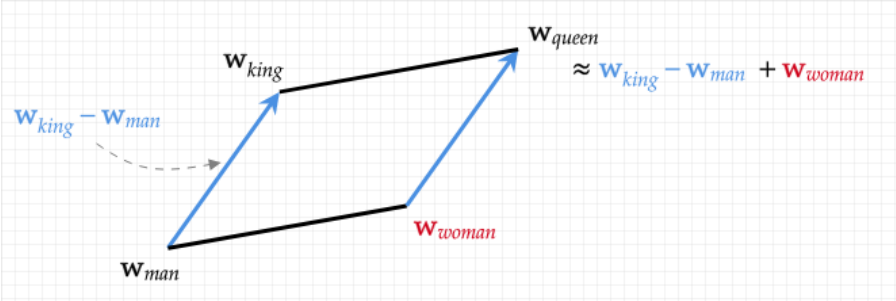


Figure 3.1: word2vec vector subtraction

The Word2Vec package that we're using has already implemented this "vector subtraction" feature, so we can easily estimate words:

```
In [5]: # It works as told here: https://p.migdal.pl/2017/01/06/king-man-woman-queen-why.html/
        # @ref https://tedboy.github.io/nlps/generated/generated/gensim.models.Word2Vec.most_similar.html
        # @ref https://stackoverflow.com/a/68681997/9178470
        def get_counterword(target: str, departing_from: str, departing_to: str, num_results: int = 10):
            return word_vectors.most_similar(positive=[target, departing_to], negative=[departing_from], topn=num_results)

        get_counterword("foot", "watch", "watches", num_results=3)

Out[5]: [('miles', 0.5845972299575806),
         ('quarters', 0.5825550556182861),
         ('feet', 0.5742671489715576)]
```

One impressive prompt is that it was able to tell numbers: "if 'two' is to 2, what is to 5?" // "five".

[4] https://p.migdal.pl/2017/01/06/king-man-woman-queen-why.html/

4. Conclusions

In this project we've learned how to process input text and how to work with *word2vec* in order to increase accuracy of the DNN.

Additionally, I've learned how to setup Jupyter and link it with a Conda environment (useful for keeping your system clean of Python packages).

Due to the high-density input of the first approach (I believe it required TB of RAM for one epoch) I was forced to research how to make a custom "input generator" (called in Tensorflow a "Sequence" class), so I've also learned how to work with big training data.