Assignment 1

Faharad Bayrami, Gianluca Di Mauro, Kaan Molla and **Leonardo Monti**
Master's Degree in Artificial Intelligence, University of Bologna
{ farhad.bayrami, gianluca.dimauro, mahmutkaan.molla, leonardo.monti3 }@studio.unibo.it

Abstract

Part Of Speech tagging (POS tagging) is the process of assigning a part of speech to each word in a text. Automation of this task has already provided successful results, so that accuracies as great as 97% have been achieved with different techniques (Jurafsky and Martin, 2009). This report presents several experiments on POS tagging using variations of a model based on LSTM neural networks (Hochreiter and Schmidhuber, 1997). The obtained results are presented and compared with the structure of the corpus.

1 Introduction

Several techniques have been developed for the task of POS tagging. Hidden Markov Models follows the sequential nature of text but lacks the possibility of exploiting arbitrary features like the knowledge that an article is never followed by a verb. Conditional Random Fields allow the designer to select the features to use (Jurafsky and Martin, 2009). Several neural approaches exist, where modern Large Language Models play a major role. This report focuses on the Bidirectional LSTM neural approach. Our interest is developing three models based on similar architectures able to perform POS tagging on English sentences taken from the Penn TreeBank corpus (Marcus et al., 1993). The assigned architectures include a baseline based on a Bidirectional LSTM layer and two variations of that. Our approach involves training each model on three different seeds, for robust metrics estimation, then comparing the results to pick the best performing model and observe how each model performs at identifying each part of speech. Operating on a dataset of only 200 sentences, of which only 100 could be used for training, promising results were obtained, also on the less represented classes.

2 System description

The system runs entirely on Python. The neural library of choice is Keras (Chollet et al., 2015). First the dataset is split in to train, test and validation datasets using the data management library Pandas without randomization or stratification (McKinney et al., 2010). As for the text preprocessing we only transformed the sentences to lower case. Then text encoding is applied. GloVe embeddings (Pennington et al., 2014) were used, with an embedding dimension of 100. The tokenizer of choice, is the Keras Tokenizer. This transforms text into padded sequences of embeddings. Moreover, punctuation symbols were included in the tokenization, but a a list of the punctuation tokens was created in order not to consider them when computing the performance metrics in the evaluation phase. The tags were transformed in one-hot encoding for training the models. The first model, baseline (Figure 1), is a simple sequential arhcitecture composed of Bidirectional LSTM layer with a Dense layer on top. The aforementioned two variations of the baseline were implemented by adding respectively an additional BiLSTM layer (Figure 2) and an additional Dense layer to the baseline (Figure 3). All the three models take advantage of Keras' LSTM, Bidirectional, Dense and TimeDistributed layers (Chollet et al., 2015). The standard Scikit-Learn f1_score function (Pedregosa et al., 2011) has been used to calculate metrics, excluding the punctuation and symbols from the computation. Several analysis steps are then performed to check and explain the results obtained.

3 Experimental setup and results

To train the models, Adam optimizer, provided by the Keras library, was the opted choice. Also the loss function is provided by Keras: Categorical-Crossentropy. Some experiments were done using

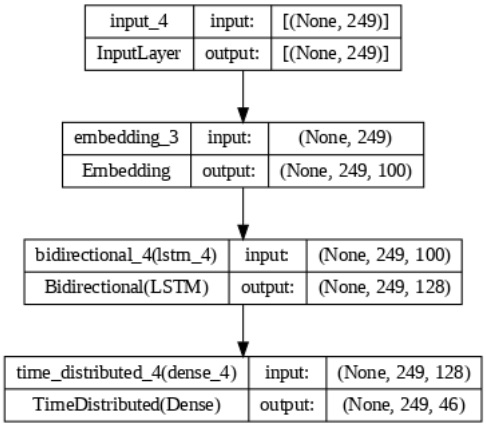

Figure 1: Baseline model architecture.

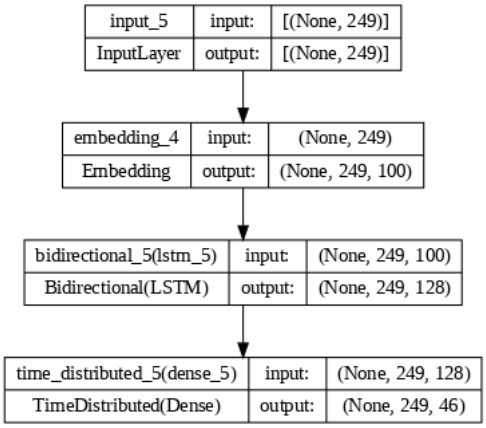

Figure 2: Model 1 architecture.

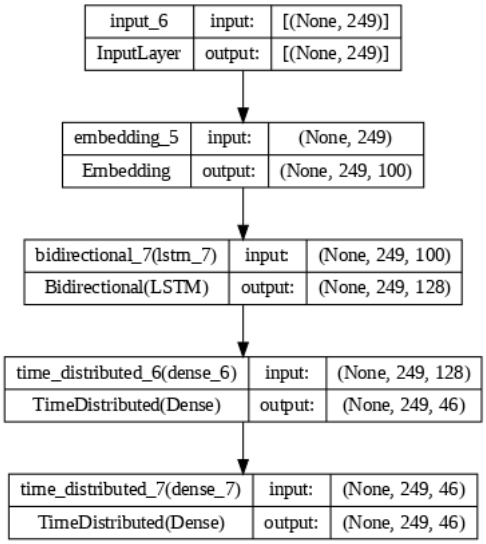

Figure 3: Model 2 architecture.

instead the SparseCategoricalCrossentropy function, with poor results. The CategoricalCrossentropy requires labels to be provided in one_hot representation because it models the classification output as a probability curve (Chollet et al., 2015). Smoothing of the target labels was tested but it did not provide any substantial gain. Initial tests with a learning rate up to 1e-2 were done, which was clearly too high, producing very inaccurate models. Further experiments were done lowering the learning rate and by reducing it during the training process. Keras provides a very convenient feature for automatically reducing the learning rate when a metric of the model stops improving. Also the training can be cut off when no improvement happens. They are ReduceLROnPlateau and EarlyStopping callbacks (Chollet et al., 2015). Many experiments were done to configure the best set of hyperparameters for the training. The validation accuracy was the targeted metric, for which the ReduceLROnPlateau was set to have a patience of seven epochs with no improvement before reducing the learning rate by a factor of ten. After each reduction it cools down for one further epoch to avoid a misleading measurement of the metric. EarlyStopping was set with a patience of 20 epochs of no improvement, to avoid stopping too early. In order to focus only on significant changes the minimum improvement of validation accuracy was set to 0.001 for both the callbacks. This setting proved to be very effective at improving the quality of the result and the speed of the training. The same set of hyperparameters was kept for all of the models, after being determined based on the performance of the baseline model. Table 1 shows the results for starting learning rate of 5e-3 for the baseline and the two other models. Worth of mention is the validation accuracy of the models that exceeded 98% with peaks of 99%.

Model	F1 score for lr=5e-3
Baseline	0.6727
Model1	0.6714
Model2	0.6697

Table 1: Macro F1 Scores of the 3 models

4 Discussion

The accuracies and F1 scores roughly matched the expectations. The models provided similar results for the Macro F1 score. Results showed that

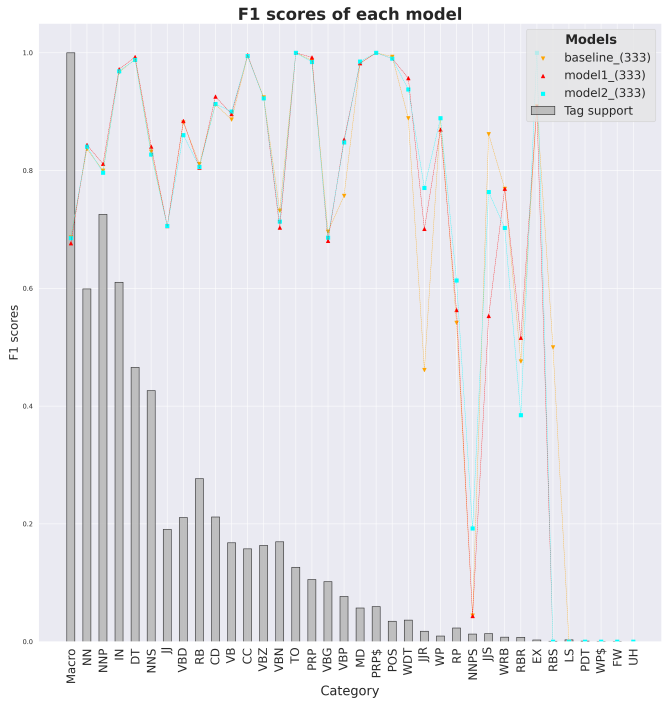

Figure 4: Average F1 scores of Baseline, Model 1 and Model 2 for each tag class, compared with class support.

the baseline model was the best performing one. In Figure 4 the performances of the classifiers with the supports of the class in the training set are compared. All the models show very similar behaviors on each class. It is evident how the performance is very satisfactory for the classes that have a support bigger than a minimum threshold. The performances on the less represented classes vary drastically. The reason for this is the unevenness of the classes support among the train, validation and test data splits. This caused the model to learn how to tag only some of those classes. For example the class 'WP' is more represented in the train and validation splits than in the test split, so the performance on test set is still very good. The presence of classes that are not, or nearly not, represented in the train split reduces drastically the average F1 score.

5 Conclusion

The performed experiment showed that comparable results can be achieved with three different models for POS tagging. The quantitative results showed that the BiLSTM based models can learn to classify correctly also the less represented classes. The baseline model performing marginally better contradicted our expectations. This can be due to the hyperparameters of the training being tuned for the baseline model and applied to the others without change and to the need for less examples to train a smaller model. A possible further improvement is on how the sentences of the corpus where split into train, validation and test. Having a more homogeneous representation of the classes would benefit the average F1 score. Also increasing the size of the corpus, with attention to the less represented classes, would increase the performances noticeably. Finally, for showing the best performance each model can achieve it is possible to customize the hyperparameters for each model, instead of providing a common base.

6 Links to external resources

All the materials of this assignments are available at the following GitHub repository

References

François Chollet et al. 2015. Keras. https://keras.io.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Dan Jurafsky and James H. Martin. 2009. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Pearson Prentice Hall, Upper Saddle River, N.J.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Wes McKinney et al. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.