# Natural Language Processing Assignment 2: Fact-checking Using Deep Learning Architectures

Ramez Nafeh, Rasha Jabbour, Jasmine El Afyouni

15/05/2022

## Abstract

The aim of this assignment is to implement and experiment with several deep learning architectures to tackle the fact-checking task. The Keras API was used for pre-processing as well as the building and training of the models. In this report we will discuss our findings as well as what we considered to be our best models, after a number of combinations.

## Dataset Pre-processing

### Text Cleaning

Since the dataset contains many special and non english characters, some cleaning needed to be done. (i) For what concerns **diacritics** and other **non-english characters** (such as in 'Krabbé', 'Königgrätz', etc.), we decided to keep them since they need to be matched against the evidence by the network and slightly improved performance at negligible cost. (ii) However we discarded the **Phonetic transcriptions** such as "tɔma bɑ̃galtɛʁ" and tags such as '-lsb-' as they do not help with the classification. (iii) We experimented with **lemmatization** but achieved minimal performance (<0.5% F1-score and val_accuracy) compared to its high cost, therefore we didn't use it. (iv) **Stopword removal** actually hurt performance badly (>5\% val_accuracy) even when keeping verbs, and that might be because fact checking requires a high level of precision. We evidently didn't use it. We used post-padding of sentences based on the 99th percentile of sentence length.

### Word Embeddings

For the word embeddings, we decided to leave it to the model to learn them since using pre-trained embeddings such as Glove would have resulted in too many OOV words (>50\%).

## Models

A example of one of the model architectures is Figure 01. Models were implemented using the Keras Functional API which makes it easier to work with multiple inputs (Claim and Evidence sentences). Two branches are created, one for Claims and another for Evidences and each consists of:

1. Input layer followed by a trainable Embedding layer
2. The sentence embedding strategy: an LSTM layer was used to output the sentence embeddings (for the first 2 strategies)
3. The input merging strategy using basic Keras layers such as Average and Add. Where cosine similarity was appended to the input, it was calculated using a Keras Dot layer with normalize = True and on the embedding_dim axis
4. Output layer: a single unit Dense layer with a sigmoid activation

The models were compiled using an 'adam' optimizer and a 'binary crossentropy' loss.

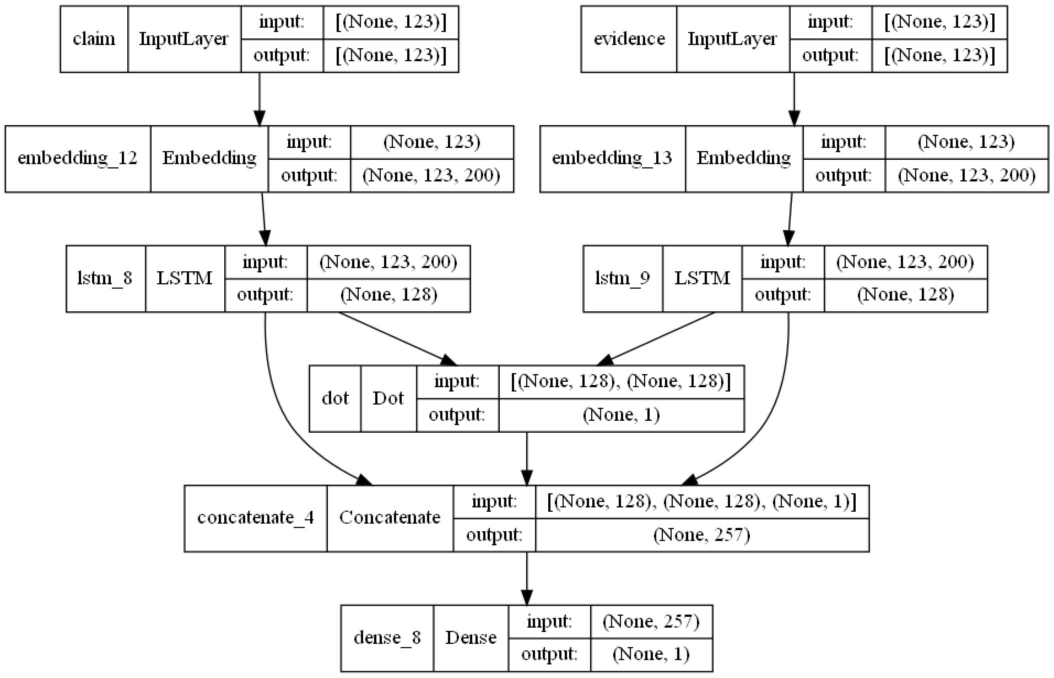

claim | InputLayer | input: [(None, 123)] | output: [(None, 123)]
evidence | InputLayer | input: [(None, 123)] | output: [(None, 123)]
embedding_12 | Embedding | input: (None, 123) | output: (None, 123, 200)
embedding_13 | Embedding | input: (None, 123) | output: (None, 123, 200)
lstm_8 | LSTM | input: (None, 123, 200) | output: (None, 128)
lstm_9 | LSTM | input: (None, 123, 200) | output: (None, 128)
dot | Dot | input: [(None, 128), (None, 128)] | output: (None, 1)
concatenate_4 | Concatenate | input: [(None, 128), (None, 128), (None, 1)] | output: (None, 257)
dense_8 | Dense | input: (None, 257) | output: (None, 1)

*Figure 01: Example of the model architecture using an LSTM for sentence embeddings and using input concatenation*

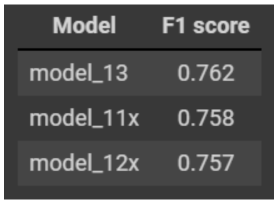
## Results

Models were trained with a batch size of 1024 with Early stopping with a patience of 3 epochs. Tables and detailed results are in the python notebook however this is a summary: our findings showed that using the last state of an RNN, LSTM in our case, gave the best results for sentence embeddings. Out of the 3 input merging strategies, mean of the inputs performed the best, and cosine similarity as an added input slightly improved performance (~0.5%) in two cases or slightly hurt it (~0.5%) in one case which can probably be attributed to randomness.

## Discussion

From the results we see that the F1 score was consistenlty better in all models for the SUPPORT class than for the REFUTE class, and that may be due to the class imbalance in the training set that favours the SUPPORT class. Moreover, while 8.7% of the claims had multiple evidences, given by:

```
(total nb of test claims - nb of duplicate test claims) / total nb of test claims
```



| Model | F1 score |
|---|---|
| model_13 | 0.762 |
| model_11x | 0.758 |
| model_12x | 0.757 |

*Figure 02: Results on test set*

The results barely varied between the 1st and 2nd evaluation type (w/o vs with majority voting), precisely by a maximum of 0.3%.

An improvement to this model architecture could be to use Attention layers.