NLP Assignment 1

Niccolai Lorenzo
Pozzoli Davide
Vizzuso Simone

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

Assignment 1 report for
Natural Language Processing

Artificial Intelligence
University of Bologna
Italy

Contents

1 Assignment 1: POS Tagging

1.1 Abstract

We tackled the POS-tagging with neural networks, working on a limited amount of data and trying different kind of structures. We chose to output a one-hot encoding of the class and used a single fixed matrix for the embedding layer. The OOV words have been mapped into a random vector, then we trained and conducted a validation step after each epoch. After this we evaluated the model on training data and looked at the outputs.

1.2 Task Description

This assignment task consists in Part Of Speech (POS) Tagging, which is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and its context. Some examples of part of speech are: noun, verb, article, adjective. Different POS are identified by Tags e.g. CC means coordinating conjunction (and, or). The corpus used in this assignment is the *treebank* corpus, a dataset composed by 200 documents made of a different number of sentences, separated by a full stop.

We decided to divide the documents in sentences after the required splitting and to output a one-hot encoding representing the tag. So subsequent to splitting in train, validation and dataset we tackled the embedding, to do so after tokenizing the sentences of each split we have obtained the differences between the 3 vocabularies and we have concatenated them.

After this we initialized an `embedding_matrix` which maps each word index to its GloVe embedding (100 dimensions). In this step we had to consider what to do with the words that were not present in the downloaded embeddings: we chose to map them randomly. This matrix is used in the first layer of the network.

We padded the sentences to the length of the longest one present in the train split. Then we calculated `f1_macro` on data to evaluate the different models without considering punctuation and padding. To do this we implemented a custom callback from scratch which computes the said metric on the classes we wanted and passed it to the `model.compile`. We also used an `early_stopping callback` which monitored `val_loss` to prevent *overfitting* during the training.

Then to evaluate the model we had to obtain a one-hot prediction of the class, because the last layer outputs a continuous vector: to do this we took the `argmax` of the output vector. We used the `sklearn` built-it function to calculate the `f1_macro` score considering only the allowed tags.

1.3 Models Description

The models have been created using the `Keras Sequential` class, which allowed to add the layers sequentially to the model. The fixed parts are:

- **Input layer**: `InputLayer(input_shape=(MAX_LENGTH, ))` is the input layer of the neural network with shape of a sequence of the maximum length possible
- **Embedding layer**: `embedding_layer = Embedding(num_tokens, embedding_dim, embeddings_initializer=keras.initializers.Constant (embedding_matrix), trainable=False)` is the embedding layer which maps into GloVe with 100 dimensions, it is freezed and immutable
- **Variable part of the network**: which can consist of :
  1. 1 or 2 `Bi-LSTM` layer(s) of size 256 and 128 respectively
  2. a `GRU` layer of 256 units
- **Final part**: 1 or 2 `Time Distributed Dense` layers, the last one must have `number of possible tags` size with a `softmax` activation for the final distribution

The model is compiled using `categorical_crossentropy` as loss and `Adam(0.001)` as optimizer. Then the training is done for 40 epochs on a batch size of 128. The metrics gathered on the model are:

- `accuracy` which is the basic metric for machine learning, naively computes the ratio of predictions equal to targets and total predictions
- `f1` the `macro_f1` score, which as stated by the sklearn documentation is the unweighted mean of the `f1_scores` calculated on each possible output class, in our case the tags
- `f1_i` in addition we also output the components of the macro to see if there is some unbalance between the mean components

1.4 Tables and Findings

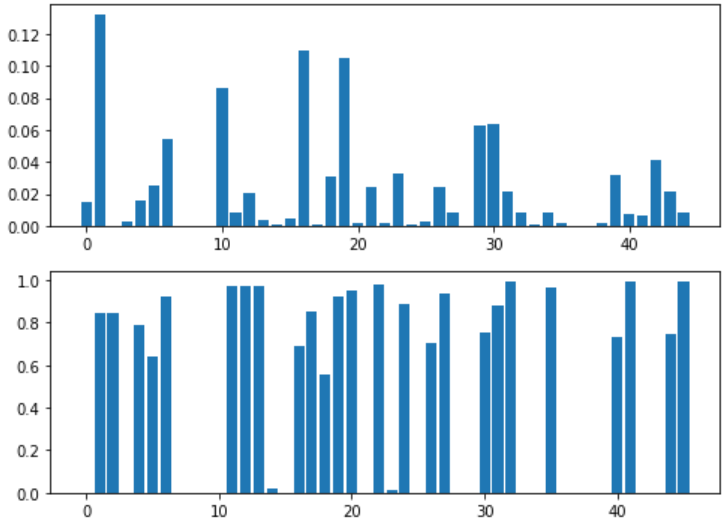
Table with training and validation metrics of the four models:

Model Variable Part	Training Accuracy	Validation Accuracy	Validation F1_Macro
LSTM, Dense	0.992	0.988	0.570
2 LSTM, Dense	0.985	0.983	0.417
LSTM,2 Dense	0.990	0.987	0.504
GRU, Dense	0.989	0.985	0.514

From these findings we chose the 2 models to conduct testing on: the `LSTM, Dense` (which was also the baseline model) and the `GRU, Dense`.

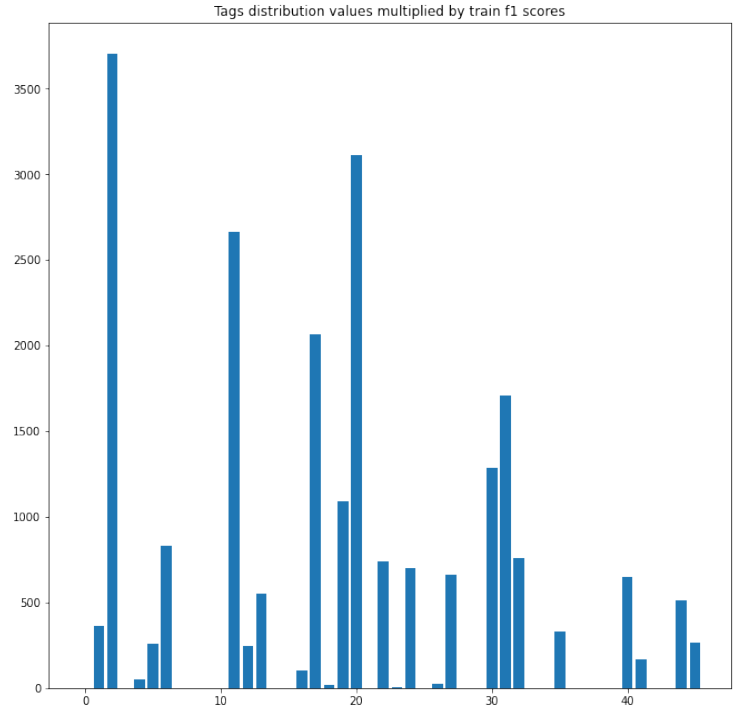
In Figure 1 we can see from the histograms comparison an early impression of the model behavior on different classes (tags), in fact we see that some classes have a near 0 occurrence probability in the distribution and thus the prediction quality is worse than others.

Figure 1: Comparison between tags distribution in validation split and f1 score obtained by the model on validation for each tag



In Figure 2 the histogram plots the f1 score on each class multiplied by the occurrences of such tag in the validation set. This boosts the bars of high f1 score and high number of occurrences and shrinks low f1 and low occurrences tags.

Figure 2: Multiplication of f1 on tags and their count in validation set



1.5 Error Analysis

In conclusion we list here the `f1_macro` score on test set of each of the 2 models:

1 BiLSTM and 1 Dense this model obtains a score of 0.630

1 GRU and 1 Dense this model obtains a score of 0.577

The results are coherent to the findings on the validation set, anyway the small quantity of data taken into account make this model analysis quite specific. In other words we are not so sure that the model would generalize perfectly to real world data to predict.

Talking about error analysis we think that the `f1_macro` is held down from the high number of classes which obtain near 0 `f1` scores. These classes usually occur way less often than those which have higher scores, but are weighted the same in the macro computation.

In conclusion to improve the model and the results a bigger dataset with more variability could be used. Of course also an augmentation in the number of layers and/or their size could do something.