NLP Assignment 2

Niccolai Lorenzo
Pozzoli Davide
Vizzuso Simone

GitHub:
https://github.com/AirNicco8/NLP_Assignments

Assignment 2 report for
Natural Language Processing

Artificial Intelligence
University of Bologna
Italy

Contents

1 Assignment 2: Fake Checking

1.1 Abstract

We tackled the problem of Fact Checking using the FEVER dataset. We decided to create a Model that would take as input two embedding vectors, one for the claim and one for the corresponding evidence, merge them together and output the result (support, refutes) in one-hot encoding format. We tried different models suggested, but the one that produced the better results was the MLP. It consists in a dense layer applied before the merging of the two vector and after a reshape layer. On this model we tried the three merging strategies and evaluate the results with the Multi-input classification and the Claim verification evaluation.

1.2 Models Description

The models have been created using the `Keras Functional` paradigm, which allowed us to concatenate the layers by calling them as function on a given input, this choice was mainly due to the need of multi-inputs. The structure of the models are:

- **Input layers**: one for Claims and one for Evidences, with the shape of a sequence of the maximum length possible in each input category.
- **Embedding layers**: one for Claims and one for Evidences - which maps into GloVe with 100 dimensions. They are `trainable` to adapt during the training and provide even better embeddings.
- **Sentence Embedding part of the network**, which can consist of :
  1. `LSTM` layers to embed both the claim and the evidence in a tensor of `embedded_dimension`, with 2 variants: `output only the last state` or `output the mean of all the states`.
  2. `Multi Layer Perceptron` in which we first reshape the 3D input tensor from '[batch_size, max_tokens, embedding_dim]' to '[batch_size, max_tokens * embedding_dim]' and then apply to both branches a dense layer.
  3. `Average Bag-of-words` where first we apply the mean of its token embeddings and then we implement a dense layer to both branches before merging.
- **Input Merging layer**: we chose the Mean as baseline, so this is executed by the `tf.keras.layers.Average` layer that computes an element-wise arithmetic average on a list of tensors (in our case the embedded inputs). But for the best performing Sentence Embedding we tried also the other techniques proposed: `Sum` and `Concatenation`.

All the models derived from the baseline have also the *Cosine Similarity* extension, which is done by the `keras.layers.Dot(axes=1, normalize=True)` called on the 2 embedded sentences and the concatenated to the merged inputs.

- **Dropout layer** with a Dropout parameter of 0.25 to avoid overfitting.
- **Final part**: 1 `Dense` layer of size 2 with a `softmax` activation for the final output bit.

The model is compiled using `BinaryCrossentropy` as loss and `Adam` as optimizer. Then the training is done for 10 epochs on a batch size of 1048 for the RNNs, 512 for the MLP and 128 for the Bag of words. The metrics gathered during training are:

- **accuracy** which is the basic metric for machine learning, naively computes the ratio of predictions equal to targets and total predictions
- **f1 score** which is the main evaluation metric, taking into account precision and recall

1.3 Tables and Findings

Table with training and validation metrics of the four sentence embedding strategies, all use **Mean** as input merging technique:

Sentence Embedding Type	Training Accuracy	Validation Accuracy	Validation F1
Last RNN State	0.813	0.644	0.596
RNN States Mean	0.879	0.725	0.721
MLP	0.907	0.743	0.738
Average Bag-of-words	0.890	0.708	0.695

In the next table we report the results for the different input merging applied on the MLP with baseline, this comprehends the Cosine Similarity extension:

Input Merging	Training Accuracy	Validation Accuracy	Validation F1
Sum	0.734	0.504	0.335
Concatenate	0.734	0.504	0.335

Finally, a table which lists the evaluation of the models on the test set (for MLP is used the best variant, Mean):

Model	Test Accuracy	Test macro-F1
Last RNN State	0.715	0.710
RNN States Mean	0.715	0.710
MLP Mean	0.727	0.720
Average Bag-of-words	0.711	0.670

1.4 Error Analysis

We noticed that even if the two RNN models reach similar results in comparison to MLP and BoW ones (especially the states mean), in order to reach such performance more training is needed in terms of time. This may be due to the nature of RNN layers which take some effort in terms of computing power.

Another thing we observed is that both the variations of input merging techniques on the MLP got stuck in a specific minima during training (they yielded identical metrics). This is probably caused by the optimizer which may take longer to achieve a better loss.

A big part of the poor results on validation is caused by the enormous imbalance in the training set, in fact as can be seen in Figure 1 the proportion of *supports* tag is over 80%.

The distribution was sensitively different in both validation and test (Figure 2) so the models struggled to generalize the training on different data.

The evaluation of the models confirms the validation metrics, in fact both accuracy and F1 are coherent with them. From the `classification report` we see that for every model the performance is strongly better on the `SUPPORTS` labels, which again give credit to the fact that the imbalance in the training set conditions the predictions.

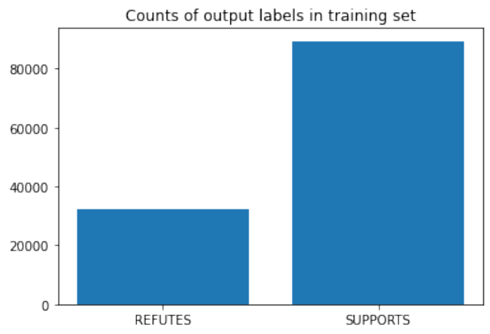

Figure 1: Training imbalance

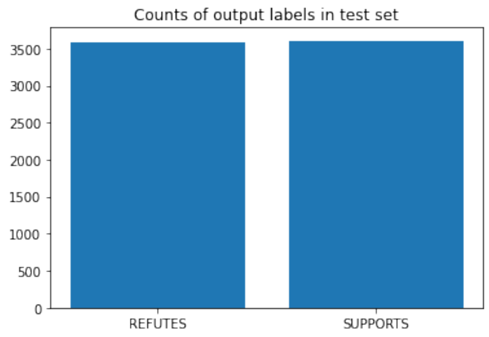

Figure 2: Test set is nearly divided equally