# Assignment 1

**Morotti Daniele, Sciamarelli Andrea** and **Valente Andrea**
Master's Degree in Artificial Intelligence, University of Bologna
{ daniele.morotti, andrea.sciamarelli, andrea.valente6 }@studio.unibo.it

**Abstract**

The objective of this assignment is performing Part Of Speech tagging using some specific neural architectures. The dataset features tags whose number of examples is unbalanced. However, this did not directly correlate to bad performances. We defined the networks using the Keras library on top of Tensorflow. We noticed an interesting phenomenon: it's not necessarily the case that a model learns how to recognize a certain class better if there are more examples available. Some simple considerations have been done to shed light on the reason behind this peculiar observation.

## 1 Introduction

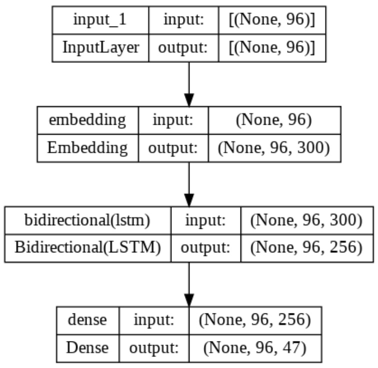
Part Of Speech Tagging is a very well known task in the NLP community, and a lot of different approaches that have been studied in order to tackle it. Researchers have tried probabilistic based, rule based and neural network based supervised and unsupervised methods. Recently, supervised based neural network based solutions have progressively become better and better and nowadays are also the state of the art. Our approach consists of using some specific neural networks, trained on a given dataset. Additionally to the layers we've been requested to use, we inserted some `Dropout` layers (Zaremba et al., 2014) that helped to achieve better performances.

## 2 System description

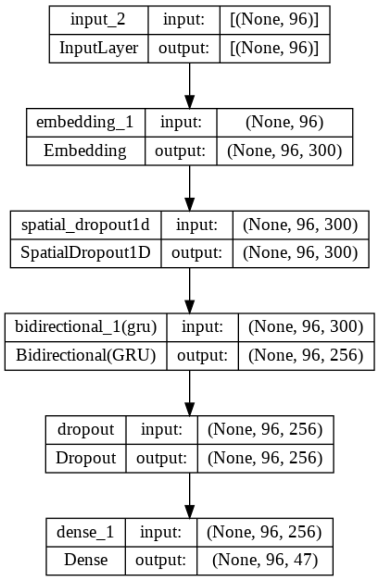
We have implemented four different neural network architectures, very similar in structure. The baseline model is built as follows: at the bottom there's an `Input` Layer, which is a vector of (non-negative) integers of size `MAX_SEQUENCE`, a constant that has been defined equal to 96 and that has been also used in the `EncodeLabels` and `TokenizeFeatures` classes (see section 3); immediately follows a non-trainable `Embedding` layer, initialized with an embedding matrix built on top of the publicly available GloVe embeddings (taken from `https://huggingface.co/`) of dimension 300, with 6 billion tokens, with some additional vectors that have been randomly generated for OOV words, following the provided procedure; on top of it there's a Bidirectional LSTM Layer; on top there's a fully connected `Dense` Layer with a number of units equal to the number of all the possible Part Of Speech tags (we wanted the network to output a probability distribution, reason for which we used the `softmax` activation function). The other three models have been defined starting from the baseline: instead of the Bidirectional LSTM Layer, the next two models feature, respectively, a Bidirectional GRU Layer and two Bidirectional LSTM Layers. The last one, instead, features two Dense Layers instead of a single one (of which the second from the top is a Time-Distributed `Dense` Layer). Furthermore, around the layer that has replaced the Bidirectional LSTM Layer (or the layer itself in the fourth model), we decided to add a `Dropout` and a `SpatialDropout1D` Layer, inspired from a particular piece of literature (Zaremba et al., 2014). The figures 1a, 1b, 2a, 2b provide a pictorial representation of the tested architectures.

## 3 Experimental setup and results

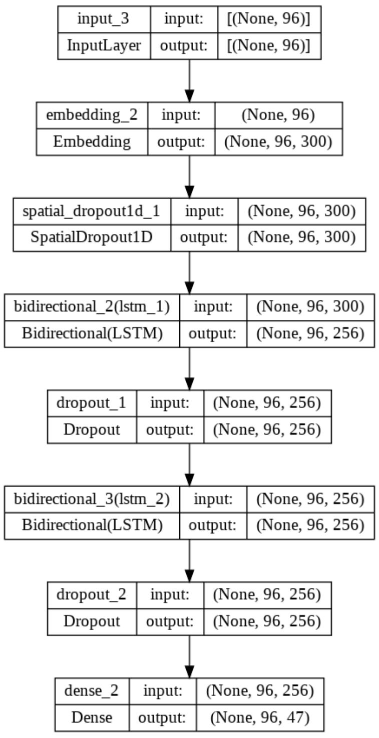
The distribution of tags is not uniform, however this didn't prevent us to train in a satisfactory way the neural architectures we've tried. The preprocessing pipeline can be summed up as follows: we first loaded the data into a Pandas DataFrame and then to a Tensorflow Dataset, useful for applying transformations through the `map` method, intensively used. We defined some specific operations to apply to the data as the classes `EncodeLabels` - which featured a `StringLookup` layer, used to


input_1 | input: | [(None, 96)]
InputLayer | output: | [(None, 96)]
embedding | input: | (None, 96)
Embedding | output: | (None, 96, 300)
bidirectional(lstm) | input: | (None, 96, 300)
Bidirectional(LSTM) | output: | (None, 96, 256)
dense | input: | (None, 96, 256)
Dense | output: | (None, 96, 47)

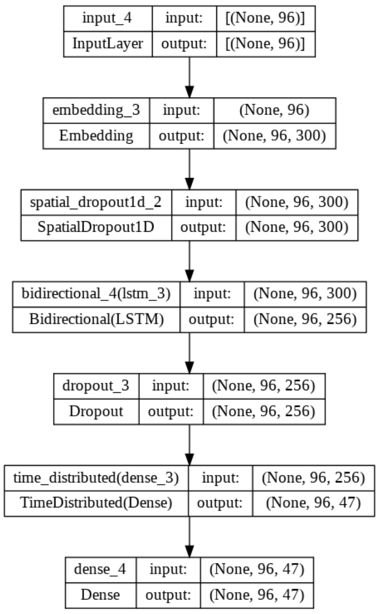
(a) Baseline model architecture


input_2 | input: | [(None, 96)]
InputLayer | output: | [(None, 96)]
embedding_1 | input: | (None, 96)
Embedding | output: | (None, 96, 300)
spatial_dropout1d | input: | (None, 96, 300)
SpatialDropout1D | output: | (None, 96, 300)
bidirectional_1(gru) | input: | (None, 96, 300)
Bidirectional(GRU) | output: | (None, 96, 256)
dropout | input: | (None, 96, 256)
Dropout | output: | (None, 96, 256)
dense_1 | input: | (None, 96, 256)
Dense | output: | (None, 96, 47)

(b) Model with a Bidirectional GRU Layer


input_3 | input: | [(None, 96)]
InputLayer | output: | [(None, 96)]
embedding_2 | input: | (None, 96)
Embedding | output: | (None, 96, 300)
spatial_dropout1d_1 | input: | (None, 96, 300)
SpatialDropout1D | output: | (None, 96, 300)
bidirectional_2(lstm_1) | input: | (None, 96, 300)
Bidirectional(LSTM) | output: | (None, 96, 256)
dropout_1 | input: | (None, 96, 256)
Dropout | output: | (None, 96, 256)
bidirectional_3(lstm_2) | input: | (None, 96, 256)
Bidirectional(LSTM) | output: | (None, 96, 256)
dropout_2 | input: | (None, 96, 256)
Dropout | output: | (None, 96, 256)
dense_2 | input: | (None, 96, 256)
Dense | output: | (None, 96, 47)

(a) Model with two Bidirectional LSTM Layers


input_4 | input: | [(None, 96)]
InputLayer | output: | [(None, 96)]
embedding_3 | input: | (None, 96)
Embedding | output: | (None, 96, 300)
spatial_dropout1d_2 | input: | (None, 96, 300)
SpatialDropout1D | output: | (None, 96, 300)
bidirectional_4(lstm_3) | input: | (None, 96, 300)
Bidirectional(LSTM) | output: | (None, 96, 256)
dropout_3 | input: | (None, 96, 256)
Dropout | output: | (None, 96, 256)
time_distributed(dense_3) | input: | (None, 96, 256)
TimeDistributed(Dense) | output: | (None, 96, 47)
dense_4 | input: | (None, 96, 47)
Dense | output: | (None, 96, 47)

(b) Model with two Dense Layers

map the tags of the sentences to numbers that can be used during the trainings of the tested neural architectures - and `TokenizeFeatures`, which used the `TextVectorization` layer class from `Keras` to vectorize each sentence (i.e. assigning each token to a number and putting those into equally sized vectors) and to generate a vocabulary. After that, we loaded the GloVe embedding (see section 2) and loaded the vectors corresponding to the words in the pretrained embedding. We handled the OOV terms as instructed, and finally built the embedding matrix used in the `Embedding` Layer of the tested neural architectures. The way the architectures were defined didn't leave much hyperparameters to choose: we just had to specify the rates of the `SpatialDropout1D` and of `Dropout` layers and they have been decided differently across models. The activation function of the final `Dense` layer has been chosen to be the usual `softmax` in order to make sure that the output of the networks resembled a probability distribution. Coherently, we used the `SparseCategoricalCrossentropy` as loss function. We also monitored the "accuracy" metric during training, due to the impossibility of aggregating partial f1-scores computed on the batches of data. We chose the Adam algorithm as optimizer, as it is "computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters" (Kingma and Ba, 2014). We tested the network in both the validation and test sets, and we measured the quality of the predictions of the networks by calculating precision, recall, and f1-score for each possible tag. We also measured the overall performances by considering the respective macro averages. By comparing the results for the tags whose support was comparable over validation and test sets, we ascertained that the results were coherent. In the table 1, a summary of our findings can be found. Even though all the networks performed similarly, the last two models were the ones that performed best in the validation set, reason for which we conducted an extensive error analysis only on those, as instructed.

## 4 Discussion

Our experiments confirmed that the majority of the tags are correctly identified by our networks and the last two models perform significantly better than the baseline (see table 2). Due to the unbalanced numbers of examples for each tag, we expected that the tags for which more examples were available should have been be the ones better recognized from the networks. However, this wasn't always the case: we realized that the tag "PRP$" is recognized much better than "NN", the most frequent tag in our dataset. We also noticed that this behavior could be observed regardless of the network, which made us suspect that the causes had to be found in the train set. The tag "PRP$" stands for "Possessive pronoun", whilst the tag "NN" is assigned to "Noun, singular or mass". This gave us a huge hint: there are way more nouns than possessive pronouns, reason for which it's way more likely that the train set contains all the possessive pronouns that the test set contains, rather than the same thing with the nouns. Furthermore, this is way more likely for tags associated to a restricted set of tokens. We were able to empirically confirm our hypothesis by checking that the ratio between unique possessive pronouns which could be found in both train and test set and the unique possessive pronouns which could be found in the test set is 1 (which could explain the good performances for identifying possessive pronouns), while the same ratio calculated for nouns ("NN") is about 0.61.

## 5 Conclusion

All the tested models were able to perform well in our tests. We were surprised to see the apparent discrepancy discussed in the previous paragraph (see section 4), which was, however, ultimately explained - as discussed - by performing a study on the frequencies of the tags in the training set. A possible source of improvement could be the `MAX_SEQUENCE` parameter. We have chosen 96 after an analysis of the training set, but increasing the value to 128 could lead to improvements, at the cost of the necessity of more computational power needed in order to train the networks.

## References

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *CoRR*, abs/1409.2329.

| Macro averages | Val | | | Test | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | F1-score | Precision | Recall | F1-score |
| **BiGRU** | 0.742 | 0.738 | 0.731 | 0.840 | 0.853 | 0.844 |
| **Dual BiLSTM** | 0.779 | 0.763 | 0.761 | 0.867 | 0.846 | 0.841 |
| **Dual Dense** | 0.706 | 0.699 | 0.693 | 0.849 | 0.837 | 0.835 |

Table 1: Performances of the tested networks.

| Macro averages | Val | | | Test | | |
|---|---|---|---|---|---|---|
| | Precision | Recall | F1-score | Precision | Recall | F1-score |
| **Baseline** | 0.694 | 0.674 | 0.674 | 0.819 | 0.807 | 0.806 |
| **BiGRU** | 0.742 | 0.738 | 0.731 | 0.840 | 0.853 | 0.844 |
| **Dual BiLSTM** | 0.779 | 0.763 | 0.761 | 0.867 | 0.846 | 0.841 |

Table 2: Comparison between baseline and the two best models.