# Assignment 1

**Davide Brescia, Daniele Marini** and **Iulian Zorila**
Master's Degree in Artificial Intelligence, University of Bologna
{ davide.brescia, iulian.zorila, daniele.marini3 }@studio.unibo.it

## Abstract

The process of POS Tagging presents various challanges, in this paper we will look at different models and analyze the most common missclassifications, specifically, sentences which the models have most difficulty to deal with, and the extent to which the composition of the dataset affected performance.

## 1 Introduction

POS Tagging is an operation in which each part of the text is assigned a label representing its grammatical component within the sentence. Our goal is to find a good model that can perform POS tagging automatically.

After a brief analysis of the data we noticed how the dataset was considerably unbalanced towards certain tags, reflecting the written language. Proceeding we implemented word embeddings by making the model learn the word-tag associations within the sentence. Initially we tested the effectiveness of the baseline model (a model containing the embedding, LSTM and fully connected layers). Then, by the means of a Grid Search method we compared different architectures with various combinations involving the layers units, finding the two best models. Lastly we implemented an evaluation process by obtaining some information regarding issues and strengths of both models.

## 2 System description

We can summarize our work in four main areas:

**Import, preprocessing and exploration of the dataset**: to import the dataset, we used the `download_dataset` and `extract_dataset` functions provided to us in Lab 1. Furthermore we used the `set_reproducibility` function from Lab 2 to have consistent results across experiments. As the only preprocessing element, we transformed the words entirely into lowercases and then visualized the distribution of tags noting a significant disparity between the different tags.

**Embedding Phase**: to obtain the embeddings corresponding to each term, we relied on the `KerasTokenizer` class provided in the Lab 2, to get the pretrained Glove embedding, check the OOV words (4.85%) and build the embedding matrix.

**Find the Best Model**: we applied a Grid Search by comparing multiple models with each other to better explore the space of possible solutions. The overall search took around an hour, using the Colab GPU, returning `variation_lstm_128_dense_2048` and `variation_lstm_32_dense_2048` as the best models.

**Evaluation**: at this stage, the behavior of the two selected models in performing the POS tagging task was analysed in detail. In particular, we analysed the final scores of the models and compared the scores across tags. We also visualized the confusion matrix and then moved on to sentence analysis. Specifically, we did experiments on sentence length and the presence or absence of out-of-vocabulary words. Finally, we analysed which tags and terms were the most misclassified.

## 3 Experimental setup and results

We used same Adam optimiser, early stopping, batch size of 64 and 100 epochs for all the models and among them, we chose the ones that performed best in the test set: `variation_lstm_128_dense_2048`

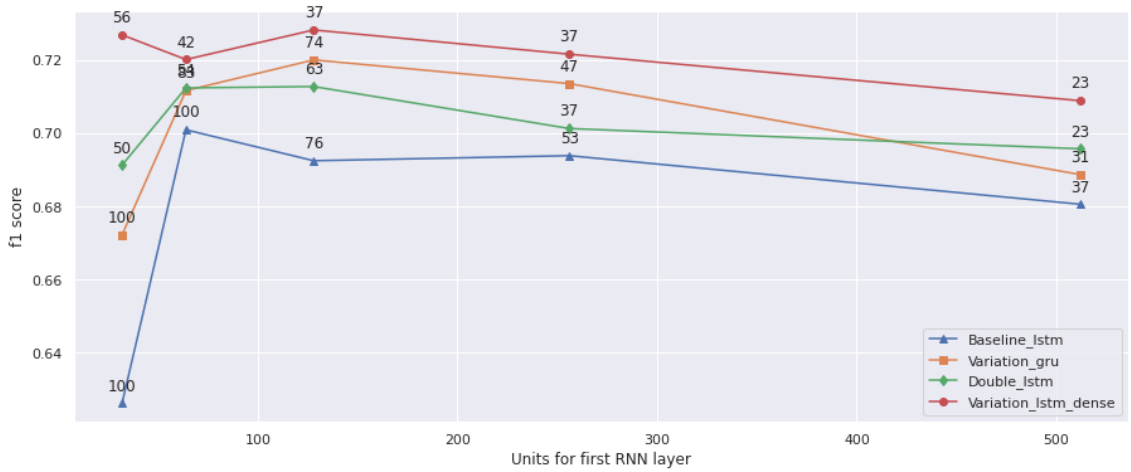


Figure 1: Models comparision with F1 score on test set (the numbers above every curve point represents the number of trained epochs). The number of units combinations to test in the first RNN layer is the same across different models (32, 64, 128, 256, 512).

and `variation_lstm_32_dense_2048` obtaining F1 scores (with `zero_division=0`, since some tags are not present in the test set) of 0.7283 and 0.7269 respectively, while the best baseline model, `baseline_lstm_64` reached 0.7010 (see Figure 1).
We carried out a Grid Search, changing the number of units per layer and testing a total of 60 models: 5 for the baseline, following the variations, 5 for the GRU, 25 for the LSTM + LSTM and 25 for the LSTM + dense. Increasing the model capacity, i.e. number of units, reduces the number of trained epochs, as we can notice from 128 units onwards. Moreover the LSTM + dense architecture has a better performance as the dense layer employes 2048 units, hence it has more expressive power.

## 4 Discussion

The performance of the two models is very similar, generally they obtain good results for the following classes: *in* (Preposition or subordinating conjunction), *dt* (Determiner), *cc* (Coordinating conjunction), *to*, *prp* (Personal pronoun), *md* (Modal), *pos* (Possessive ending), *prp$* (Possessive pronoun), *wdt* (Wh-determiner). While they have below-average results in these classes: *jj* (Adjective), *vbn* (Verb, past participle), *vbg* (Verb, gerund or present participle), *rp* (Particle). It should be emphasized that there are certain classes in the test set in small numbers that affected the performance of the model.

We noticed that models have difficulty classifying the following sentences (or similar ones): "*grains and soybeans:*" and "*dow jones industrials 2645.90, up 0.82; transportation 1206.26, up 1.25; utilities 220.45, up 1.26.*". After some deeper research, we noticed that sometimes models: think they are faced with singular words instead they are plural and fail to detect cardinal numbers.

Understandably, the most missclassified word is Unknown, followed by words that depend especially on the context of the sentence and are frequently used, some of them are: "*up*", "*yen*", "*that*" and "*fiscal*". Delving deeper into this concept, we noticed how the class assignment distribution depends greatly on the training set assignment distrubution. Thus if a word is often associated with a tag and in the test set the latter does not fall into this association, the model is likely to make an error.

Finally, the most common classification errors are: predict *dt* (determiner) instead of *in* (subordinating preposition or conjunction) or viceversa and predict *jj* (Adjective) instead of *in* and viceversa.

## 5 Conclusion

We mainly focused on changing the number of units for each layer with early stopping to observe the F1 score trend among model variations. However one could test different hyperparameters (batch size, optimizer) and callbacks, keeping relatively low the number of units in the RNN layers.