UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Pavlos Ntais*
Classifier: Recurrent Neural Network

Course: *Artificial Intelligence II*
Semester: *Fall Semester 2023*

Contents

1. Abstract

The task given is to develop a sentiment classifier using Recurrent Neural Network for a twitter dataset about the Greek general elections. The classifier deals with 3 classes: POSITIVE, NEUTRAL and NEGATIVE. I plan to analyze the data, clean it by pre-processing and finally start the training process.

2. Data processing and analysis

2.1. Pre-processing

The pre-processing proved to be a rather difficult task because of the vastness of the Greek language. After experimenting I came up with the following methods:

- Remove hashtags
- Remove mentions
- Remove URLS
- Remove excess tabs and white spaces
- Remove numbers and special characters
- Remove stop words
- Remove accents
- Basic emoji replacement
- Insert the party the tweet is being referred to at the beginning of the text
- Lemmatization using the spaCy library

2.2. Analysis

Upon visualizing the dataset, it became apparent that the data had a substantial degree of noise. For example spelling and grammar errors, mentions etc. Also, it naturally contained a lot of stop words, hash tags and emojies which also add noise. So, in order to process the data and remove the noise I removed stop words, hash tags and emojies and used lemmatization to group together inflected or variant forms of a word.

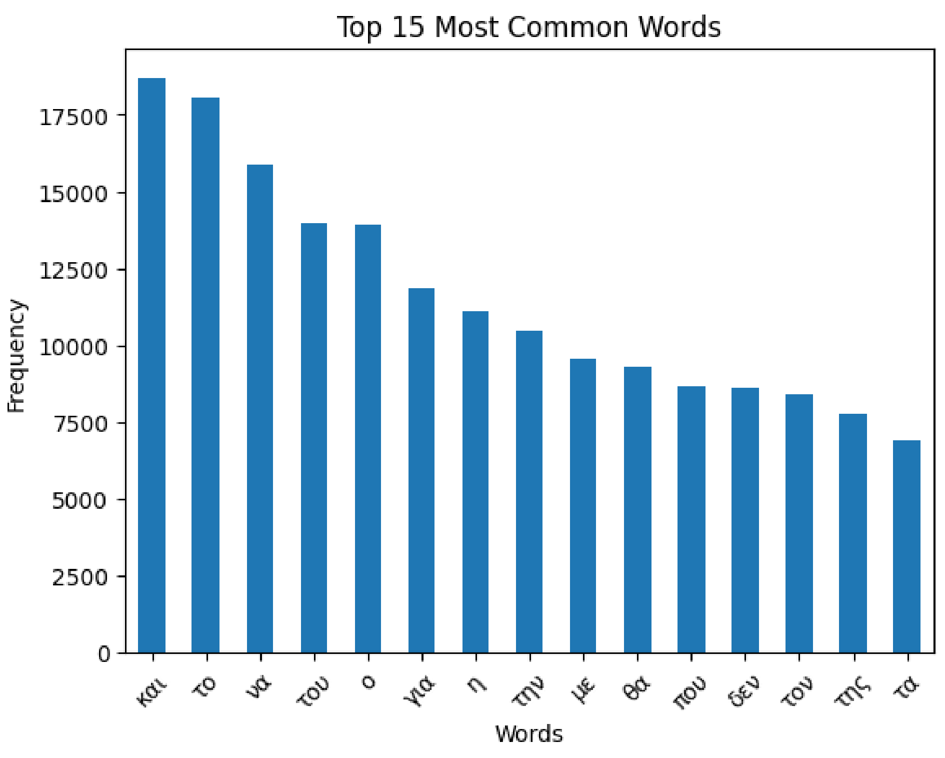

Figure 1: Before pre-processing

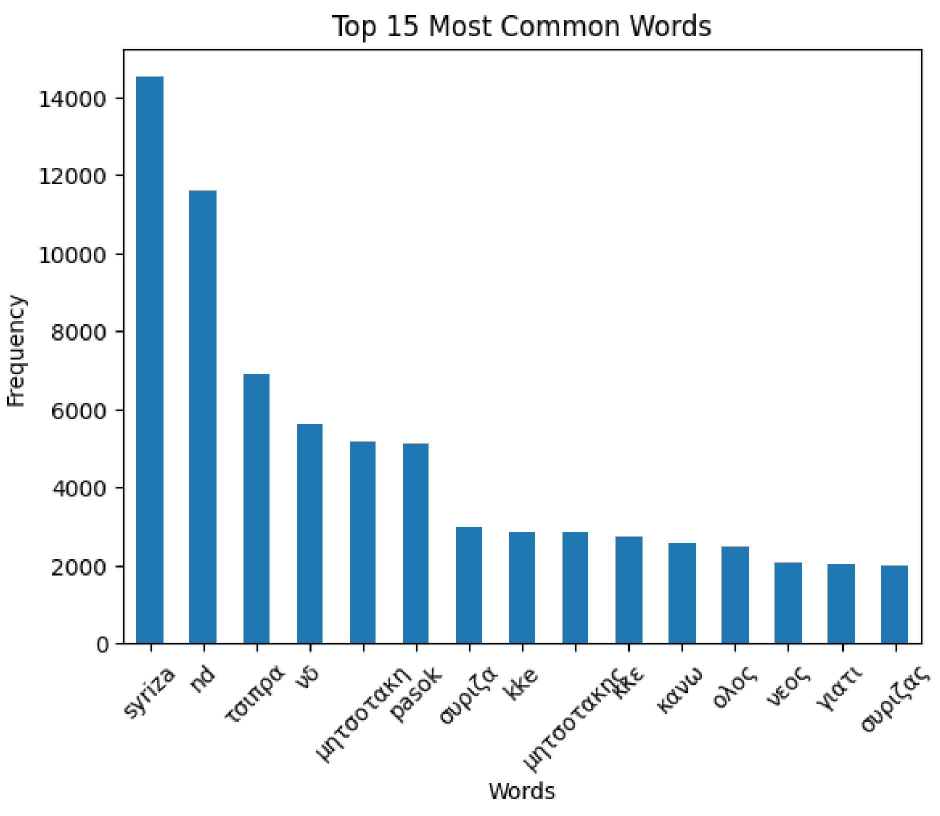

Figure 2: After pre-processing

2.3. Data partitioning for train, test and validation

For the training and testing I used the given datasets because I evaluated that their respective size is sufficient, so no partitioning occurred.

2.4. Vectorization

The pre trained word embeddings used were found on the fasttext website. Initially, we extract the embeddings, creating the embeddings array and the word-to-index dictionary. The **word-to-index** dictionary assigns a unique index to each word in the vocabulary and simply stores this mapping between all the words of the vocabulary and their indices.

At index 0, the "empty" word ("") is assigned, which does not appear in the vocabulary (represented with only zeros). I added it because it helps in converting Twitter sentences into index sequences and padding the sequences as it will be mentioned mention below. The embeddings array is simply the array where, in row **i**, we have stored the word embedding vector of the word corresponding to index i in the word-to-index dictionary.

Then, as mentioned, each sentence needs to be converted into a sequence of numbers, which will be passed as input to the models. This will no longer be done using the naive method of averaging the embedding vectors of each sentence but by converting each sentence into a sequence of integers/indices. This sequence is obtained by replacing each word in the sentence with the index corresponding to it in the word-to-index dictionary. If a word does not exist in the vocabulary (and therefore in the word-to-index dictionary), it is mapped to the empty word (index 0).

An important detail is that the length of the sequences in each batch must be constant, I chose each resulting sequence to have a predetermined, fixed, length which will be long enough not to lose much information but also short enough not to slow down training. The fixed length in this case was chosen to be 44. Therefore, the sequences of sentences that have more than 44 words are limited to the indices of the first 44 words (losing the sentiment of the remaining words), while the sequences of sentences that have fewer than 44 words are filled with the sentiment of the empty word.

3. Algorithms and Experiments

3.1. Experiments

I experimented using a different number of stacked RNNs, type of cells, skip connections. I also experimented by adding attention to the (final) model. More details below.
The optimizer tested was Adam.

Notes:

- Early stopping stopping is also used in order to stop the training process if no improvements have been made during a certain number of epochs, when it comes to the loss.

Trial	Precision	Recall	F1 score
GRU & def hyperparameters	0.37166577839204	0.36716360856269	0.35644510091321
LSTM & def hyperparameters	0.39165816043068	0.38226299694189	0.36339442717595
GRU & def hyperparameters & SC	0.37119840442594	0.37079510703363	0.34328529561066
LSTM & def hyperparameters & SC	0.38054241421705	0.37824923547400	0.37420699355515
Optuna 1	0.41327394280970	0.40672782874617	0.39825356772090
Optuna 2	0.40749675101181	0.40615443425076	0.40102608688979
Optuna 3	0.41727203313458	0.40615443425076	0.38515732127783
Final Model	0.40167904936263	0.40214067278287	0.39884574754952
Final Model & Attention	0.40897128840940	0.40271406727828	0.39264675294618

SC = Skip Connections

Table 1: Trials

3.1.1. Table of trials.

3.2. Hyper-parameter tuning & Optimization techniques

Selecting an efficient configuration for the model proved to be a rather hard task. Due to the relatively large number of hyperparameters it can be almost impossible to select the "correct" ones by hand. So, for optimization I used the Optuna framework, an open-source tool designed for automating hyperparameter optimization. I primarily fine-tune my model running it, ultimately enhancing its performance. More specifically, ran optuna 3 times (for about 100 iterations), each time reducing the search range in order to find the hyperparameters that give the best results.
In the optimization function we tune:

1. **Number of stacked RNNs**
2. **Number of hidden layers**
3. **Type of cells**
4. **Skip connections**
5. **Gradient clipping**
6. **Dropout probability**
7. **Learning rate**

3.3. Evaluation

I am evaluating the predictions using 3 curves:

- **ROC curve**
  The ROC curve is a graphical representation of a binary classification model's ability to distinguish between positive and negative classes.
- **Loss learning**
  Loss is the difference between the predicted values of a model and the actual ground truth labels, guiding the optimization process to minimize this discrepancy and improve the model's performance.

- **Learning rate**
  A learning curve visualizes how a model's performance improves with more training data using the fscore. Fscore is a performance metric that combines precision and recall into a single value.

I also provide the precision & recall scores. The most important thing I'll be taking in mind though, since neither precision nor recall seem to be our priority, is the fscore. It provides a balance between these 2 and makes the model more "generic".

3.3.1. Curves. Here are the curves of the trials showed above:
You have to zoom in order to see the results clearly.

At first, we will be testing models using a custom number of layers and neurons per layer (no optimization).

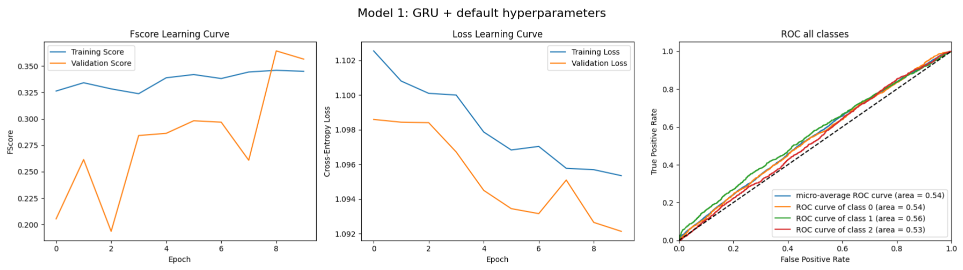

Figure 3: Model 1: GRU + default hyperparameters

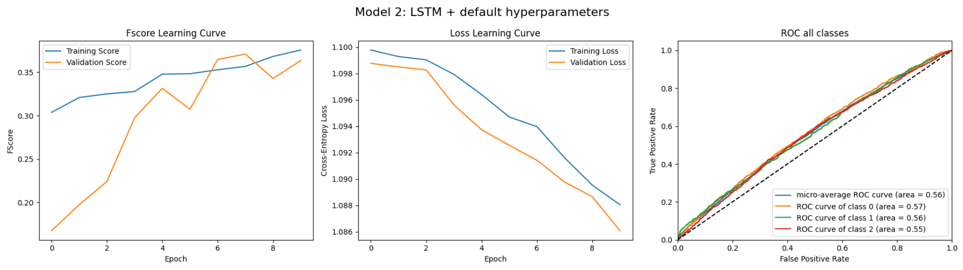

Figure 4: Model 2: LSTM + default hyperparameters

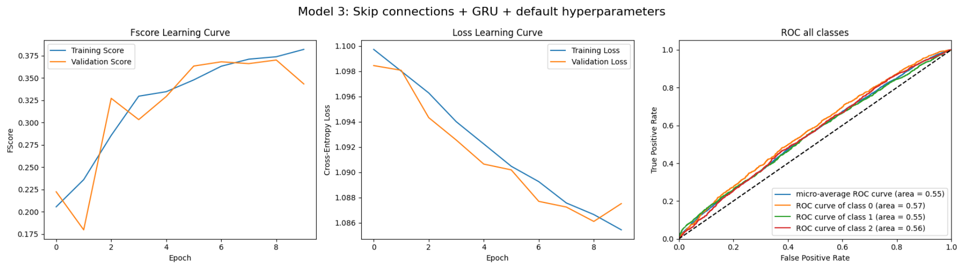

Figure 5: Model 3: skip connections + GRU + default hyperparameters

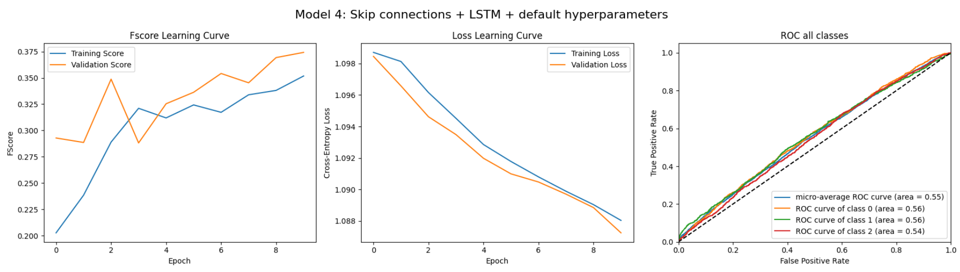

Figure 6: Model 4: skip connections + LSTM + default hyperparameters

Now we will be running Optuna to hopefully optimize our model.

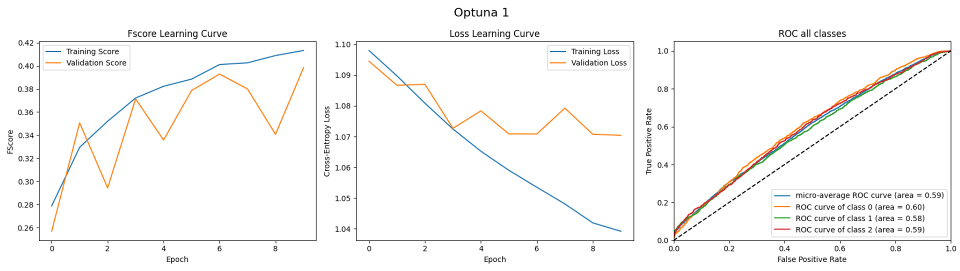

Figure 7: Optuna 1

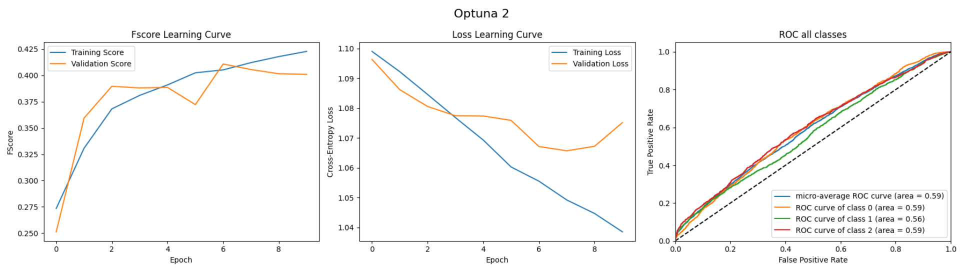

Figure 8: Optuna 2

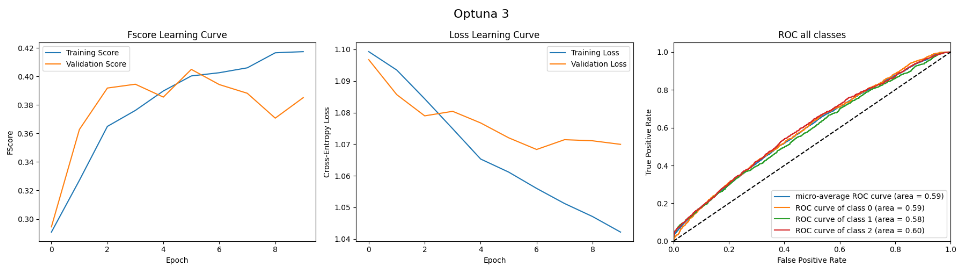

Figure 9: Optuna 3

3.4. Attention

Now we'll try to add attention to the model. What happens is that we now no longer use only the output of the last rightmost hidden state of the top layer but the output of the entire top layer, from all hidden states.

The architecture of the model, implemented in the **AttentionRNN** class, is as follows:

1. Non-trainable embedding layer as the first input layer
2. Bidirectional stacked RNN with either LSTM or GRU cells
3. A linear layer (attention layer), whose trainable weights are multiplied by the output of the RNN, resulting in the attention weights (one weight per hidden state) as output.
4. The output of the attention layer passes through a tanh activation, and then through a softmax, so that the attention weights sum up to 1.
5. Attention weights are (element-wise) multiplied with the output vectors of the hidden states of the last layer of the RNN, and the resulting vectors are added.
6. A linear output layer that generates the scores for each of the 3 classes.

Unfortunately though, adding attention to the final model did not seem to yield better results. The score was about the same and we can (below) see slight overfit.

4. Results and Overall Analysis

4.1. Results Analysis

Overall, although the performance of the model fell short of my expectations, there are several factors contributing to this outcome:

- The dataset presented inherent challenges due to its Greek language content. Greek, being linguistically complex and distinct from widely-used NLP languages, posed inherent difficulties for the model. Also, the limited availability of good performing Greek NLP libraries posed a significant challenge.
- The dataset itself may have contributed to the suboptimal performance. A good number of labels were not correct thus hindering the ability of the model to learn.

4.1.1. Best trial. Ultimately my best trial came after running hypertuning optimization. The model uses 3 stacked RNNs and 29 hidden layers.

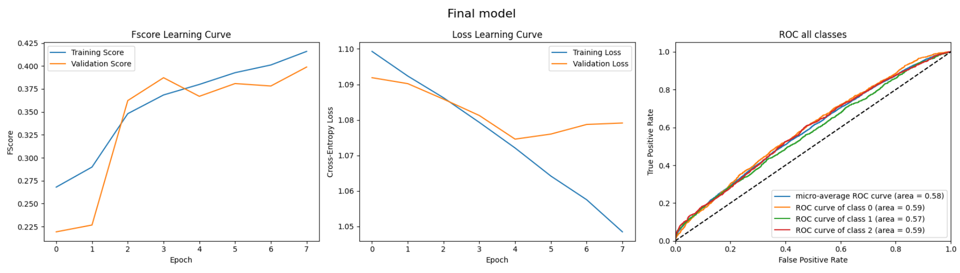

Figure 10: Final Model

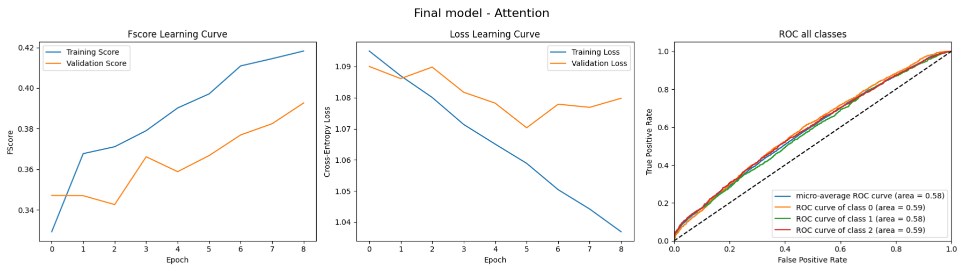

Figure 11: Final Model + Attention

4.2. Comparison with Softmax Regression

Overall, in comparison with the Softmax Regression, our model has about the same performance, maybe slightly better.

4.3. Comparison with Feed Forward Neural Network

Overall, in comparison with the Feed Forward Neural Network, our model performs better (roughly by 1%).

5. Bibliography

References

[1] Optuna. Optuna hyperparameter tuning. https://optuna.org/#code_pytorch.

[2] Stack Overflow. Stack overflow q&a. https://stackoverflow.com/.

[3] Pytorch. Pytorch tutorial. https://pytorch.org/tutorials/.

[4] Stanford. Speech and language processing. https://web.stanford.edu/~jurafsky/slp3/.

[4] [2] [3] [1]