Machine Learning - Winter 2023 - Mini Project II

Alex Montoya Franco
alex.montoyafranco@abo.fi
EDISS Master's Programme - Åbo Akademi University, Turku, Finland

1. Introduction

500 million tweets are sent every day [1]. Millions of users converge on Twitter and many other social media apps to discuss a wide range of topics. To make their voices heard. And now, more than ever before, a single comment made by any person around the world could affect any given brand and could significantly damage its reputation. How should companies prepare for this? What are the tools at their disposal to quickly understand what are customers and the general public talking about? Is the conversation around a specific brand positive or negative?

In this project we aim at analyzing tweets for sentiment analysis using two different machine learning models. In this way, we compare our models and determine which are suitable methods to answer the questions presented above.

We use Recurrent Neural Networks (RNNs). They are distinguished by their "memory" as they take information from prior inputs to influence the current input and output [2]. We also use the RNN architecture of Long short-term memory (LSTM).

In addition to this model, we also use Convolutional Neural Networks (CNNs). The main advantage of CNN compared to its predecessors is that it automatically detects the important features. CNN is also computationally efficient. It uses special convolution and pooling operations and performs parameter sharing [3].

2. Data Processing

The data used in this experiment is a sample extracted from the Sentiment140 dataset, a publicly available dataset created by three graduate students at Stanford University: Alec Go, Richa Bhayani, and Lei Huang. The original data comprises approximately 1,600,000 automatically annotated tweets. For our project the initial sample had 160.000 tweets and it was reduced to 60.000 to perform computations faster.

Data contains both the tweet text and the sentiment label. Originally (0 = negative, 4 = positive). Although, 4 was changed to 1 for standardisation purposes. Data is also balanced. Originally containing 80.000 positive tweets and negative tweets. This ratio was kept when reducing the data for experimentation. No missing values were present.

Next, we present the different data processing steps

- **Converting text to Lower case.** So that same words can be detected as such and not be consider as different words based on upper case and lower case letters.
- **Removing Stopwords.** Stopwords are the words in any language which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence [4]. Examples of stop words in English are "a", "the", "is", "are" and etc.
- **Removing URLs, Numbers, and Twitter handles (@user).** Here we remove urls which might be indicatives of advertisement to a particular brand but will not be useful by themselves. Numbers are also removed and twitter handles which are widely used across the social network and might become noise for our models.
- **Removing special characters, repeating characters and punctuation.** The punctuation removal process will help to treat each text equally. For example, the word data and data! are treated equally after the process of removal of punctuation [5]. Special characters goes in line with this and repeating characters tries to get a cleaner dataset. Some tweets have shown people finishing certain words by repeating the last letter multiple times so we try to avoid this issue.
- **Removing short words.** We decide to remove words of one or two characters after analyzing some tweets and seeing words like 'vh' and contractions like 'u' that were not removed in previous steps.
- **Tokenizing the text.** Tokenization is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning [6]. In our case these units are individual words which will be later vectorized (transformed into numbers) to be processed by our models.
- **Applying Stemming.** Stemming is a technique used to extract the base form of the words by removing affixes from them. It is just like cutting down the branches of a tree to its stems. For example, the stem of the words eating, eats, eaten is eat [7]. The process of stemming is used to normalize text and make it easier to process [8].

[1] https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/
[2] https://www.ibm.com/topics/recurrent-neural-networks
[3] https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2
[4] https://medium.com/@saitejaponugoti/stop-words-in-nlp-5b248dadad47
[5] https://www.analyticsvidhya.com/blog/2022/01/text-cleaning-methods-in-nlp/
[6] https://www.tokenex.com/blog/ab-what-is-nlp-natural-language-processing-tokenization/
[7] https://www.tutorialspoint.com/natural_language_toolkit/natural_language_too
[8] https://www.geeksforgeeks.org/introduction-to-stemming/

This preprocessing of the data allowed an initial curated dataset for which we visualize frequent words for both positive and negative tweets. This gave us an initial assurance that there are clear distinctions between tweets with different labels so that future models have a good baseline to work with.

For instance, **Figure 1** shown right below highlights frequent words of positive labeled tweets. We can see in here words such as 'thank', 'good', 'love', 'nice', 'great' being highlighted, showing not only the right correlation with the label, but also the work done on preprocessing.

Figura 1: **Frequent Words of positive labeled tweets**

Moreover, **Figure 2** shows frequent words of negative labeled tweets. Here we can see more of a mix between 'positive' and 'negative' words. Although we suspect some of these positive words like 'love' might be included in either a bad or melancholic context. However, some words indeed reflect the right label, such as 'hate', 'miss', 'sorry'.

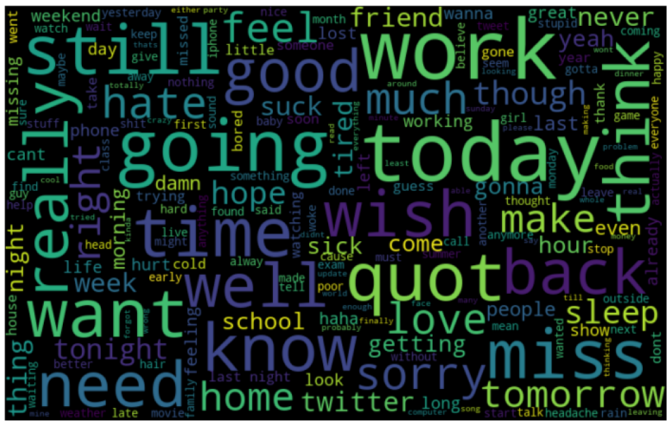

Figura 2: **Frequent Words of negative labeled tweets**

3. Modelling

We first divided our data into training and testing, as usual for this type of projects, with 80 % for training and 20 % for testing.

We transformed our tokenized texts into sequences which transforms each text to a sequence of integers. And we proceed to apply padding given that not all the sentences have the same length. In other words, naturally, some of the sentences are longer or shorter. However, we need to have the inputs with the same size, this is where the padding is necessary [9].

Finally, we have what is called a vectorized equivalent to our texts, which is what we can use when training our models.

3.1. Recurrent Neural Network (RNN)

Our RNN has an input length of 500, which was chosen as the maximum length for the padding function. We added a embedding layer and a LSTM (Long short-term memory) of 128 which means the dimensionality of the output space.

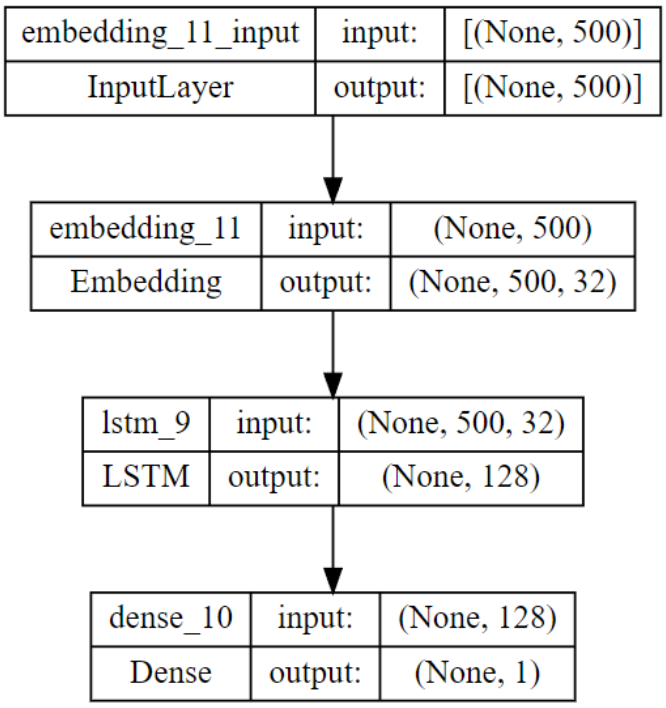

Figura 3: **RNN Architecture**

We ran this model with multiple configurations in terms of layers and epochs. We decided to focus on the presented architecture and run the program with 5 epochs. Accuracy could certainly be improved but higher sample size, additional layers and additional epochs just were adding more time to the training process and were not providing a good enough pay off. For instance, 10 epochs and an additional layer was not improving the accuracy as expected. However, time was rapidly increasing.

Figure 4 shows the accuracy for our RNN model. Training accuracy reach 78 % while validation accuracy reach 72 %.

3.2. Convolutional Neural Network (CNN)

Our CNN shares the same input length than our RNN but changes the LSTM layer for a Conv1D layer and GlobalMaxPooling1D layer. **Figure 5** shows this change.

The main difference between a CNN and an RNN is the abi-

[9] https://medium.com/@canerkilinc/padding-for-nlp-7dd8598c916a

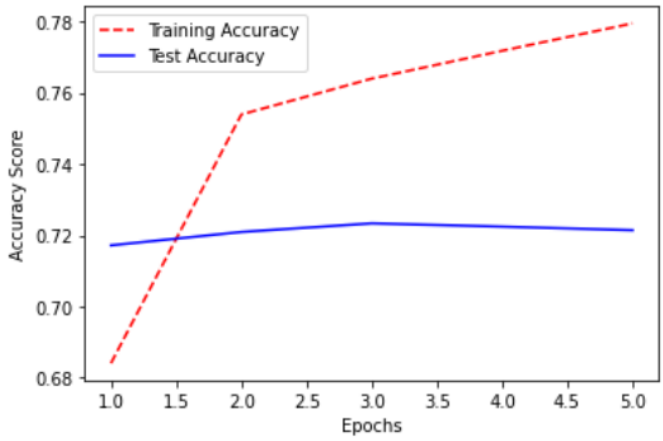

Figura 4: **RNN Accuracy**

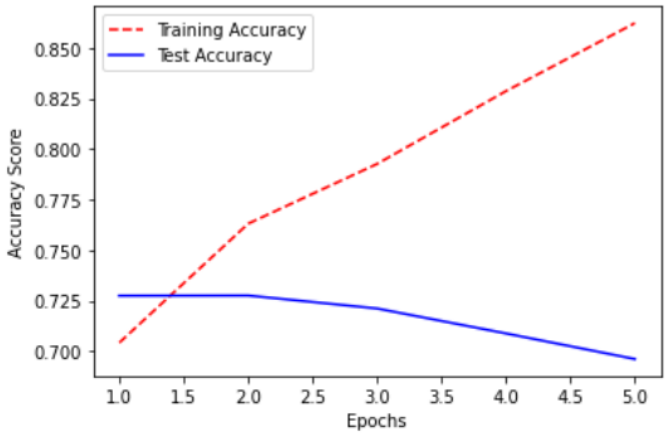

Figura 6: **CNN Accuracy**

lity to process temporal information — data that comes in sequences, such as a sentence. Recurrent neural networks are designed for this very purpose, while convolutional neural networks are incapable of effectively interpreting temporal information [10].

Use cases for CNNs include facial recognition, medical analysis and classification. Use cases for RNNs include text translation, natural language processing, sentiment analysis and speech analysis [11].

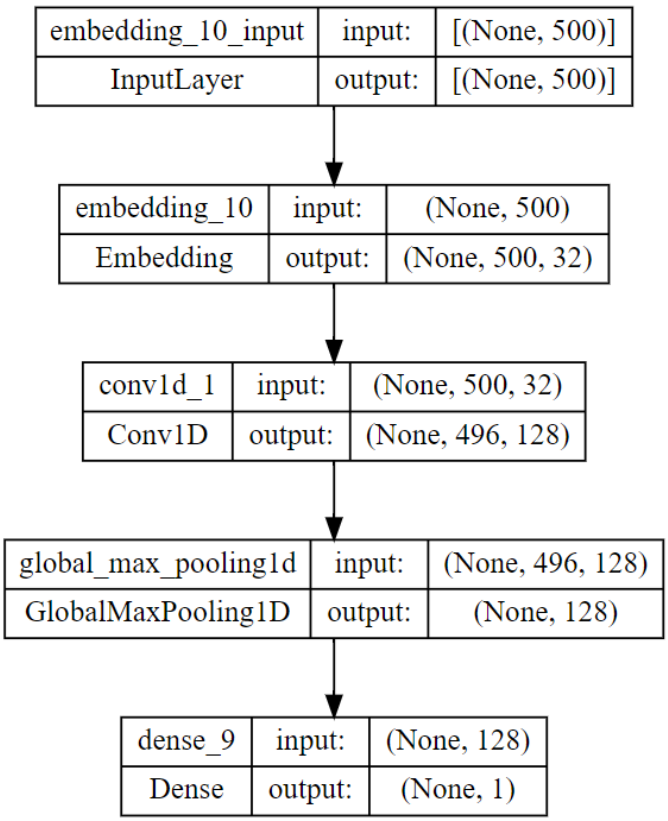

Figura 5: **CNN Architecture**

[10] https://www.telusinternational.com/insights/ai-data/article/difference-between-cnn-and-rnn

[11] https://www.techtarget.com/searchenterpriseai/feature/CNN-vs-RNN-How-they-differ-and-where-they-overlap

These examples gave us the initial impression that RNN would perform better for our use case. However, accuracy was quite similar across the two models, with CNN beating RNN in terms of training accuracy, reaching more than 80 %. However, both models underperformed in the test accuracy, in this case RNN beating CNN. RNN with 72 % and CNN with 69 %.

We recognize there is not a clear winner in terms of performance with the two architectures being rather simple.

4. Conclusion

Natural language processing is one of the pillars of the AI revolution. Giving machines the ability to understand how we communicate and even to communicate themselves in a comprehensive way for us is today, more than ever before, a reality. However, how much of this process do we really understand?

Preprocessing of text for NLP purposes was a new task but quite straight forward. A wide range of blogs and online resources agree on the common steps that we should follow to correctly prepare text for use cases like ours. However, choosing the right models from the different types of techniques available and the wide range of neural network architectures, in addition to finding the right hyperparameters and understanding how the different layers interact was the challenging part.

Different settings for the chosen neural networks showed little to none improvement in the performance. On the other hand, the time it takes to train these models grows rapidly. Our approach to these issues was to keep it simple, defining a manageable architecture, running a small number of epochs guaranteeing a minimum performance and taking advantage of the GPU runtime on our environment to speed up our training process.

More experience is certainly required with neural networks (CNN, RNN, and other architectures) and natural language processing. This exercise served as an introduction to both topics.