ASSIGNMENT 2 (Fact Checking)

Davide Mercanti (davide.mercanti@studio.unibo.it)
Riccardo Fava (riccardo.fava6@studio.unibo.it)
Luca Bompani (luca.bompani4@studio.unibo.it)

0.1 Abstract

In this work we have tried several different model architectures to address the Fact checking task. First of all we applied a pre-processing phase cleaning our dataset. Then, starting from the baseline, we modified the classification head, replaced the Bi-RNNs with a Bi-GRU and performed hyperparameters tuning on our models. According to the results, the best model seemed to be the one based on the mean of the Bi-GRU outputs for the sentence embedding part and a double FC layer with a skip connection (MRNN).

0.2 General setting and data handling

Our task consists in performing fact-checking using neural architectures. We decided (and later regretted) to rely on the Pytorch framework for the implementation of the neural networks and to use the Pandas library to handle the data. First of all, we preprocessed our dataset applying cleaning operations to remove stopwords and various artefacts.
The dataset is saved in the form of a pandas Dataframes and encoded using a learned FastText embedding model. The embedding model is generated over the given training corpus by having the FastText model learn to predict a target word by looking at its nearby words. For handling the out of vocabulary terms (OOV) we kept relying on the FastText utility as it allows us to retrieve the OOV word embedding based on the similarity between character-level n-grams.
We applied padding to have sentences of the same length in each batch and we left its embedding to 0 to avoid tampering with the other embeddings.
To emulate a real-world scenario we have computed the test-only-related embeddings and added them to the vocabulary at a different time step concerning the training-related embeddings. The two labels are trivially one-hot encoded. To determine the best models we have tried varying the different hyperparameters of models.

0.3 More about the models used

The models are composed of two parts: a sentence embedding part and a classification head. The classification head is common for all the models and it consists of 2 interconnected dense layers with a skip connection to bypass the first one.
We tried different sentence embedding strategies:
- a Bi-GRU layer taking the last state as the sentence embedding,
- a Bi-GRU layer taking the mean of all the states as the sentence embedding,
- an MLP,
- the mean of all the word embeddings.

Since our task requires to embed two sentences (i.e. the claim and the evidence), once we had the two sentences embeddings (computed as above), we applied these methods to merge them: concatenation, sum and averaging.

0.4 Experiments

First of all, we made some experiments by defining a base model in which the classification head has been implemented as a single dense layer. Having noticed that this model was performing poorly, we decided to improve performance by modifying the classification head. First, we inserted dropout to reduce the discrepancy between the training and the validation performances. Furthermore, having observed that

adding a second dense layer decreased performances, we tried to introduce a skip connection to pass through some lower-level information to the last dense layer.
To increment the quality of the information given to the model, besides applying different forms of sentence cleaning, we tried to increase the size of the word embedding from 100 up to 300. After some trials, we noticed that the best word embedding dimension is 200.

Other tests we performed were:
- replacing the bidirectional RNN with a BiLSTM model;
- replacing the bidirectional RNN with a BiGRU model;
- replacing the final dense layer with a smooth-size-decreasing pool of dense layers.

Only the experiment concerning the BiGRU showed an increase in terms of performances, so we decided to maintain this modification in our architectures. The training dataset is greatly unbalanced. To mitigate the effects of the unbalancing we tried 2 different ways of balancing the data: evening out the classes presented in the batch by introducing a weighted sampler based on the distribution of the classes, and weighting the loss. It seems that these two strategies give comparable results (the second one leads to slightly better accuracy, while the first one provides a smoother descent of the loss).
To optimize our solutions we made different trials varying the hyperparameters obtained by tuning the different networks on F1-score and accuracy. The hyperparameters used in this process were the size of the hidden layers (64 - 256), the learning rate (0.1 - 1e-5) and, separately, the batch size (64-256).
For all the experiments we have utilized the ADAM optimizer with a StepLR scheduler.
Last, to avoid overfitting, we have also introduced an early stopping procedure based on the loss of the validation set.

0.5 Results

The following results have been obtained by training our neural networks for twenty epochs (or less, in the case of early stopping) on the best configuration of the hyperparameters on the validation set.
We notice that with the cosine similarity extension we obtain slightly better results. So, for all our best models, we decided to keep this extension.

Results on the **validation set** are the following:

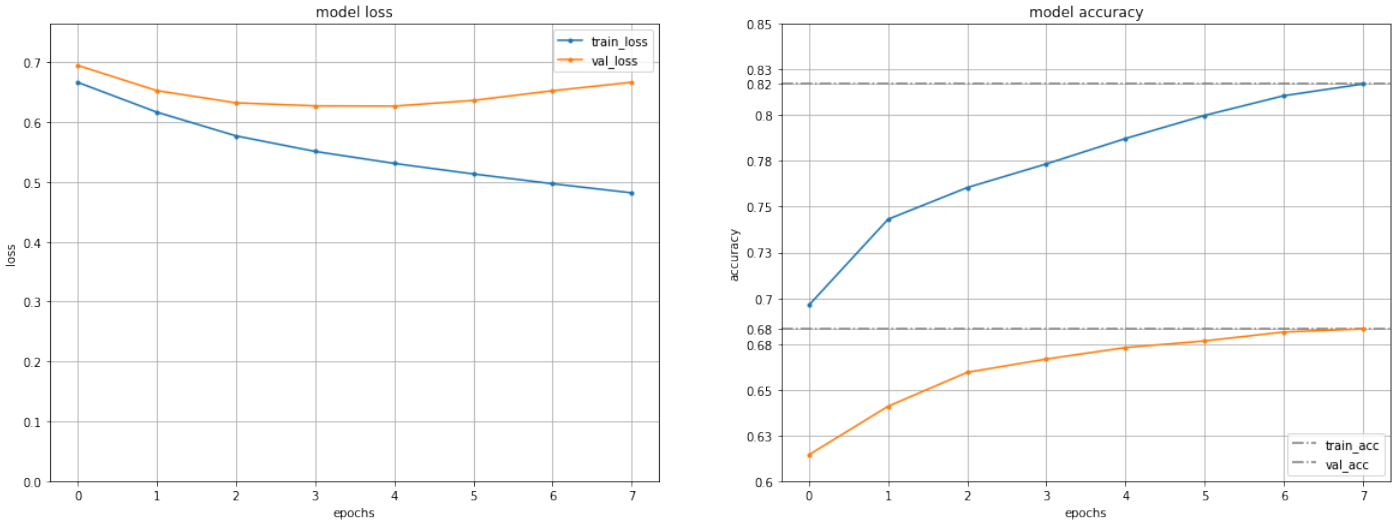
	Accuracy	F1-score	Recall	Precision
LRNN Mean	0.6715	0.6708	0.6712	0.6704
MRNN Concat	0.6831	0.6824	0.6828	0.683
MLP Mean	0.6540	0.6441	0.6527	0.6613
ME Concat	0.6693	0.6680	0.6689	0.6665

For each architecture, we insert in the table only the model with the best performances among the different merging modes. LRNN="last layer of RNN"; MRNN="mean of RNN layers"; MLP="multi layer Perceptron"; ME="Mean of Embeddings"

According to a comparison made on the validation set, the best architecture turned out to be the MRNN model with the concatenation merging mode.

We applied the final evaluation on this model, particularly, its results on the **test set** are the following:

	Accuracy	F1-score	Recall	Precision
Multi-input classification evaluation	0.6630	0.6601	0.6628	0.6685
Claim verification evaluation	0.6599	0.6624	0.6651	0.6706



MRNN Concat Training

0.6 Error analysis

To analyze the predictions that our best models make, we printed some misclassification errors. The main prediction errors that our networks generate are:
- Due to the absence of meaning-bearing words in common between the evidence and the claim. For example the name of the person to which the statement refers to or in some cases the lack of background knowledge of the NN. For example, we all know that a fisherman works with a net but the concept may not be understood by the network. This is especially true in the case of polysemous words, whose embeddings may be far from each other due to the shift brought by the other meanings.
- Some Evidence-Claim pairs are really difficult. For example, the claim uses completely different words to express the same concept of the evidence; this does not allow the network to make correct predictions.
- There are some bad samples in the dataset, some have no direct information in the Evidence about the Claim.

0.7 General conclusions or what we would have liked to do

As mentioned before, the architectures that have reached the best performances are the: MRNN model with the concatenation and LRNN with the concatenation.
For what concerns the MRNN, we can hypothesize that the mean over all the states of the RNN output allows it to recover more contextual information from the sentence than the encoding generated by the last layer of the RNN, which still outperformed the other architectures.

Possible **extensions** of this work could be:
1) The implementation of a custom metric to take into account a more real-world scenario in which false positives and false negatives have different costs.
2) A better handling of the input; in particular a more clever use of the labels at the end of each evidence, that being substantially different in structure from the preceding sentence and deficient of positional information, could be directly embedded (per label) and brought forward to the classification head instead of being included into the sentence embedding module.
3) Feeding the network with handcrafted features that we know are useful to better perform our task.