Assignment 1

Fabio Zanotti, Antonio Morelli, Federico Rullo, and **Edoardo Conca**
Master's Degree in Artificial Intelligence, University of Bologna
{ fabio.zanotti, antonio.morelli, federico.rullo, edoardo.conca }@studio.unibo.it

Abstract

In this project, we present the implementation and evaluation of a Part-Of-Speech (POS) tagging system using neural architectures. The objective is to predict the POS tag for each word in a given corpus of documents. We explore three different neural models: a baseline model with a single Bidirectional Long Short-Term Memory (LSTM) layer, an extended model with an additional Bidirectional LSTM layer, and a variant with an extra Dense layer. The dataset is preprocessed, encoded, and GloVe embeddings are utilized to initialize the model weights, leveraging pre-trained word representations. To address Out-Of-Vocabulary (OOV) words, they are integrated into the vocabulary and their embeddings are computed based on tag-specific means from existing words. The results showed how the deeper models behaved better than the baseline, although the simpler architecture performs comparably to the more complex models, which may suggest that the complexity introduced by the extra layers did not provide substantial benefits for the task.

1 Introduction

The Assignment consisted in implementing a Part Of Speech Tagging system using neural architectures, given a corpus of documents, the objective was to predict the Part Of Speech tag for each word. The system has been implemented using three different models, and comparing them in order to see which one would perform better on the given data. The three different models are:

- Baseline - consisting of a single Bidirectional LSTM layer;
- Baseline with an additional Bidirectional LSTM layer, to further capture contextual dependencies;
- Baseline with an additional Dense layer, to enhance the feature representation;

Bidirectional LSTM layers are able to process sequential data in both forward and backward directions, this allows the model to capture contextual information from both past and future. This is particularly useful for natural language processing tasks as the meaning of words can sometimes depend on the context in which they are used, one advantage of using it for POS tagging is that it allows one to predict the tag for each word in a sentence simultaneously, this is particularly useful when dealing with long sentences.

Our approach was to implement these different models and compare them based on the average Macro-F1 score which is a common metric used to evaluate the performance of models and was also the requested metric.

2 System description

We initially imported the dataset from the Natural Language Toolkit(NLTK) library, which we preprocessed and then encoded. We removed the "-NONE-" tag, this way the model will have fewer examples of unstructured data to learn, and can focus on the examples that are most relevant to the POS tagging increasing the accuracy of the results. After visualizing some plots that gave us an idea of the dataset composition, which seemed to be affected by some inbalance in between classes, we built the vocabulary; for this, we used GloVe (Global Vectors for Word Representation) which is an embedding method for learning vector representations of words based on word-word co-occurrence statistics.

Since there are different versions of GloVe, each offering embeddings with varying dimensions, we settled for the 100-dimensional version, which strikes a balance between computational efficiency and capturing richer semantic relationships compared to smaller dimensions. By using GloVe embedding to set the initial weights of the model we could take advantage of the pre-trained word repre-

sentations and fine-tune them to suit our task.

Because some words may not be present in the GloVe vocabulary, we expanded it by integrating Out-Of-Vocabulary (OOV) words present in the dataset. The treatment of these words is crucial for the proper functioning of the model. Hence, we decided not only to assign a random vector to OOV words, but also to leverage prior knowledge by computing the means per tag of the words already present in the dictionary. This ensures that the final embedding for an OOV word slightly differs from those values yet remains meaningful, enhancing the model's performance.

The vocabulary is then computed by iteratively inserting Out-Of-Vocabulary (OOV) words respectively from the training set, the validation set, and the test set, as suggested by the professor.

Encoded sentences are padded to the length of the second-longest sentence since we noticed that the longest one is an outlier. The newly inserted label is defined as `-PAD-`, following treebank notation.

The next step has been to define the three models for the POS Tagging system. All these models have the same input and output layers; we used GloVe Embedding as an input layer and TimeDistributed as an output layer. The models differ in the hidden layers. The first model, which is the Baseline model, is composed of only one Bidirectional LSTM hidden layer (see Figure 1). The second model is the Baseline model extended with an additional Bidirectional LSTM layer (see Figure 2). The third, and final, model is the Baseline with an additional Dense layer (Figure 3).

3 Experimental setup and results

3.1 Parameter Tuning

Before conducting experiments on the data, we performed hyperparameter tuning for each model. During the tuning phase, we observed significant changes in scores due to the units used in the LSTM layer and the batch size. Typically, a higher number of units in the LSTM layer increases the model's capacity for complex representations. However, excessive units may lead to overfitting. We found that setting the number of units between 128 and 256 helps the model memorize important features of the data while maintaining generalization.

For the batch size, we opted for a value of 32. A smaller batch size improves performance and alleviates issues related to high variance in the estimated mean. Additionally, a smaller batch size allows the model to observe a more diverse range of samples in each batch, thereby mitigating class imbalance. We used the same numbers of units and batch size for all models to ensure consistency.

We also employed callbacks to monitor and control the training process effectively. The following callbacks were used for all models:

- `EarlyStopping`: Monitors the validation loss and stops training if no improvement is observed after a certain number of epochs (patience). The model's best weights are then restored.
- `ReduceLROnPlateau`: Reduces the learning rate when the validation loss plateaus, allowing the model to converge more effectively.

3.2 Metrics

Some custom metrics are defined to take into account the fact that some classes are non-informative for evaluating a POS tagging model. We decided to ignore the following classes:

- `:`: Colon
- `#`: Pound sign or hash symbol
- `$`: Dollar sign
- `-LRB-`: Left round bracket (opening parenthesis)
- `-RRB-`: Right round bracket (closing parenthesis)
- `,`: Comma
- `.`: Period or full stop
- `“`: Opening single quotation mark or backtick
- `”`: Closing single quotation mark
- `SYM`: Symbol
- `LS`: List item marker
- `-PAD-`: Padding token

The custom metrics defined include a masked F1 macro score, as outlined, which is utilized for evaluating the models post-training. Due to the inability to continuously monitor this metric with TensorFlow, we decided to define a custom accuracy

metric as well. This custom accuracy metric ignores certain classes and assigns different weights to errors on different classes. This was done to counteract the imbalance in the dataset, as normal accuracy was yielding excessively high values.

The accuracy metric works by first computing the per-sample accuracy, a binary tensor indicating if the prediction for each sample is corrected or not, then multiplying it with the weights for the corresponding true class to obtain a weighted per-sample accuracy. After it creates a binary mask which indicates what samples to ignore in the computation of the accuracy, this mask is initialized as all ones and then updated to exclude samples with class labels specified in the arguments. Finally, the overall weighted accuracy is computed by summing the weighted per-sample accuracy and dividing it by the number of non-ignored samples.

3.3 Results

To validate the results, we conducted training with three different seeds and then averaged the performances to establish which model is the best one.

Table 1: Average F1 Scores on Validation Set

	Macro F1 Scores
Baseline Model	0.7891
Additional LSTM	0.7966
Additional Dense	0.7992

Table 2: Average F1 Scores on Test Set

	Macro F1 Scores
Baseline Model	0.7810
Additional LSTM	0.7836
Additional Dense	0.7851

4 Discussion

From the results obtained, we can observe that although the performances of the two deeper models are similar, the one with two Dense layers seems to perform slightly better on average. However, the outline requests to perform the error analysis over the best single model, and that was one of the LSTM ones, which obtained a macro F1-score of 0.7939. This observation suggests that LSTM layers may have effectively captured sequential dependencies and long-range relationships within the input data, which are particularly crucial for POS tagging tasks.

4.1 Error analysis

We observed a higher number of errors on the validation set than on the test set. This may be due to the fact that the particular samples chosen for the validation set are more challenging or contain more edge cases, leading to a higher error rate.

We noticed the congruence of four classes among the top-5 mismatched tags for each model: `NN`, `NNP`, `JJ`, and `NNS`. The larger presence of these classes in the validation set, along with `VBN` tags, explains the higher number of errors.

4.1.1 Nouns

- `NN`: Noun, singular or mass
- `NNP`: Proper noun, singular
- `JJ`: Adjective
- `NNS`: Noun, plural

The number of errors made among `NN`, `NNP`, `JJ`, and `NNS` represents almost half of the mistakes made by the models, so we further investigated. We hypothesized that one potential reason for the frequent mistakes between these classes could be their semantic and syntactic similarities, leading to confusion for the model. For instance:

- `NN` (Noun, singular or mass) and `NNS` (Noun, plural): Both represent nouns but differ in number (singular vs. plural). However, certain words might function as both singular and plural forms, contributing to misclassifications.
- `NN` (Noun, singular or mass) and `NNP` (Proper noun, singular): While `NNP` represents singular proper nouns, `NN` encompasses singular common nouns. Proper nouns often refer to specific entities, which might overlap with common nouns, resulting in misclassifications.
- `NN` (Noun, singular or mass) and `JJ` (Adjective): Adjectives often modify nouns, and distinguishing between them can be challenging, especially when dealing with descriptive language.

4.1.2 Verbs

We then focused on `VBN`, which is also in the top 5 mistaken classes; many errors occur due to confusion between `VBN` and an `JJ`, so we investigated

over it and tags representing different forms or aspects of verbs.

- `VB`: Verb, base form
- `VBD`: Verb, past tense
- `VBG`: Verb, gerund or present participle
- `VBP`: Verb, non-3rd person singular present
- `VBZ`: Verb, 3rd person singular present
- `VBN`: Verb, past participle
- `JJ`: Adjective

It turned out that past participle verbs tend to be often mistaken for past tense verbs or for adjectives.

- `VBN` (Verb, past participle) and `VBD` (Verb, past tense): Past participle verbs and past tense verbs can have similar forms, especially in irregular verbs where the past tense and past participle forms are identical (e.g., "broken"). Furthermore, some past participles and past tense forms may have overlapping semantic meanings, further complicating classification. For example, "finished" can be both a past tense verb ("He finished the book") and a past participle verb/adjective ("the finished product").
- `VBN` (Verb, past participle) and `JJ` (Adjective): Past participle verbs can also function as adjectives in certain contexts. For example, "the broken window" uses "broken" as an adjective modifying "window."

4.1.3 Prepositions

The last class that is highly mistaken both in the validation and in the test set is the `IN` (Preposition or subordinating conjunction) tag. This tag is often confused with `RP` and `RB`.

- `IN`: Preposition or subordinating conjunction
- `RP`: Particle
- `RB`: Adverb

These errors likely arise from the fact that prepositions and subordinating conjunctions can both function as particles or adverbs, depending on the context of the sentence.

5 Conclusion

Surprisingly, we found that the simpler Baseline model performances are worse by a marginal margin of approximately $\approx 1\%$. This suggests that the additional layers in the deeper models did not significantly improve performance. One possible explanation for this is that the complexity introduced by the extra layers did not provide substantial benefits for the POS tagging task. It's possible that the Baseline model's simpler architecture was already effective in capturing the necessary patterns in the data, leading to comparable performance despite its simplicity.

6 Links to external resources

THIS SECTION IS OPTIONAL

Insert here:

- a link to your GitHub or any other public repo where one can find your code (only if you did not submit your code on Virtuale);
- a link to your dataset (only for non-standard projects or project works).

DO NOT INSERT CODE IN THIS REPORT

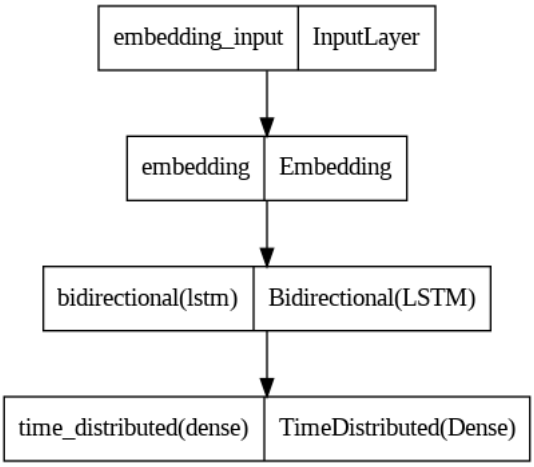

Figure 1: Baseline architecture

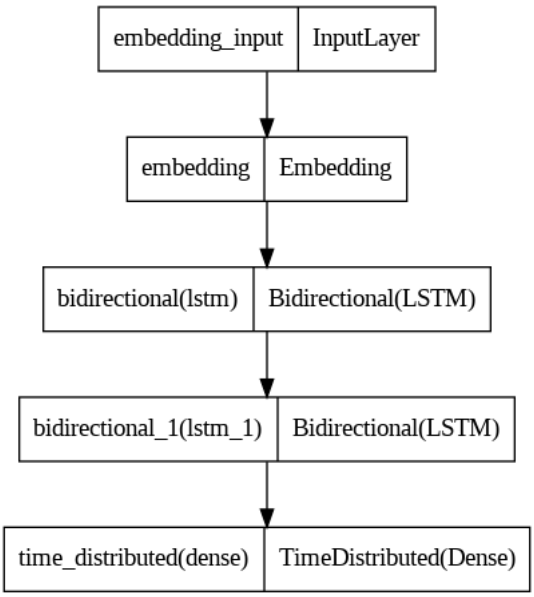

Figure 2: Additional LSTM architecture

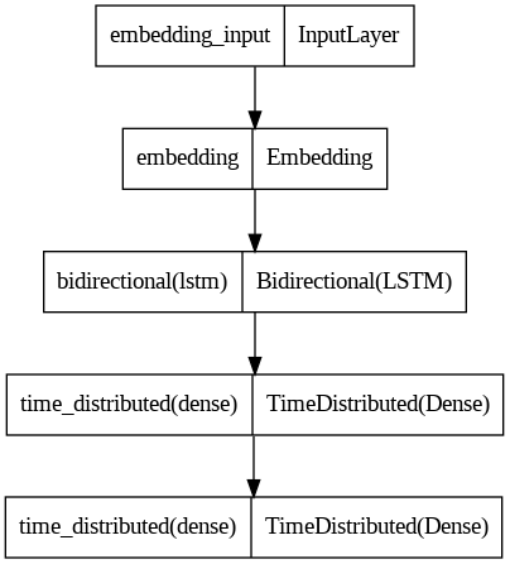

Figure 3: Additional Dense architecture